

# Bug Bounty Tips and Tricks using CHATGPT #1

Joas Antonio dos Santos

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

# Amass



list IP Address, domains and subdomains using amass with brute force

This content may violate our [content policy](#). If you believe this to be in error, please [submit your feedback](#) — your input will aid our research in this area.



To use brute force enumeration with Amass to list IP addresses, domains, and subdomains, you can use the `-brute` flag and specify a wordlist to use for the brute force search. For example:

Copy code

```
amass intel -brute -w /path/to/wordlist.txt -d example.com
```

This will use the wordlist at `/path/to/wordlist.txt` to perform a brute force search for subdomains of `example.com`. It will then list any IP addresses, domains, and subdomains that are found.

You can also use the `-min-for-brute` flag to specify the minimum subdomain length for which Amass will use the brute force enumeration method. For example:

Copy code

```
amass intel -brute -min-for-brute 3 -w /path/to/wordlist.txt -d example.com
```

This will only use brute force enumeration for subdomains of length 3 or greater.

# Amass + Nuclei: Finding domain

search domain using amass + nuclei

You can use Amass and Nuclei together to search for a domain and perform vulnerability scanning on the domain's subdomains. Here is an example of how to do this:

1. First, use Amass to gather intelligence on the domain and list its subdomains:

```
amass intel -d example.com
```

Copy code

2. Save the list of subdomains to a file, for example `subdomains.txt`.
3. Use Nuclei to scan the subdomains for vulnerabilities. For example, to scan for subdomains that are vulnerable to subdomain takeover attacks:

```
nuclei -t templates/subdomain-takeover.yaml -l subdomains.txt
```

Copy code

This will scan the subdomains listed in `subdomains.txt` for subdomain takeover vulnerabilities using the template in `templates/subdomain-takeover.yaml`.


# Amass + Nuclei 2: Finding domain



create script to search domain using amass + nuclei



Here is an example script that you can use to search for a domain using Amass and Nuclei:

 Copy code

```
#!/bin/bash

# Set the domain to search
domain="example.com"

# Use Amass to gather intelligence on the domain and list its subdomains
amass intel -d "$domain" > subdomains.txt

# Use Nuclei to scan the subdomains for vulnerabilities
nuclei -t templates/subdomain-takeover.yaml -l subdomains.txt
```

This script will gather intelligence on the domain `example.com` using Amass, and then save the list of subdomains to a file called `subdomains.txt`. It will then use Nuclei to scan the subdomains for subdomain takeover vulnerabilities using the template in `templates/subdomain-takeover.yaml`.

# Finding subdomains with jsubfinder + httpx




## Finding subdomains with jsubfinder + httpx



To find subdomains using jsubfinder and httpx, you can use the following steps:

1. First, use jsubfinder to gather subdomains for the target domain:


```
jsubfinder -d example.com
```

 Copy code

This will gather subdomains for the domain `example.com` using jsubfinder's default sources.

2. Save the list of subdomains to a file, for example `subdomains.txt`.
3. Use httpx to perform HTTP requests to the subdomains and check if they are available:

```
httpx -follow-redirects -threads 100 -timeout 5s -c 200 -silent -o results.txt -l subdomains.txt
```

 Copy code

This will send HTTP requests to the subdomains listed in `subdomains.txt` using a maximum of 100 threads. It will follow redirects, and consider a subdomain to be available if it returns a status code of 200. The results will be saved to a file called `results.txt`.

# Using FFUF to finding RCE



using ffuf to finding rce



To use ffuf to find Remote Code Execution (RCE) vulnerabilities, you can use the following steps:



1. First, create a list of payloads that you want to use to test for RCE. These payloads could include commands or scripts that, if successfully executed, would indicate that RCE is possible.
2. Use ffuf to send HTTP requests to the target URL with the payloads in the request body or as part of the URL query string. For example:

Copy code

```
ffuf -w payloads.txt -u http://example.com/path?q=FUZZ
```

This will send HTTP GET requests to `http://example.com/path?q=FUZZ`, replacing `FUZZ` with each payload in the `payloads.txt` file.

3. Use the `-mc` flag to specify the status codes that ffuf should consider a successful response. For example:

3. Use the `-mc` flag to specify the status codes that ffuf should consider a successful response. For example:

Copy code

```
ffuf -w payloads.txt -u http://example.com/path?q=FUZZ -mc 200,204,301,302
```

This will consider a status code of 200, 204, 301, or 302 to be a successful response.

