The Bug Hunter's Methodology Live

DAY 2 - APPLICATION ANALYSIS

INTENDED USAGE

THERE ARE MANY LIKE IT, BUT THIS IS MINE

Many people can teach:

```
';alert('xss');//
```

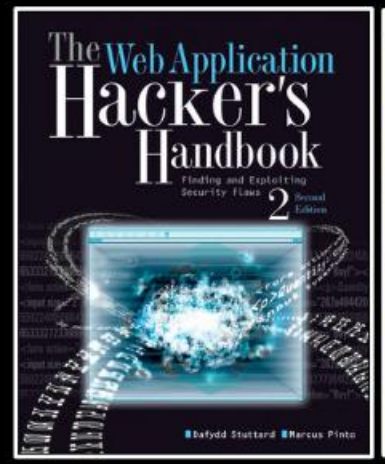
Less people teach where to look for web bugs.

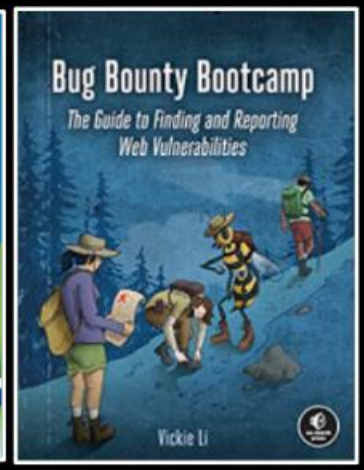
This course is a collection of my favorite tips, tools, and tricks.

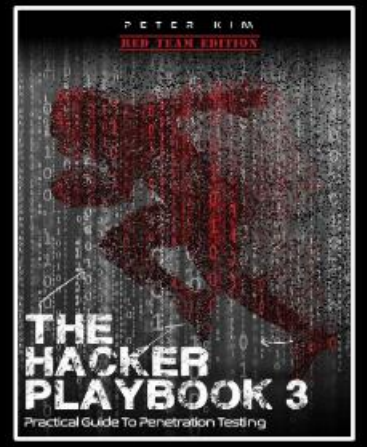
If you want a holistic course...

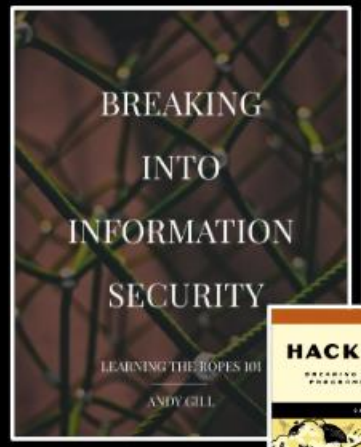
PRINT RESOURCES

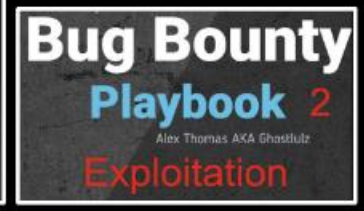
TBHM will attempt to give you tips, tricks, and tools related testing but there are many great wholistic texts available to supplement your app hacking journey.



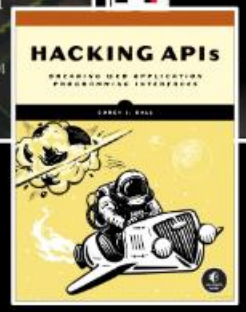


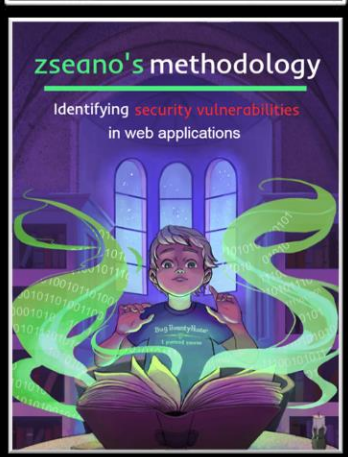






New! -->





HOLISTIC COURSES

Web Security Academy is by far the best training ground for an introduction to web application security.

It is completely free and provides text-based instructions on how to approach its labs.

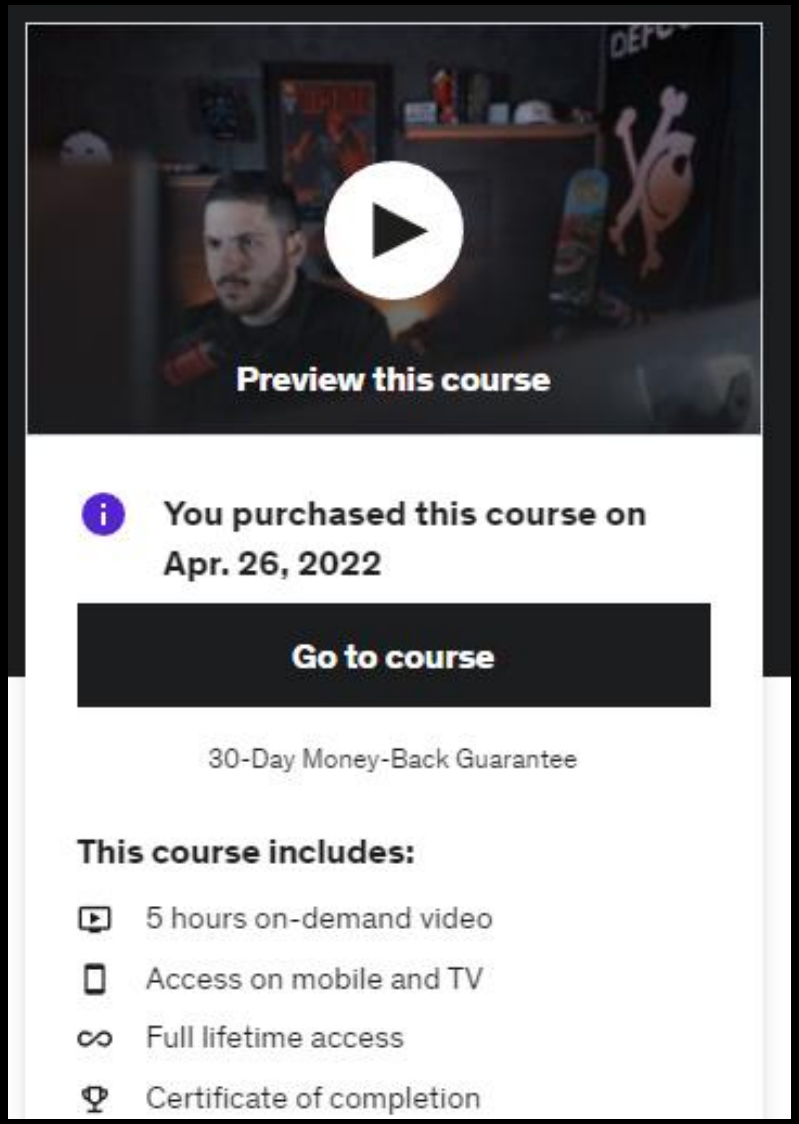
If you want supplemental guidance to this resource, check out Rana Khalil's free YouTube channel and then her very cheap course.

Web Security Academy

HOLISTIC COURSES

Ben aka Nahamsec also has an excellent course for introduction to web hacking and bug bounty hunting.

It is also relatively cheap.