4. Use the `-cs` flag to specify the content strings that ffuf should look for in the response body to indicate a successful payload. For example:

Copy code

```
ffuf -w payloads.txt -u http://example.com/path?q=FUZZ -mc 200,204,301,302 -cs "command executed successfully"
```

This will look for the string "command executed successfully" in the response body to determine if a payload was successful.

You can find more information and additional options in the ffuf documentation:

<https://github.com/ffuf/ffuf/blob/master/README.md>

# Full Account Takeover Technique in API/Register



## Full Account Takeover Technique in API/Register



A Full Account Takeover (FATO) technique in an API or registration system involves compromising a user's account in such a way that the attacker has complete control over it. This could involve stealing the user's login credentials, or bypassing the login process altogether.



Here are some potential steps that an attacker might take to perform a FATO attack on an API or registration system:

1. Gather information about the target API or registration system, such as the type of authentication it uses, the data it requires for registration, and any vulnerabilities that have been identified in the past.
2. Attempt to intercept user login credentials by setting up a phishing campaign or using a Man-in-the-Middle (MITM) attack.
3. Try to bypass the login process by exploiting known vulnerabilities in the API or registration system. For example, an attacker might try SQL injection attacks, cross-site scripting (XSS) attacks, or other types of injection attacks.
4. Once the attacker has gained access to a user's account, they might use it to perform actions on behalf of the user, such as transferring money, making unauthorized purchases, or accessing sensitive information.

To prevent FATO attacks, it is important to use secure authentication methods, such as multi-factor authentication, and to regularly patch any vulnerabilities that are discovered in the API or registration system. It is also a good idea to educate users about the importance of protecting their login credentials and to implement measures to detect and prevent phishing and MITM attacks.


# Oneliner Search JS domain using subjs, anew and httpx



Oneliner Search JS domain using subjs, anew and httpx



To search for JavaScript domains using subjs, anew, and httpx in a single command, you can use the following one-liner:

 Copy code

```
subjs -silent -threads 10 -source anew example.com | anew | httpx -silent -  
threads 10 -status-code 200
```

This will use subjs to search for JavaScript domains using the `anew` source, and then pass the results to anew to resolve the domains. The resolved domains will then be passed to httpx, which will send HTTP GET requests to the domains and only show the ones that return a status code of 200.

The `-silent` flag is used to suppress output from subjs and httpx, and the `-threads` flag is used to specify the number of threads that each tool should use.



# Shodan queries to search Scada, IoT, Router Devices



Here are some Shodan queries that you can use to search for SCADA, IoT, and router devices:



- To search for SCADA devices:
  - ``product:"Supervisory Control and Data Acquisition"```
  - ``product:"Human-Machine Interface"```
- To search for IoT devices:
  - ``product:"Internet of Things"```
  - ``product:"IoT Gateway"```
  - ``product:"Smart Meter"```
  - ``product:"Smart Grid"```
- To search for router devices:
  - ``product:"Router"```
  - ``product:"Wireless Router"```
  - ``product:"Modem Router"```

You can use these queries as-is, or combine them with other filters to narrow down the search results. For example, you could use the ``country`` filter to only show devices from a specific country, or the ``port`` filter to only show devices listening on a specific port.

# Screenshot page using aquatone using domain files




Screenshot page using aquatone using domain files



To use Aquatone to take screenshots of a webpage using a list of domains, you can use the following command:



```
aquatone-scan -chrome-path /path/to/chrome -out folder -threads 10 -domain-list domains.txt
```

 Copy code

This will use Aquatone to take screenshots of the webpages for each domain in the `domains.txt` file using Google Chrome, and save the screenshots to the `folder` directory. The `-threads` flag is used to specify the number of concurrent threads to use, and the `-chrome-path` flag is used to specify the path to the Chrome executable.

Aquatone will automatically create subdirectories for each domain and save the screenshots in those directories.


# oneliner using shodan and nuclei to scanning hosts



oneliner using shodan and nuclei to scanning hosts



Here is an example one-liner that you can use to scan hosts using Shodan and Nuclei:

 Copy code

```
shodan search --fields ip_str product organization country --separator ,  
"product:Apache" | awk -F, '{print $1}' | nuclei -t templates/http-vulns.yaml -l  
-
```


This command will use Shodan to search for Apache servers, extract the IP address of each result, and pass the IP addresses to Nuclei. Nuclei will then scan the hosts for HTTP vulnerabilities using the template in `templates/http-vulns.yaml`.