PRACTICE TARGETS

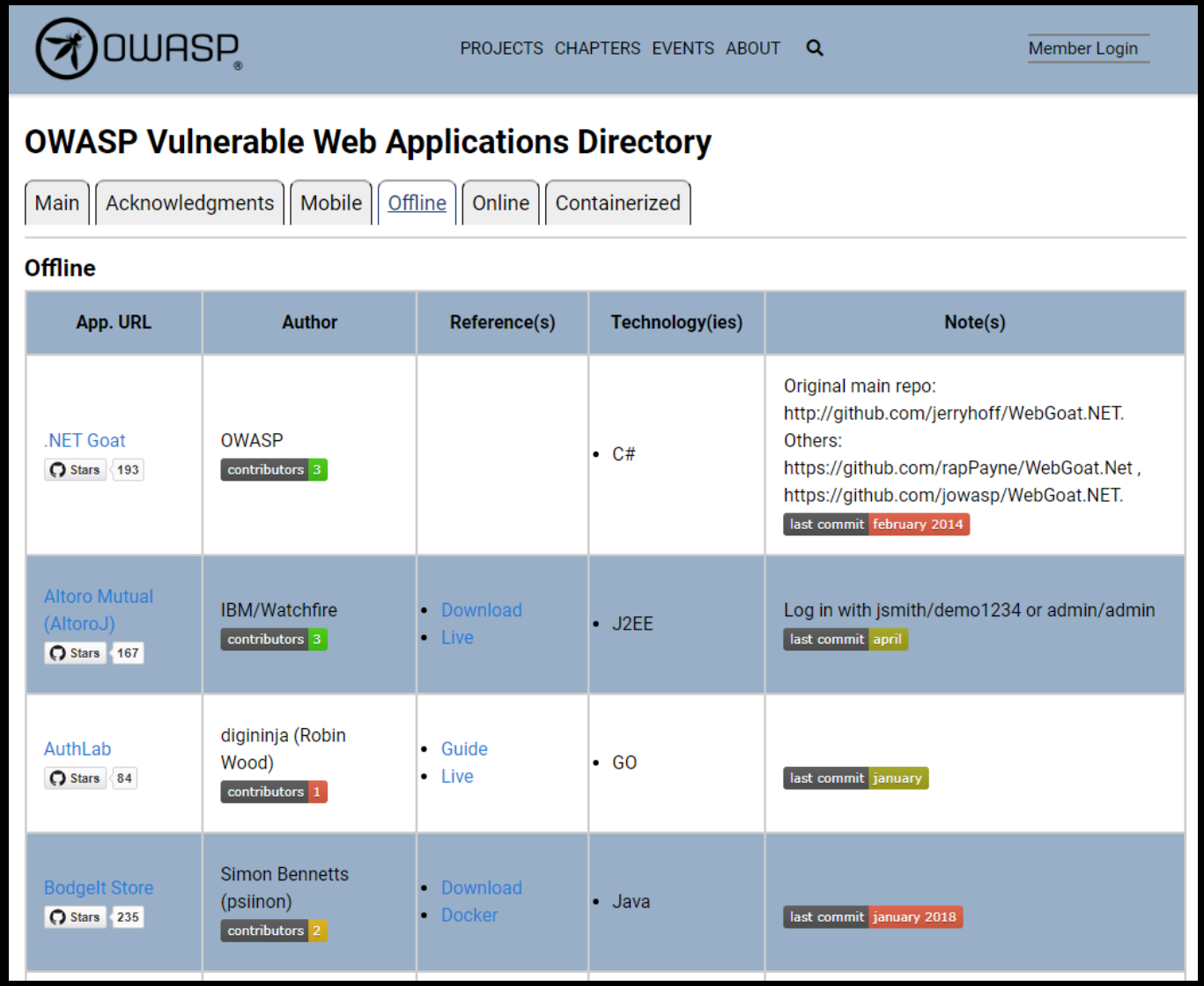
If you want additional labs to hack there exists several great resources on the web. Some are free and some are paid, but relatively cheap.

The first two PentesterLab and HackTheBox offer accompanying training to the topics whereas VulnHUB is a community of hackers uploading crackme challenges and hosting them for other people to download and solve.

PRACTICE TARGETS (OVWAD)

In addition, OWASP hosts a project called the Vulnerable Web Applications Directory which tries to keep up with all the practice applications that people make end up on GitHub.

These are mostly self hosted labs that you can run through at your own pace.



OWASP PROJECTS CHAPTERS EVENTS ABOUT Member Login

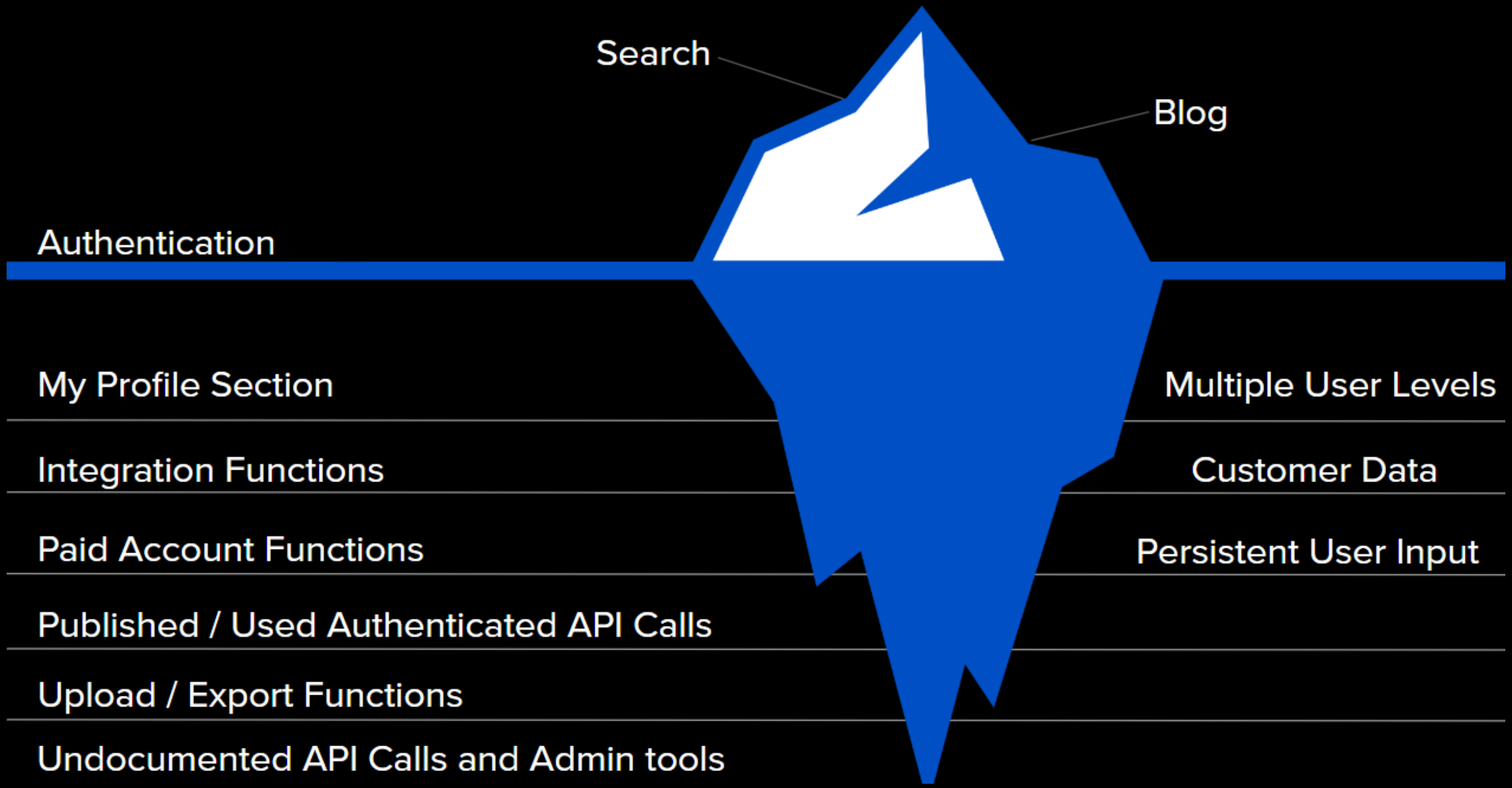
OWASP Vulnerable Web Applications Directory

Main | Acknowledgments | Mobile | Offline | Online | Containerized

Offline

App. URL	Author	Reference(s)	Technology(ies)	Note(s)
.NET Goat Stars 193	OWASP contributors 3		• C#	Original main repo: http://github.com/jerryhoff/WebGoat.NET. Others: https://github.com/rapPayne/WebGoat.Net , https://github.com/jowasp/WebGoat.NET. last commit february 2014
Altoro Mutual (AltoroJ) Stars 167	IBM/Watchfire contributors 3	• Download • Live	• J2EE	Log in with jsmith/demo1234 or admin/admin last commit april
AuthLab Stars 84	digininja (Robin Wood) contributors 1	• Guide • Live	• GO	last commit january
BodgeIt Store Stars 235	Simon Bennetts (psiinon) contributors 2	• Download • Docker	• Java	last commit january 2018

ANALYSIS CONCEPTS


Search
Blog
Authentication
My Profile Section
Integration Functions
Paid Account Functions
Published / Used Authenticated API Calls
Upload / Export Functions
Undocumented API Calls and Admin tools
Multiple User Levels
Customer Data
Persistent User Input

ANALYSIS CONCEPTS

I like to kick off the analysis course and talks discussing how I view applications holistically.