The `--fields` and `--separator` flags are used to specify which fields to include in the Shodan search results and how to separate them, and the `awk` command is used to extract the IP addresses from the results. The `-l -` flag tells Nuclei to read the list of targets from stdin.

# Oneliner finding subdomain using gospider, assetfinder, amass and nuclei



Here is an example one-liner that you can use to find subdomains using gospider, assetfinder, amass, and nuclei:

 Copy code

```
gospider -s example.com | assetfinder | sort -u | amass intel -d example.com |  
nuclei -t templates/subdomain-takeover.yaml -l -
```

This command will use gospider to gather subdomains for the domain `example.com`, pass the results to assetfinder to find additional subdomains, sort and deduplicate the results, pass them to amass to gather intelligence on the domain, and then pass the results to nuclei to scan for subdomain takeover vulnerabilities.

The `-s` flag is used to specify the seed URL for gospider, and the `-l -` flag tells nuclei to read the list of targets from stdin.


# Oneliner portscan and subdomain discovery using subfinder, cf-check, naabu and httpprobe



oneliner portscan and subdomain discovery using subfinder, cf-check, naabu and httpprobe



Here is an example one-liner that you can use to perform port scanning and subdomain discovery using subfinder, cf-check, naabu, and httpprobe:

 Copy code

```
subfinder -d example.com | cf-check | naabu -silent -ports top -targets stdin |  
httpprobe -p 80,443 | sort -u
```

This command will use subfinder to find subdomains for the domain `example.com`, pass the results to cf-check to check for Cloudflare protection, pass the results to naabu to perform port scanning on the top 1,000 ports, pass the results to httpprobe to check for HTTP servers on ports 80 and 443, and finally sort and deduplicate the results.

The `-silent` flag is used to suppress output from naabu, and the `-ports` flag is used to specify the ports to scan. The `-targets stdin` flag tells naabu to read the list of targets from stdin, and the `-p` flag is used to specify the ports that httpprobe should check.


# Oneliner search SSRF using subfinder, httpx and qsreplace



oneliner search SSRF using subfinder, httpx and qsreplace



Here is an example one-liner that you can use to search for Server-Side Request Forgery (SSRF) vulnerabilities using subfinder, httpx, and qsreplace:

 Copy code


```
subfinder -d example.com | qsreplace 'http://localhost' 'FUZZ' | httpx -silent -  
threads 10 -status-code 200
```




This command will use subfinder to find subdomains for the domain `example.com`, pass the results to qsreplace to replace `http://localhost` with `FUZZ`, and then pass the results to httpx to send HTTP GET requests to the subdomains and only show the ones that return a status code of 200.

The `-silent` flag is used to suppress output from httpx, and the `-threads` flag is used to specify the number of concurrent threads to use.

To test for SSRF vulnerabilities, you can replace `http://localhost` with a URL that points to an internal resource that should not be accessible from the outside. If the server is vulnerable to SSRF, it may allow the attacker to access the internal resource.

# Oneliner recon domain and subdomains using chaos, gospider, findomain, assetfinder, amass, httpx and anew

 oneliner recon domain and subdomains using chaos, gospider, findomain, assetfinder, amass, httpx and anew

 Here is an example one-liner that you can use to perform reconnaissance on a domain and its subdomains using chaos, gospider, findomain, assetfinder, amass, httpx, and anew:  



```
chaos -d example.com | gospider -s http://example.com | findomain -t example.com | assetfinder | amass intel -d example.com | httpx -silent -threads 10 -status-code 200 | anew | sort -u
```

This command will use chaos to gather subdomains for the domain `example.com`, pass the results to gospider to find additional subdomains, pass the results to findomain to find more subdomains, pass the results to assetfinder to find even more subdomains, pass the results to amass to gather intelligence on the domain, pass the results to httpx to send HTTP GET requests to the subdomains and only show the ones that return a status code of 200, and finally pass the results to anew to resolve the domains. The results will be sorted and deduplicated.

The `-s` flag is used to specify the seed URL for gospider, the `-t` flag is used to specify the target domain for findomain, and the `-silent` flag is used to suppress output from httpx. The `-threads` flag is used to specify the number of concurrent threads to use.

# Oneliner search xss using kxss, xargs and httpx

oneliner search xss using kxss, xargs and httpx

Here is an example one-liner that you can use to search for cross-site scripting (XSS) vulnerabilities using kxss, xargs, and httpx:  

```
kxss example.com | xargs -I % sh -c "echo '<script>alert(1)</script>' | httpx -silent -body-string @- -status-code 200 %"
```

This command will use kxss to find potentially vulnerable pages on the domain `example.com`, and pass the results to xargs. xargs will then send an HTTP POST request to each page with a simple XSS payload in the request body, and pass the response to httpx. httpx will only show the pages that return a status code of 200, indicating that the payload was successful.

The `-I %` flag is used to specify the placeholder for the input, and the `-c` flag is used to specify the command to run. The `-silent` flag is used to suppress output from httpx, and the `-body-string` flag is used to specify the request body.

To test for XSS vulnerabilities, you can use a variety of payloads to try to inject malicious JavaScript into the page. If the server is vulnerable to XSS, it may execute the injected code and allow the attacker to perform actions on behalf of the user.



# Google dork to discovery api exposure



Here are some Google dorks that you can use to discover API exposure:



- ``intext:"swagger" OR intext:"openapi" OR intext:"oas"``` - This dork searches for pages that contain the terms "swagger," "openapi," or "oas," which are commonly used to describe APIs that follow the OpenAPI specification.
- ``inurl:/api/ OR inurl:/docs/api/``` - This dork searches for pages with URLs that contain the substrings "/api/" or "/docs/api/," which are often used to indicate the presence of an API.
- ``intext:"api key" OR intext:"api_key" OR intext:"api-key"``` - This dork searches for pages that contain the terms "api key," "api\_key," or "api-key," which are often used to identify API keys in documentation or code examples.
- ``intext:"endpoint" OR intext:"api endpoint"``` - This dork searches for pages that contain the terms "endpoint" or "api endpoint," which are commonly used to describe the locations where API requests can be made.

You can use these dorks as-is, or combine them with other search terms to narrow down the results. For example, you could use the ``site:`` operator to search for APIs on a specific domain, or the ``filetype:`` operator to search for API documentation in a specific format (e.g. ``filetype:yaml``).

Keep in mind that Google dorks can return a large number of results, and not all of them will be relevant to your search. It is a good idea to carefully review the results and verify that they are relevant before proceeding.

# Create script to finder and test sql injection



Script to finder sql injection using subfinder, httpx, katana, tee, gf and sqlmap



Here is a script that you can use to find SQL injection vulnerabilities using subfinder, httpx, katana, tee, gf, and sqlmap:

Copy code

```
#!/bin/bash

# Find subdomains using subfinder
subfinder -d example.com -silent | tee domains.txt

# Send HTTP GET requests to the subdomains using httpx
httpx -silent -threads 10 -status-code 200 -input-file domains.txt | tee
urls.txt

# Use katana to extract URLs with query parameters
katana -input-file urls.txt -query-params | tee params.txt

# Use gf to search for patterns indicative of SQL injection vulnerabilities
gf sql params.txt | tee injections.txt

# Use sqlmap to test the identified URLs for SQL injection vulnerabilities
while read line; do
  sqlmap -u "$line" --batch --level 5 --risk 3
done < injections.txt
```

This script will use subfinder to find subdomains for the domain `example.com`, pass the results to httpx to send HTTP GET requests to the subdomains and extract the URLs, use katana to extract URLs with query parameters, use gf to search for patterns indicative of SQL injection vulnerabilities, and finally use sqlmap to test the identified URLs for SQL injection vulnerabilities.



The `-silent` flag is used to suppress output from subfinder and httpx, and the `-threads` flag is used to specify the number of concurrent threads to use in httpx. The `-input-file` flag is used to specify the input file for httpx and katana, and the `--batch` flag is used to run sqlmap in non-interactive mode. The `--level` and `--risk` flags are used to specify the level of tests to perform and the risk of false positives, respectively.