I usually end up approaching an application by breaking it down into its layers, profiling it, and then asking myself questions about its functionality.

I then prioritize different layers.

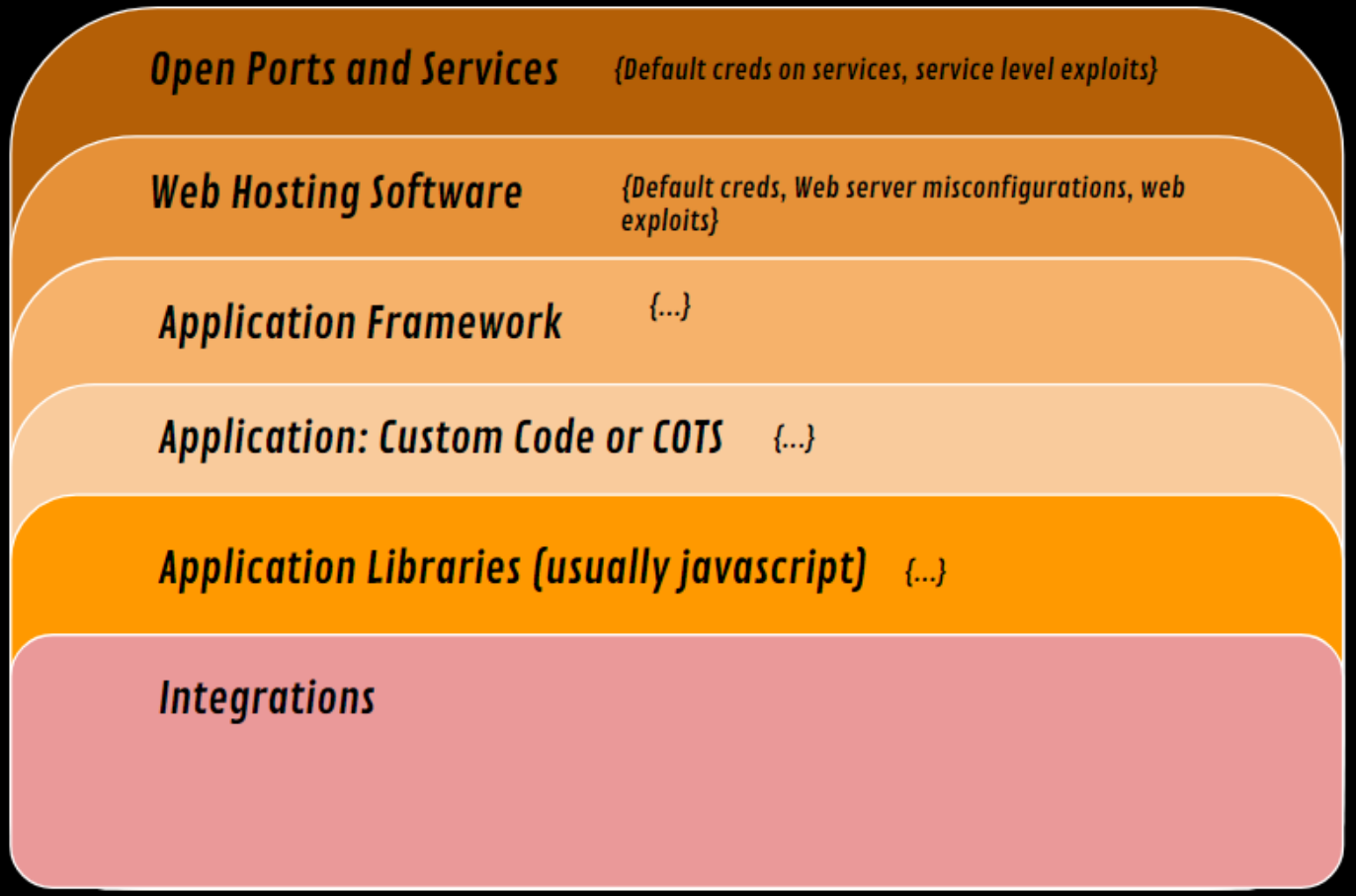
- Analysis Layers
- Application Layers as related to success.
- Tech Profiling
- The Big Questions

ANALYSIS CONCEPTS (LAYERS)

In general, when looking at an application that I am destined to hack I break it up into these layers.

While not quite the most scientific breakout, it helps me compartmentalize the different areas I might be targeting to hack.

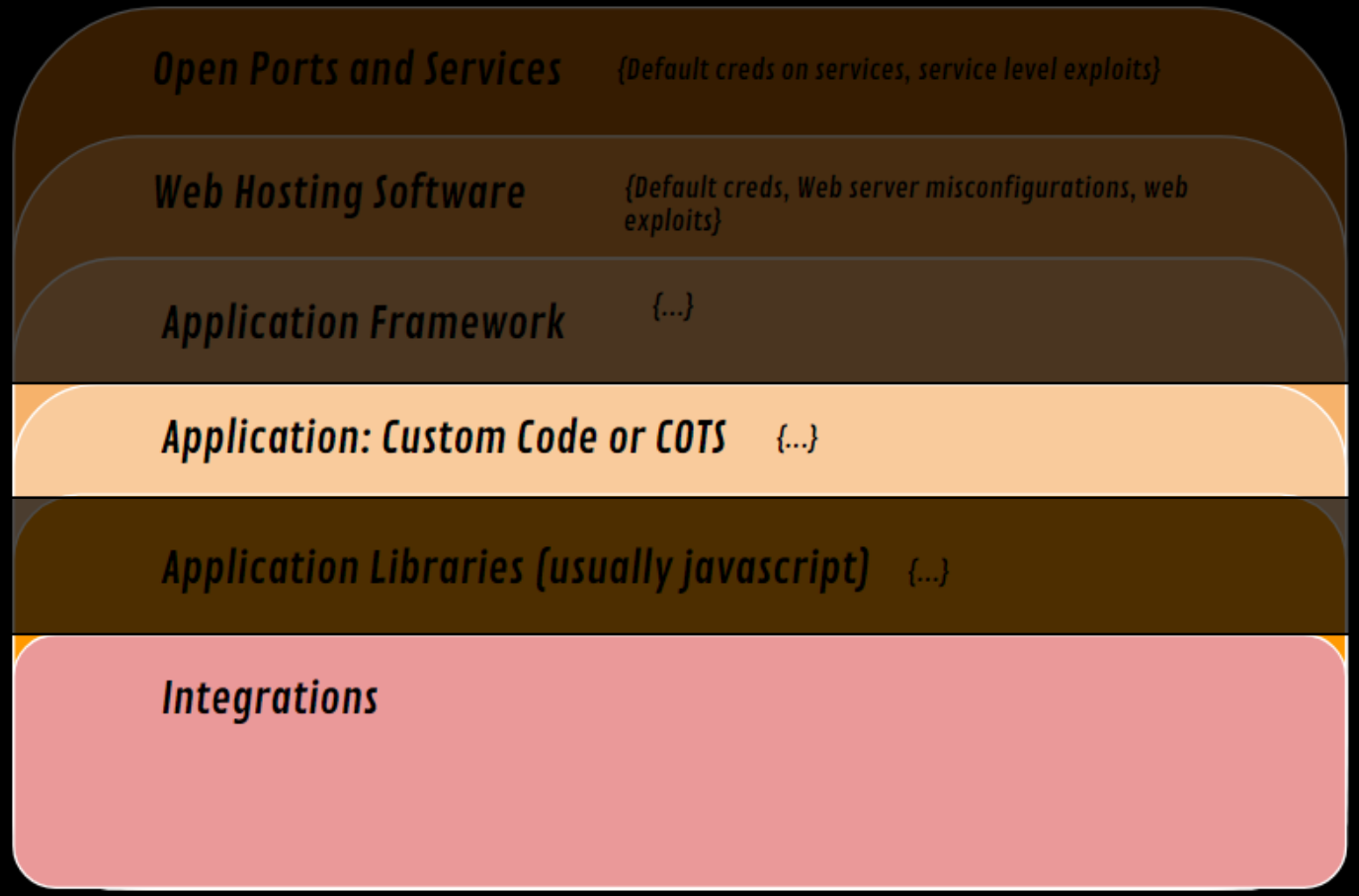
We'll jump into even more sections when we get into heat mapping in the later sections of the course but let's talk a little bit about them each here...