# Oneliner find xss using subfinder, httpx, katana, gxss, kxss and dalfox



Oneliner find xss using subfinder, httpx, katana, gxss, kxss and dalfox



Here is an example one-liner that you can use to find cross-site scripting (XSS) vulnerabilities using subfinder, httpx, katana, gxss, kxss, and dalfox:  

 Copy code

```
subfinder -d example.com | httpx -silent -threads 10 -status-code 200 | katana -  
query-params | gxss | kxss | dalfox pipe
```

This command will use subfinder to find subdomains for the domain `example.com`, pass the results to httpx to send HTTP GET requests to the subdomains and extract the URLs, use katana to extract URLs with query parameters, pass the results to gxss to find XSS vulnerabilities, pass the results to kxss to find additional XSS vulnerabilities, and finally pass the results to dalfox to find even more XSS vulnerabilities.

The `-silent` flag is used to suppress output from httpx, and the `-threads` flag is used to specify the number of concurrent threads to use. The `-query-params` flag is used to extract URLs with query parameters, and the `pipe` flag is used to tell dalfox to read the list of targets from stdin.

# Domain enumeration and discovery files using ffuf, httpx and findomain



You can use the following command to perform domain enumeration and discover files using `ffuf`, `httpx` and `findomain`:

```
ffuf -w /path/to/wordlist -u https://target/FUZZ -e .html,.php -mc all -mc 200,204,301,302,307 -o output.txt -of json -sc 200 -t 100 -timeout 20s

httpx -title -follow-redirects -silent -content-length -threads 100 -timeout 20s -retries 3 -o output.txt -mc all -mc 200,204,301,302,307 -wl /path/to/wordlist -ac -acme-dns -acme-dns-timeout 10s -acme-dns-challenge dns-01 -acme-dns-credentials "/path/to/credentials.json" https://target/

findomain -t target -q
```

This will use the wordlist at `/path/to/wordlist` to discover files on the target site `https://target/`. The `-e` flag specifies the file extensions to consider (in this case, `.html` and `.php`). The `-mc` flag specifies the HTTP status codes that should be considered as valid responses. The `-o` flag specifies the output file, and the `-of` flag specifies the output format (in this case, JSON). The `-sc` flag specifies the minimum HTTP status code to consider a valid response. The `-t` flag specifies the number of concurrent requests to make. The `-timeout` flag specifies the maximum time to wait for a response.

The `httpx` command is similar, with the `-title` flag causing it to display the title of each page it discovers, the `-follow-redirects` flag causing it to follow redirects, the `-silent` flag causing it to suppress progress output, the `-content-length` flag causing it to display the content length of each response, the `-threads` flag specifying the number of concurrent requests to make, the `-timeout` flag specifying the maximum time to wait for a response, the `-retries` flag specifying the number of times to retry failed requests, the `-o` flag specifying the output file, the `-mc` flag specifying the HTTP status codes that should be considered as valid responses, the `-wl` flag specifying the wordlist to use, the `-ac` flag enabling automatic certificate management, the `-acme-dns` flag specifying the ACME DNS challenge to use, the `-acme-dns-timeout` flag specifying the timeout for the ACME DNS challenge, the `-acme-dns-challenge` flag specifying the type of ACME challenge to use, and the `-acme-dns-credentials` flag specifying the path to the ACME credentials file.

The `findomain` command will discover subdomains for the target domain `target`. The `-t` flag specifies the target domain, and the `-q` flag tells `findomain` to be quiet (i.e., not to print any output).

# oneliner find open redirect using waybackurls, httpx, gf, anew and nuclei



oneliner find open redirect using waybackurls, httpx, gf, anew and nuclei



You can use the following command to find open redirect vulnerabilities using `waybackurls`, `httpx`, `gf`, `anew` and `nuclei`:

Copy code

```
waybackurls target.com | httpx -silent | grep "Location" | anew | nuclei -t  
/path/to/open-redirect-templates -silent  
  
gf openredirect target.com | anew | nuclei -t /path/to/open-redirect-templates -  
silent
```

The first command uses `waybackurls` to discover URLs for the target domain `target.com`, pipes the output to `httpx`, which makes HTTP requests for each URL and displays the response headers. The `grep` command filters the output for lines containing the string "Location", and pipes the output to `anew`, which removes any duplicates. The final piped command, `nuclei`, uses the specified templates at `/path/to/open-redirect-templates` to detect open redirect vulnerabilities in the input. The `-silent` flag tells `nuclei` to suppress progress output.