ANALYSIS CONCEPTS (SUCCESS)

In general, most of your hacking success when it comes to web applications will come at the custom code layer and the integrations layer.

Also unseen here is if there are stand alone APIs which are a very still vulnerable part of applications these days.



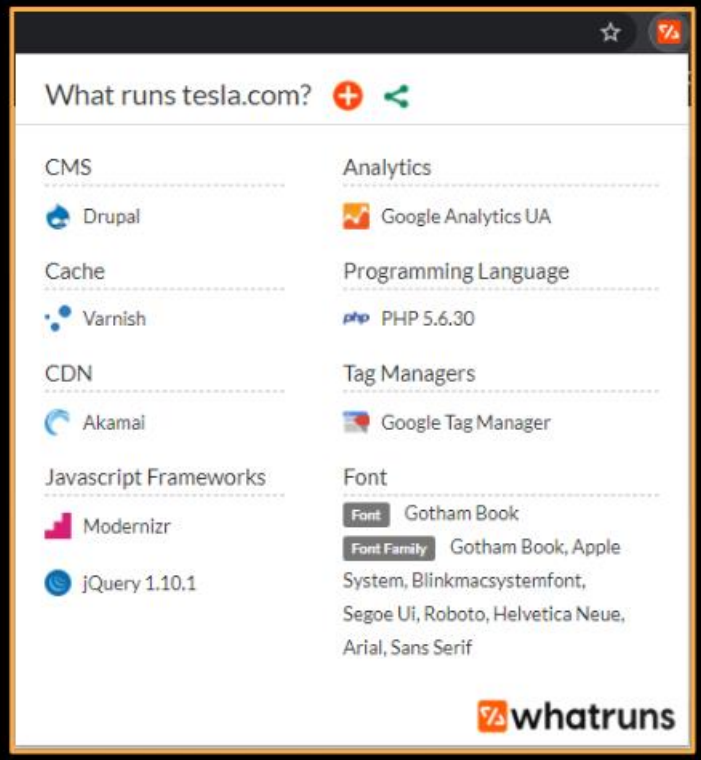
ANALYSIS CONCEPTS (TECH PROFILING)

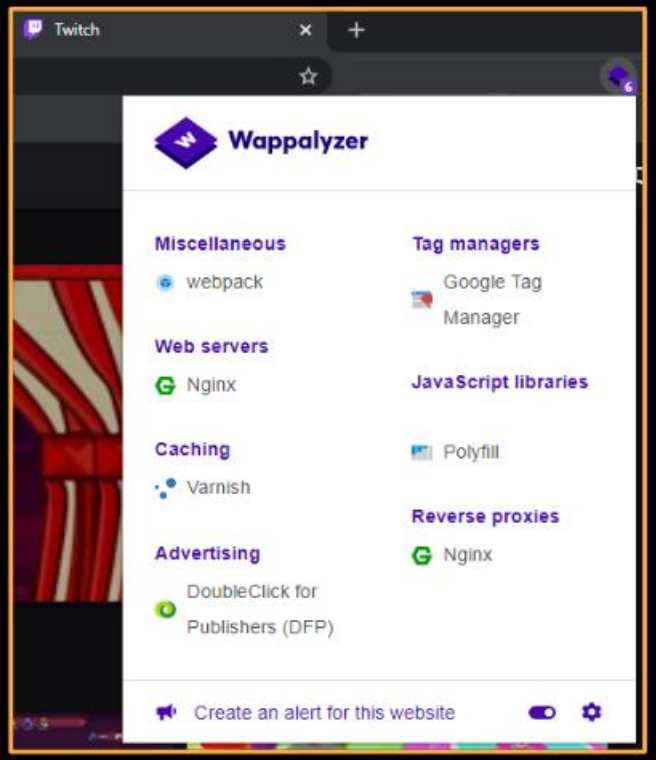
One of the first tasks we need to do when dealing with a application is to understand what web server it's running, on what programming language it's written in, and what frameworks it's using.

Browser extensions like Whatruns, Wappalyzer, and BuiltWith can give this information to us very quickly without spending too much time digging around and the HTML source.

whatruns

Wappalyzer

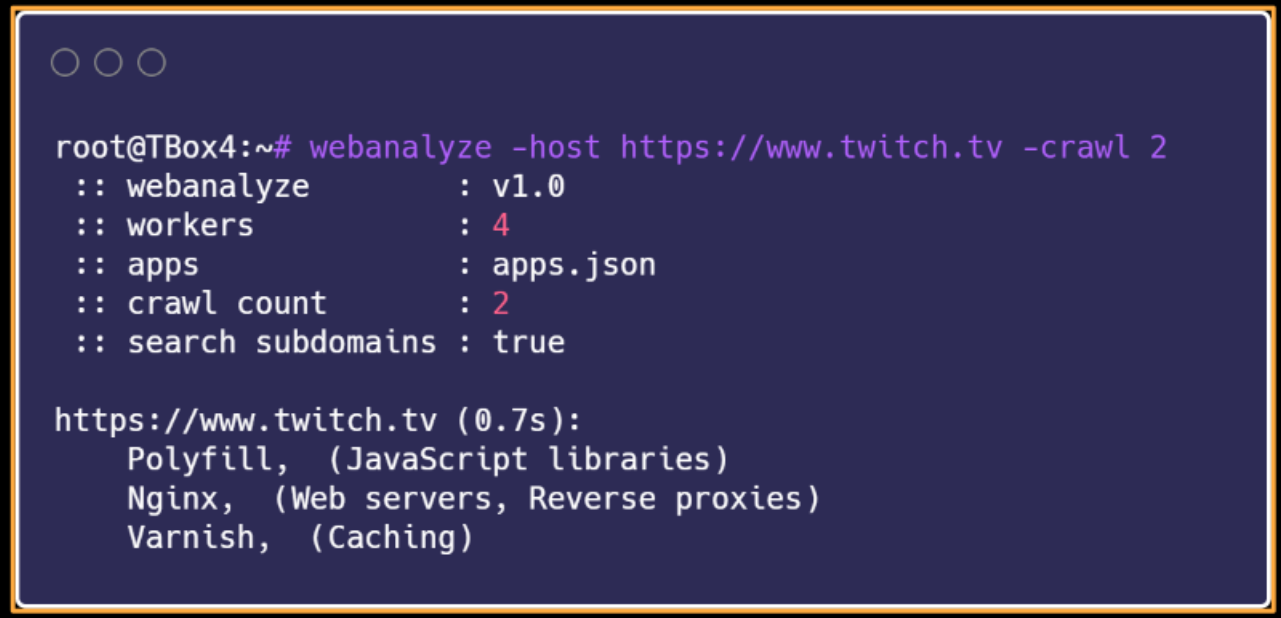




ANALYSIS CONCEPTS (TECH PROFILING)

If you don't want to use a browser extension you can automate Wappalyzer by using a tool called Webanalyze which will give you technologies of your application on the command line.

This is mostly useful if you're building your own Recon framework and you want to enrich your existing data by adding tech stacks two known websites in your data-set.



ANALYSIS CONCEPTS (THE BIG QUESTIONS)

When analyzing a web application, in addition to understanding its tech stack, it's also important to ask yourself pointed questions about how the application operates.

Here are six-ish questions that I use to lead me to better understanding of the application and how I will approach hacking it.

We will explore these in more depth later...

How does the app pass data?

How/where does the app talk about users?

Does the app have multi-tenancy or user levels?

Does the app have a unique threat model?

Has there been past security research & vulns?

How does the app/framework handle specific vuln classes?

VULNERABILTY DISCOVERY AUTOMATION

AUTOMATED VULN DISCOVERY

A contentious topic among bug bounty hunters (but usually not red teamers) is the idea of scanning for known vulnerabilities.

We'll discuss this contention a little bit later in the course but automated vulnerability discovery using tools is a tremendous advantage in several ways.

First, using automated tools as a first pass can find you vulnerabilities on scope that you found during Recon that no one else has seen yet.

Second, these tools outlined in the next section are good for more than just scanning for CVE's. They also help you identify what is running on a web application, find login panels, test for default credentials, and more.

Lastly, if you are doing your own vulnerability of research and you find something that could be present in many sites across a scope or campaign you can use these tools to automate and scan your original research over many websites and hosts.

AUTOMATED VULN DISCOVERY(NUCLEI)

Nuclei Scanner is a tool by the Project Discovery team.

- 1000+ CVE's
- 100+ Informational detections
- 500+ admin panel detectors
- 1500+ other checks
  - creds/keys
  - 67 subdomain takeover
  - Http form brute force
  - 3428 total templates


```
nuclei -l tesla_httprobe.txt -t brute-force/* -t cves/* -t basic-
detections/* -t dns/* -t files/* -t panels/* -t security-misconfiguration/*
-t subdomain-takeover/* -t technologies/* -t tokens/* -t vulnerabilities/*
```

```
[content-delivery-network:akamai] [http] https://auth.tesla.com/
[content-delivery-network:akamai] [http] http://auth.tesla.com/
[content-delivery-network:akamai] [http] https://3.tesla.com/
[content-delivery-network:akamai] [http] http://3.tesla.com/
[ntlm-directories] [http] http://autodiscover.tesla.com/powershell/
[web-server:ms-iis] [http] http://autodiscover.tesla.com/
[web-server:apache] [http] https://employeefeedback.tesla.com/
[content-delivery-network:akamai] [http] http://employeefeedback.tesla.com/
[web-server:apache] [http] https://feedback.tesla.com/
[content-delivery-network:akamai] [http] http://feedback.tesla.com/
[content-delivery-network:akamai] [http] https://edr.tesla.com/
```

AUTOMATED VULN DISCOVERY(NUCLEI)

Nuclei can be extended with a couple additional projects that add a ton of templates.

AllForOne and CENT.

It's important to understand that while you can extend your number of templates many of them might be duplicates of the core set. Additionally, many of them will not be useful at all.

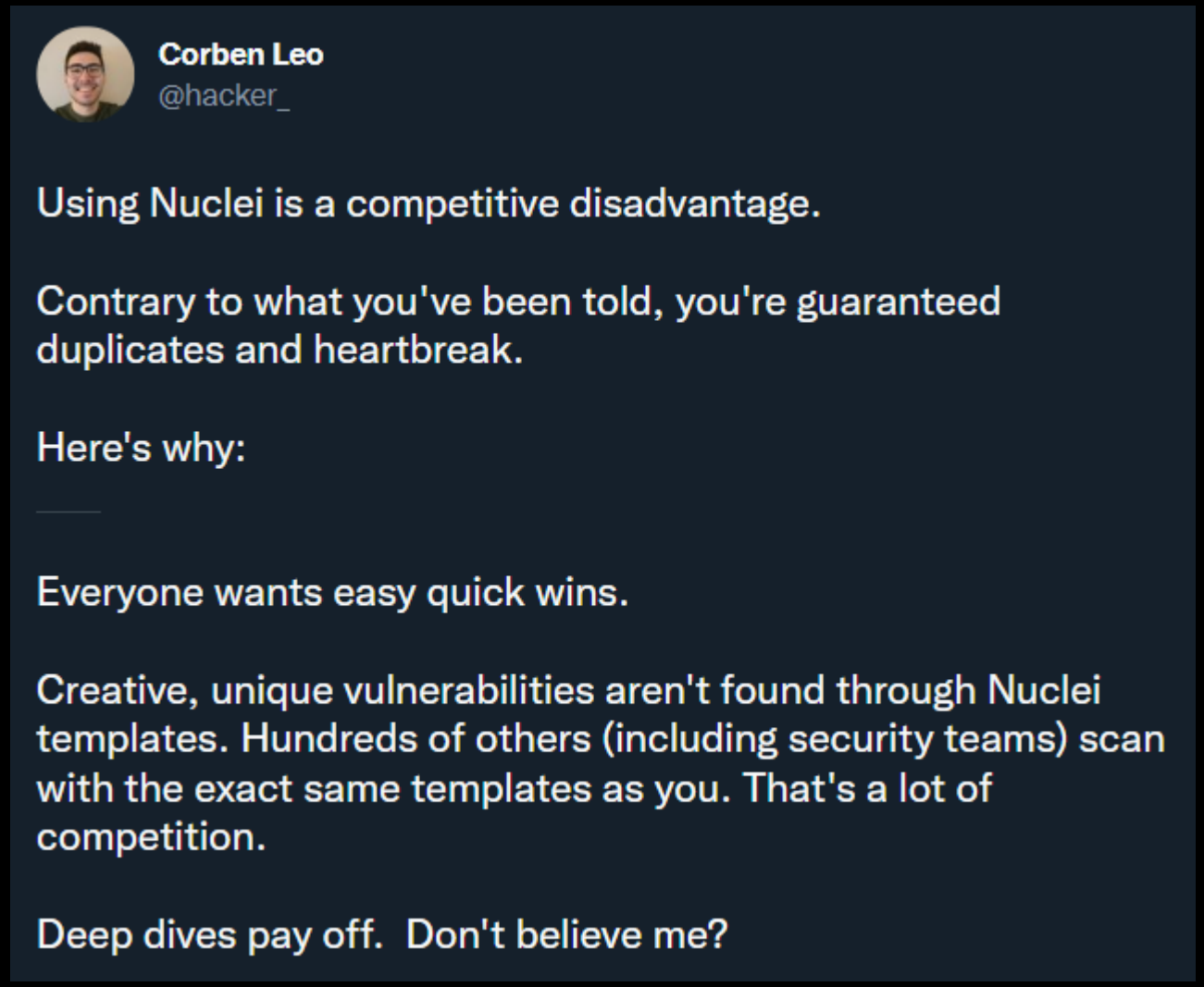
With keeping that in mind, you can find some gems in these collections of templates...

AllForOne

Nuclei Template Collector

CENT

AUTOMATED VULN DISCOVERY



Corben Leo
@hacker_

Using Nuclei is a competitive disadvantage.

Contrary to what you've been told, you're guaranteed duplicates and heartbreak.

Here's why:

Everyone wants easy quick wins.

Creative, unique vulnerabilities aren't found through Nuclei templates. Hundreds of others (including security teams) scan with the exact same templates as you. That's a lot of competition.

Deep dives pay off. Don't believe me?

Corben is right, but in a certain context.

If you are testing a main site or highly known target, then yes, scanning has probably been done already.

However, if you are testing targets that are “fresh” or found via recon and you have a feeling not many testers have seen them before, it can absolutely find great things.

In addition, making templates for new vulns or content discovery techniques in Nuclei is SUPER easy.

AUTOMATED VULN DISCOVERY (CREATING)

One other way to utilize these automated vulnerability discovery tools is to quickly port security research that's been done into a template that you write yourself and then scan targets for.

So how does one keep up to date on the most current security research to port it into a template?

Here is the workflow I've used for a long time but as currently in trouble because of Twitter API limits and changes to tweet deck ☹

Secrets of automation-kings in bug bounty - https://bit.ly/3clYgdt

AUTOMATED VULN DISCOVERY (CREATING)

- IDOR OR "insecure direct object reference" OR "broken access control“
- "SQLi" OR "SQL injection“ OR “injection flaw” OR “injection vulnerability”
- “local file include” OR “LFI” OR “directory traversal” OR “remote file include”
- “RCE” OR “command injection”
- “XSS” OR “Cross-site Scripting” OR “cross site scripting”
- “SSRF” OR “server side request forgery”
- “XXE” OR “XML External Entity attack”
- “CSRF” OR “Cross site request forgery”
- “insecure deserialization” OR “Mass assignment vulnerability”

AUTOMATED VULN DISCOVERY (retirejs)

Another place vulnerabilities can be introduced is in web libraries.