The second command uses `gf` to search for open redirect vulnerabilities in the target domain `target.com`. The output is piped to `anew`, which removes any duplicates, and then to `nuclei`, which uses the specified templates at `/path/to/open-redirect-templates` to detect open redirect vulnerabilities in the input. The `-silent` flag tells `nuclei` to suppress progress output.

Note that this is just one way to find open redirect vulnerabilities, and there may be other tools and techniques that you can use as well.

# Oneliner complete enumeration xss, lfi, ssrf in domain using gauplus, anew, gxss, gf, qsreplace, httpx and SecretFinder

oneliner complete enumeration xss, lfi, ssrf in domain using gauplus, anew, gxss, gf, qsreplace, httpx and SecretFinder

You can use the following command to perform a complete domain enumeration for cross-site scripting (XSS), local file inclusion (LFI), and server-side request forgery (SSRF) vulnerabilities using `gau`, `plus`, `anew`, `gxss`, `gf`, `qsreplace`, `httpx` and `SecretFinder`:

```
gau target.com | plus | anew | gxss -xss | gf -lfi -ssrf | qsreplace -r target.com -s target.com | httpx -silent | SecretFinder -silent
```

This command uses `gau` to discover subdomains for the target domain `target.com`, pipes the output to `plus`, which performs additional subdomain discovery using various techniques. The output is then piped to `anew`, which removes any duplicates, and then to `gxss`, which searches for cross-site scripting vulnerabilities using the `-xss` flag. The output is then piped to `gf`, which searches for local file inclusion (LFI) and server-side request forgery (SSRF) vulnerabilities using the `-lfi` and `-ssrf` flags. The `qsreplace` command replaces all instances of the target domain `target.com` in the input with itself, to ensure that all discovered URLs are in the correct format. The output is then piped to `httpx`, which makes HTTP requests for each URL and displays the response headers. Finally, the output is piped to `SecretFinder`, which searches for secrets (e.g., API keys, passwords) in the input. The `-silent` flag tells `httpx` and `SecretFinder` to suppress progress output.

This is just one possible approach to performing a complete domain enumeration for XSS, LFI, and SSRF vulnerabilities, and there may be other tools and techniques that you can use as well.

# Oneliner check cloudflare using subfinder, dnsx, cf-check, naabu



oneliner check cloudflare using subfinder, dnsx, cf-check, naabu



You can use the following command to check if a domain is protected by Cloudflare using `subfinder`, `dnsx`, `cf-check`, and `naabu`:



Copy code

```
subfinder -d target.com -silent | dnsx -silent | cf-check | naabu -silent
```

This command uses `subfinder` to discover subdomains for the target domain `target.com`, pipes the output to `dnsx`, which performs DNS enumeration for each subdomain, and then pipes the output to `cf-check`, which checks if each subdomain is protected by Cloudflare. The output is then piped to `naabu`, which performs port scanning for each subdomain. The `-silent` flag tells each tool to suppress progress output.

This is just one possible approach to checking if a domain is protected by Cloudflare, and there may be other tools and techniques that you can use as well.

# Oneliner recon jira using uncover with shodan, censys and fofa and vulnerability scan with nuclei



oneliner recon jira using uncover with shodan, censys and fofa and vulnerability scan with nuclei



You can use the following command to perform recon on a Jira instance using `uncover` with `shodan`, `censys`, and `fofa`, and perform a vulnerability scan using `nuclei`:



Copy code



```
uncover jira -s shodan -s censys -s fofa | nuclei -t /path/to/jira-vulnerability-templates -silent
```




This command uses `uncover` to discover Jira instances using the specified sources (`shodan`, `censys`, and `fofa`). The output is then piped to `nuclei`, which uses the specified templates at `/path/to/jira-vulnerability-templates` to scan for vulnerabilities in the discovered Jira instances. The `-silent` flag tells `nuclei` to suppress progress output.

This is just one possible approach to performing recon and vulnerability scanning on a Jira instance, and there may be other tools and techniques that you can use as well.