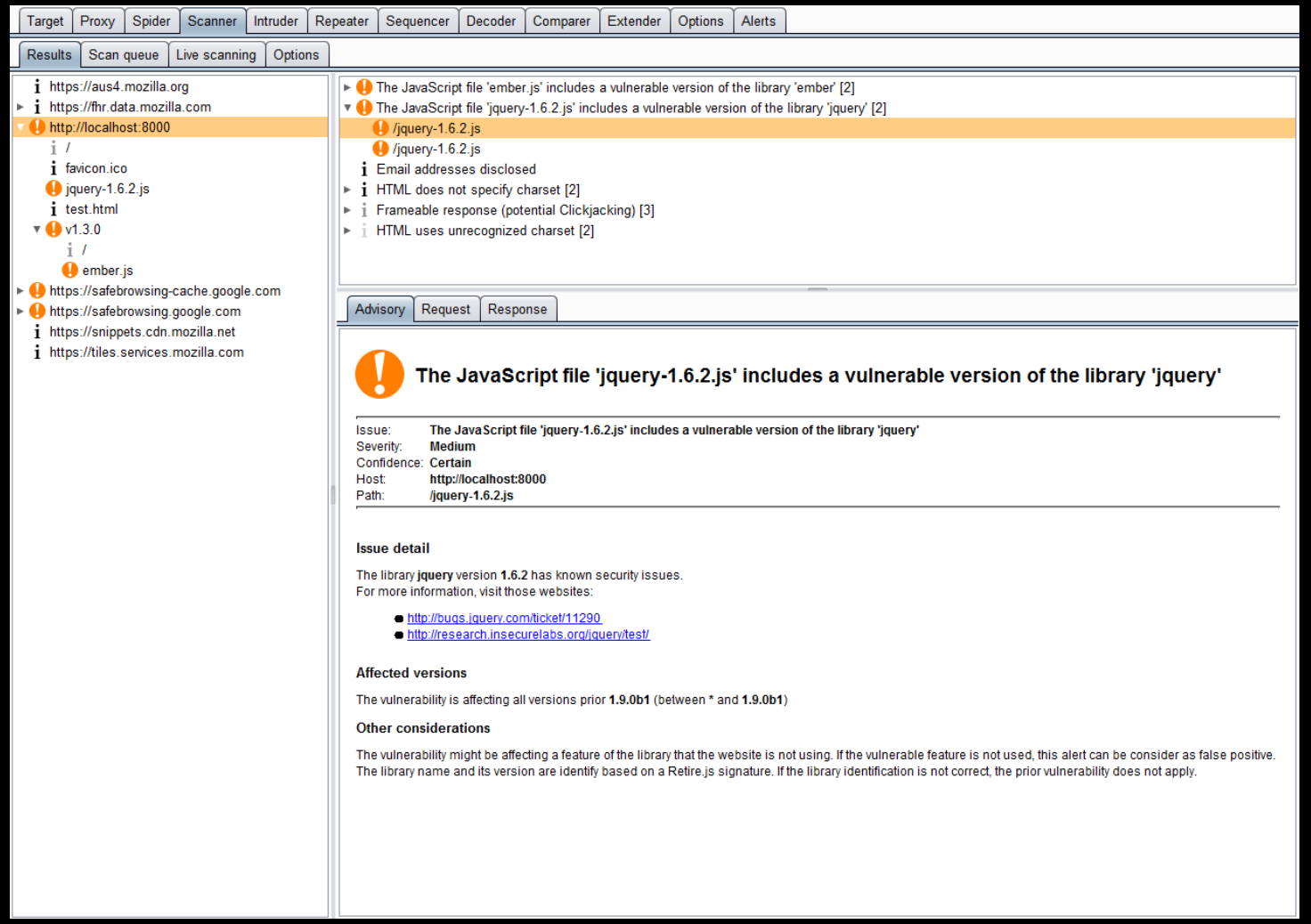
You can automatically scan for vulns in JavaScript libraries used by applications.

Luckily we've been privileged to have many open source projects that will scan for outdated libraries based on their version numbers that are usually in HTML source code.

One such tool is retireJS which on top of being a standalone tool also has a burp extension and can be instrumented in many other tools.

A word of warning, even though a version of a library might be present on a website it does not automatically mean that they haven't patched out a specific security vulnerability related to that library.

You will always have to prove out the vulnerability that any of these scanners finds for you.



CONTENT DISCOVERY

START WITH WALKING THE APP

CONTENT DISCOVERY TYPES

Content Discovery is the part of web application testing where you are trying to discover all the routes, paths, parameters, functions, and files of an application.

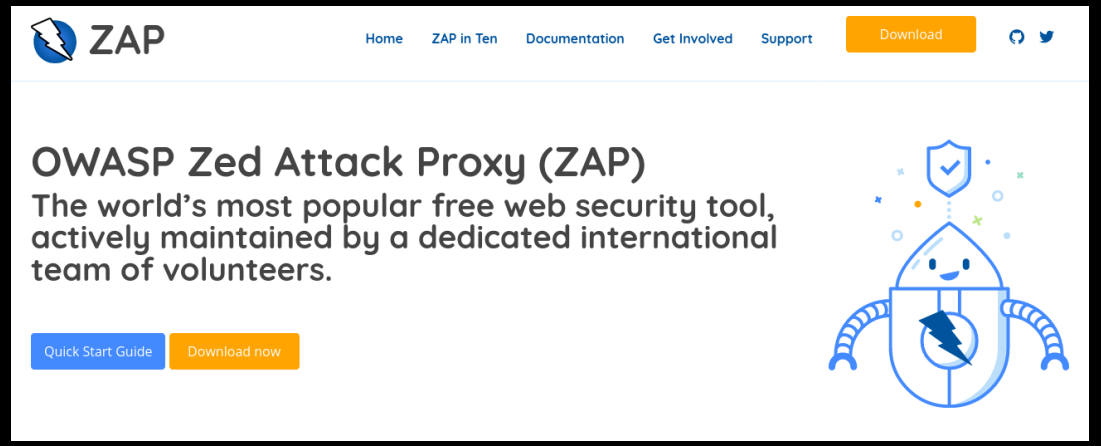
Sometimes content discovery can also be known as directory brute forcing but it is much wider topic than that.

I usually break down content discovery into six sections.

- Based on Tech
- Using known pathing (Install, DEMO, Leaked)
- Custom
- Historical
- Spidering
- Mobile

CONTENT DISCOVERY (SPIDERING)

Of course, a common way of understanding the structure of the application you're testing is just to spider it which finds all the known knowns. Most of you in this class will have spider to site before using burp suite or ZAP so we won't spend a ton of time on that. We also mentioned some of the command line crawlers that you can use in the Recon session yesterday!



CONTENT DISCOVERY (TOOLS)

TURBO INTRUDER

Gobuster v3.1.0

Gobuster is a tool used to brute-force:

- URIs (directories and files) in web sites.
- DNS subdomains (with wildcard support).
- Virtual Host names on target web servers.
- Open Amazon S3 buckets

ffuf - Fuzz Faster U Fool

FEROXBUSTER

SIMPLE. FAST. RECURSIVE.

dirsearch

dirsearch - Web path discovery

Wfuzz: The Web fuzzer

pypi v3.1.0 | license GPLv2 | python 3.4 | 3.5 | 3.6 | 3.7 | 3.8 | codecov 71%

Wfuzz provides a framework to automate web applications security assessments and could help you to secure your web applications by finding and exploiting web application vulnerabilities.

CONTENT DISCOVERY (TECH)

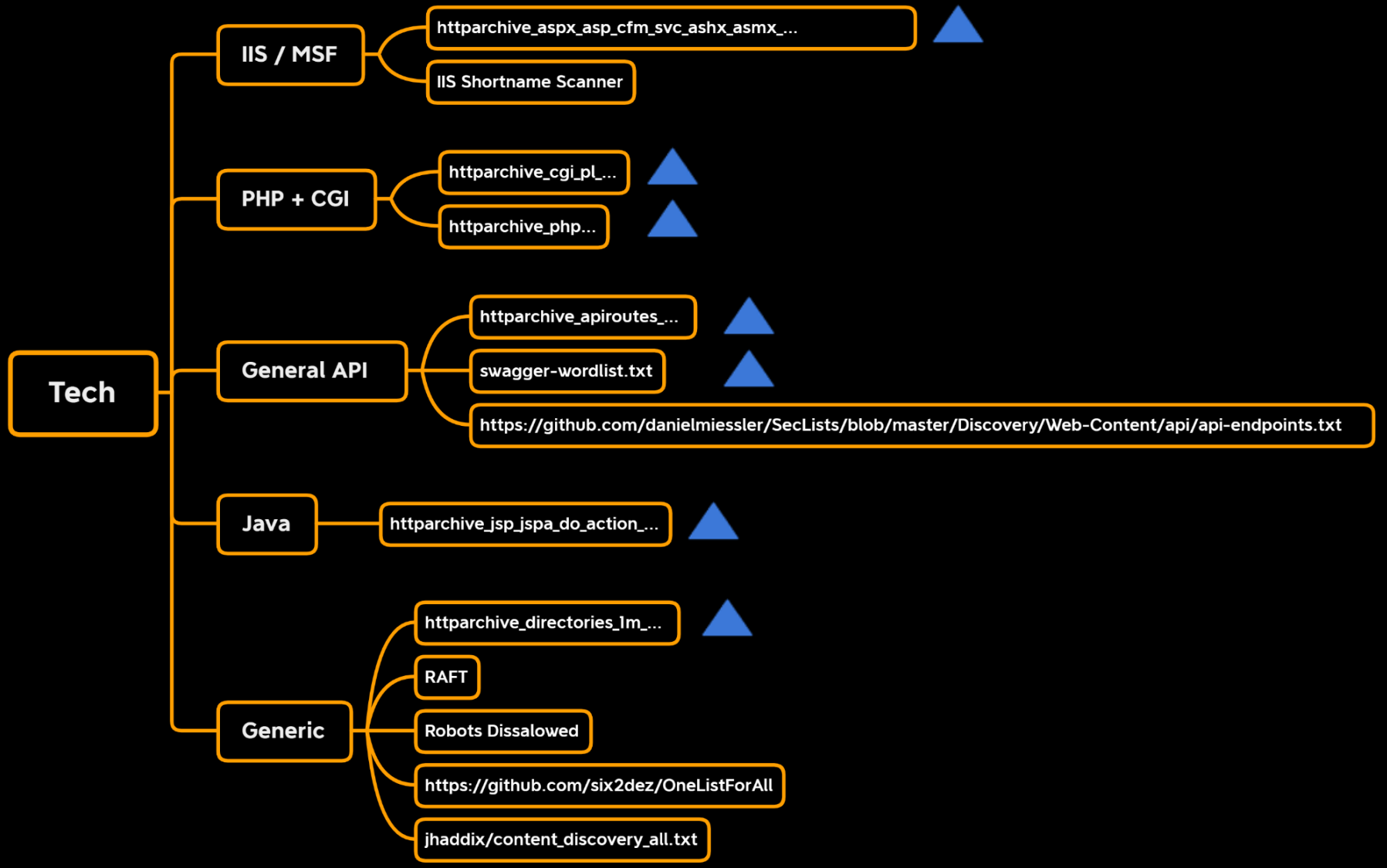
One way to approach content discovery is to use lists based on the type of technology the application uses.

This could be at its framework level, its web server level, or generically on what part of the application you are doing content discovery on (APIs).

We will use several different word lists for content discovery. The next slide outlines the lists I use based on technologies.

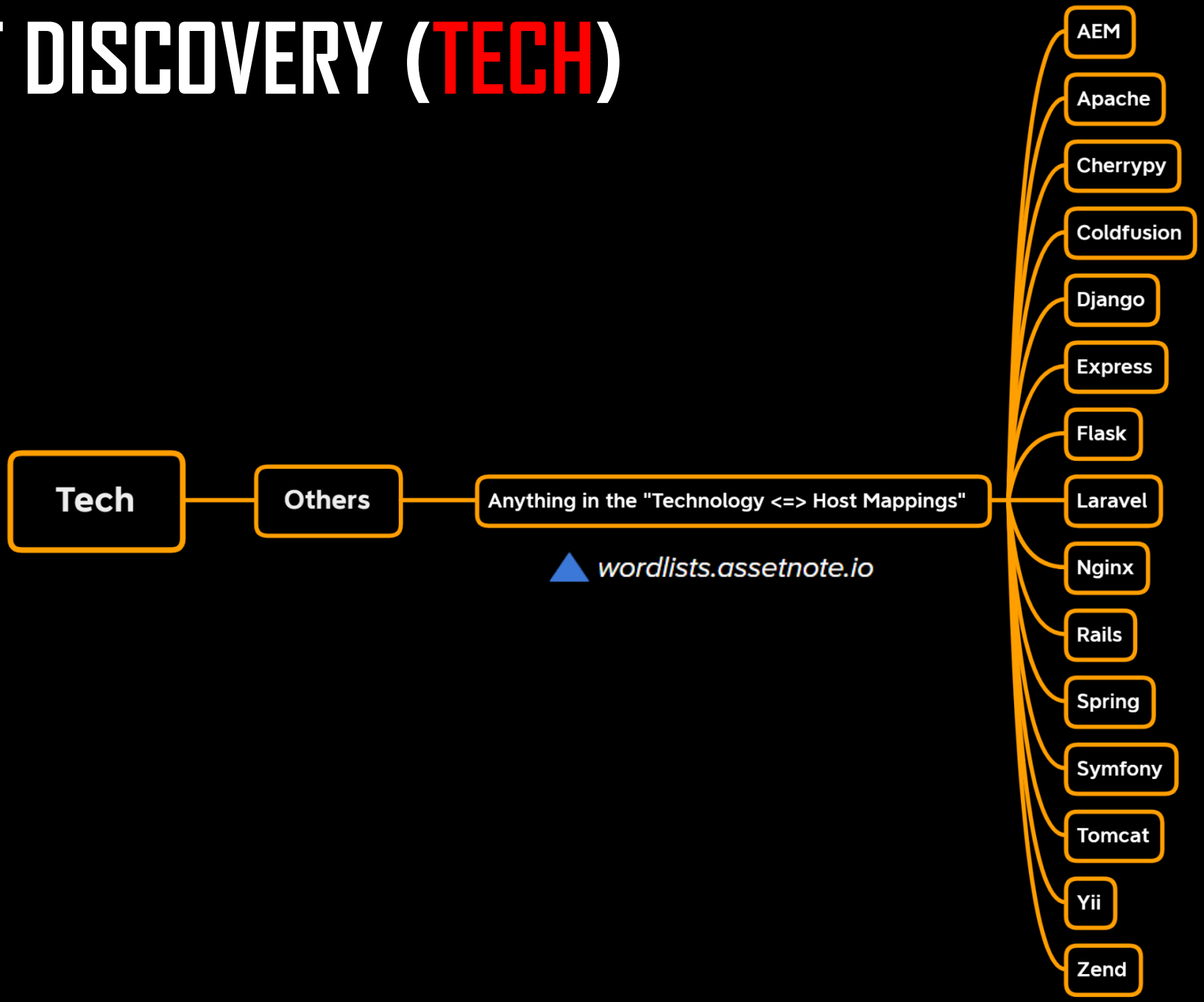
Many of them are found at wordlists.assetnote.io and in the repository Seclists.

CONTENT DISCOVERY (TECH)



wordlists.assetnote.io

CONTENT DISCOVERY (TECH)



CONTENT DISCOVERY (KNOWN PATHING)

In our quest to understand every nook and cranny of our application we might come to realize that the app we are testing is based on open-source software or paid software that you can purchase (COTS / Commercial Off The Shelf Software).