# Oneliner recon subdomain using assetfinder, httpx, xargs, waybackurls and nuclei vulnerability scan

 oneliner recon subdomain using assetfinder, httpx, xargs, waybackurls and nuclei vulnerability scan 

 You can use the following command to perform recon on subdomains using `assetfinder`, `httpx`, `xargs`, `waybackurls`, and perform a vulnerability scan using `nuclei`:  

```
assetfinder target.com | httpx -silent | xargs -I {} sh -c 'waybackurls {} | nuclei -t /path/to/vulnerability-templates -silent'
```

 Copy code

This command uses `assetfinder` to discover subdomains for the target domain `target.com`, pipes the output to `httpx`, which makes HTTP requests for each subdomain and displays the response headers. The output is then piped to `xargs`, which runs the specified command (`waybackurls {} | nuclei -t /path/to/vulnerability-templates -silent`) for each subdomain. The `waybackurls` command discovers URLs for each subdomain, and the output is piped to `nuclei`, which uses the specified templates at `/path/to/vulnerability-templates` to scan for vulnerabilities. The `-silent` flag tells `nuclei` to suppress progress output.

This is just one possible approach to performing recon and vulnerability scanning on subdomains, and there may be other tools and techniques that you can use as well.

# Oneliner extract js using haktrails, httpx, getjs, anew, tojson



oneliner extract js using haktrails, httpx, getjs, anew, tojson



You can use the following command to extract JavaScript files using `haktrails`, `httpx`, `getjs`, `anew`, and `tojson`:



Copy code

```
haktrails target.com | httpx -title -follow-redirects -silent -content-length -  
threads 100 -timeout 20s -retries 3 -o output.txt -mc all -mc  
200,204,301,302,307 -wl /path/to/wordlist -ac -acme-dns -acme-dns-timeout 10s -  
acme-dns-challenge dns-01 -acme-dns-credentials "/path/to/credentials.json" |  
getjs -silent | anew | tojson
```

This command uses `haktrails` to discover URLs for the target domain `target.com`, pipes the output to `httpx`, which makes HTTP requests for each URL and displays the response headers. The `-title` flag causes it to display the title of each page, the `-follow-redirects` flag causes it to follow redirects, the `-silent` flag causes it to suppress progress output, the `-content-length` flag causes it to display the content length of each response, the `-threads` flag specifies the number of concurrent requests to make, the `-timeout` flag specifies the maximum time to wait for a response, the `-retries` flag specifies the number of times to retry failed requests, the `-o` flag specifies the output file, the `-mc` flag specifies the HTTP status codes that should be considered as valid responses, the `-wl` flag specifies the wordlist to use, the `-ac` flag enables automatic certificate management, the `-acme-dns` flag specifies the ACME DNS challenge to use, the `-acme-dns-timeout` flag specifies the timeout for the ACME DNS challenge, the `-acme-dns-challenge` flag specifies the type of AC

# Oneliner LFI using gau, gf, qsreplace and xargs



oneliner lfi using gau, gf, qsreplace and xargs



You can use the following command to find local file inclusion (LFI) vulnerabilities using `gau`, `gf`, `qsreplace`, and `xargs`:

Copy code

```
gau target.com | gf lfi | qsreplace -r target.com -s target.com | xargs -I {} sh -c 'curl -s {} | grep -i "root:x:0:0:root:/root:/bin/bash"'
```

This command uses `gau` to discover subdomains for the target domain `target.com`, pipes the output to `gf`, which searches for LFI vulnerabilities. The output is then piped to `qsreplace`, which replaces all instances of the target domain `target.com` in the input with itself, to ensure that all discovered URLs are in the correct format. The output is then piped to `xargs`, which runs the specified command (`curl -s {} | grep -i "root:x:0:0:root:/root:/bin/bash"`) for each URL. The `curl` command makes an HTTP request to each URL, and the `grep` command searches the response for the string `root:x:0:0:root:/root:/bin/bash`, which is indicative of an LFI vulnerability.

This is just one possible approach to finding LFI vulnerabilities, and there may be other tools and techniques that you can use as well. ■

# Extra Resource

- [https://twitter.com/ReconOne\\_bk](https://twitter.com/ReconOne_bk)
- <https://github.com/KingOfBugbounty/KingOfBugBountyTips>
- <https://github.com/dwisiswant0/awesome-oneliner-bugbounty>
- <https://thevillagehacker.gitbook.io/one-liners-for-bug-bounty-hunting>
- <https://github.com/djadmin/awesome-bug-bounty>
- <https://attacker-codeninja.github.io/2022-06-06-awesome-bug-bounty-roadmap/>
- <https://tikam02.github.io/Bug-Bounty-Resources/>