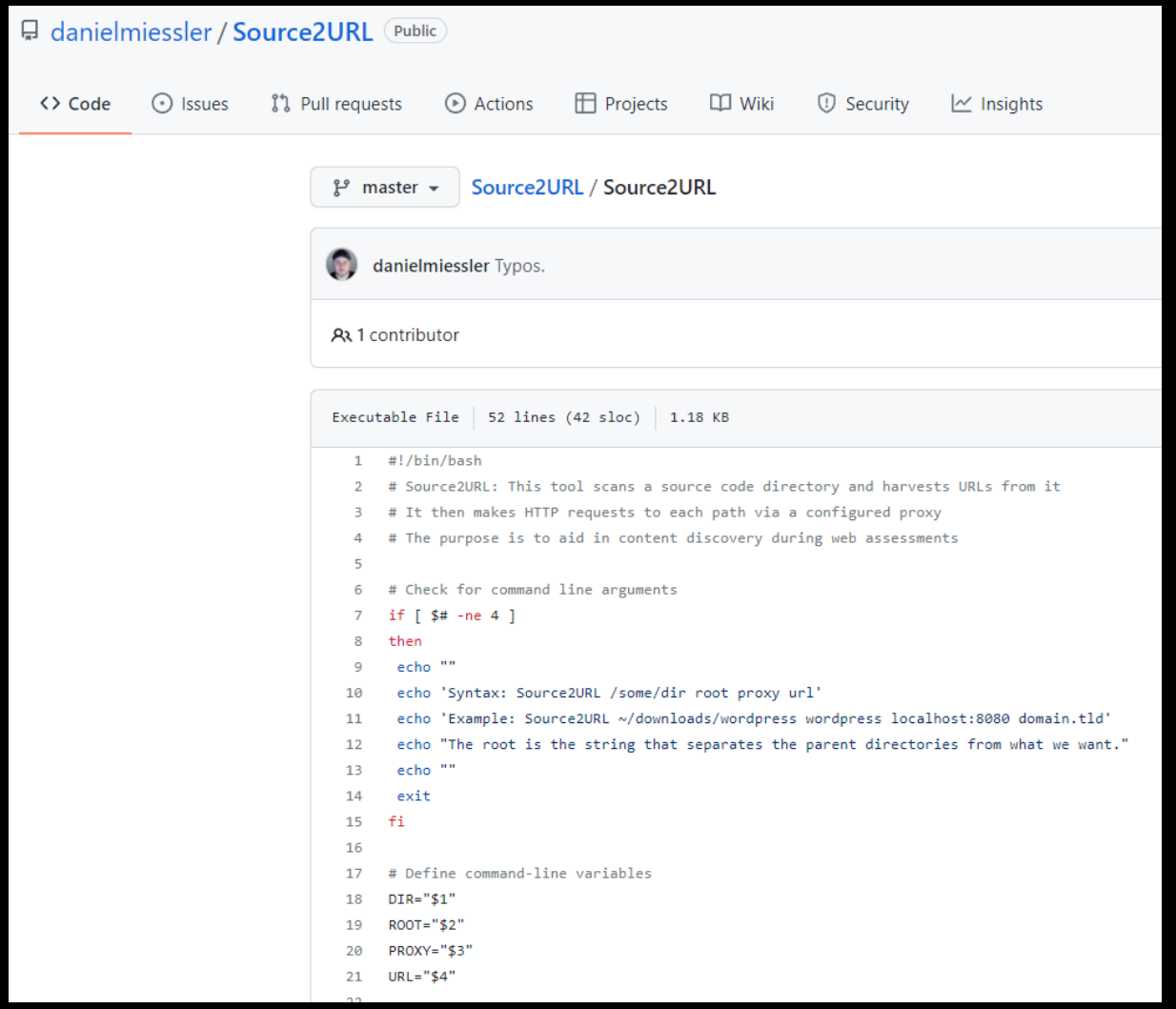
We can potentially install both types of software ourself and understand the pathing and underlying non-custom code for the application.

There are several ways to do this.

- Local Install
- Demos
- Installed and Leaked (Github or Dockerhub)

CONTENT DISCOVERY (LOCAL INSTALL)

If you can do a local install of the software because it's completely open source, then you can use Daniel Miessler's Source2URL tool to map the applications routes and endpoints and then proxy them through Burp back at your target domain.

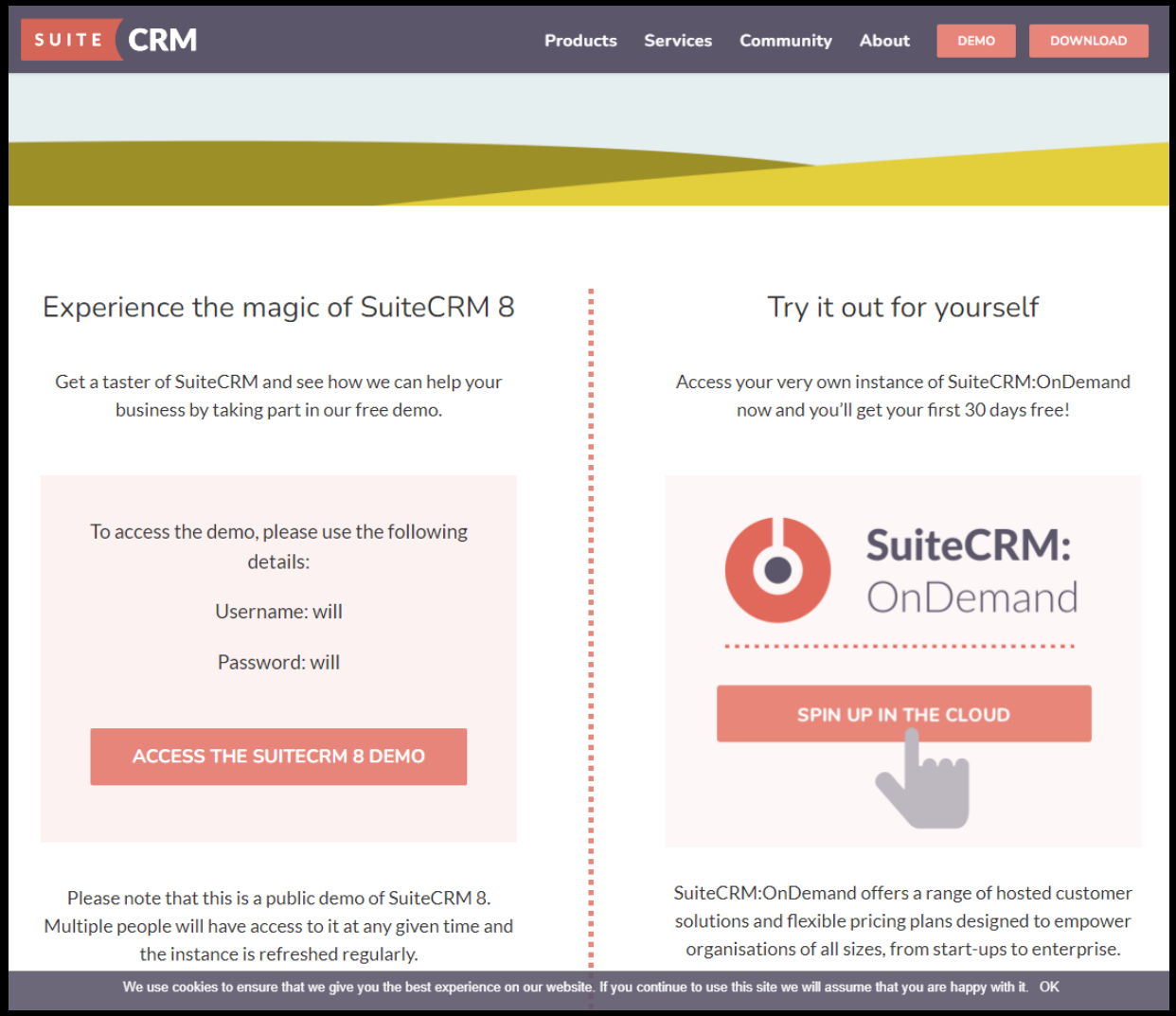


CONTENT DISCOVERY (DEMOS)

If you identify the application is a piece of paid software or COTS there is a chance that the vendor selling that software has a demo instance or a process to request a demo.

Getting access to a demo instance will allow you to route the application through burp suite and grab the pathing for that software.

When doing this it's especially important to make sure that you have access to the admin functionality of the software and you grab all those paths, routes, or parameters when proxying.



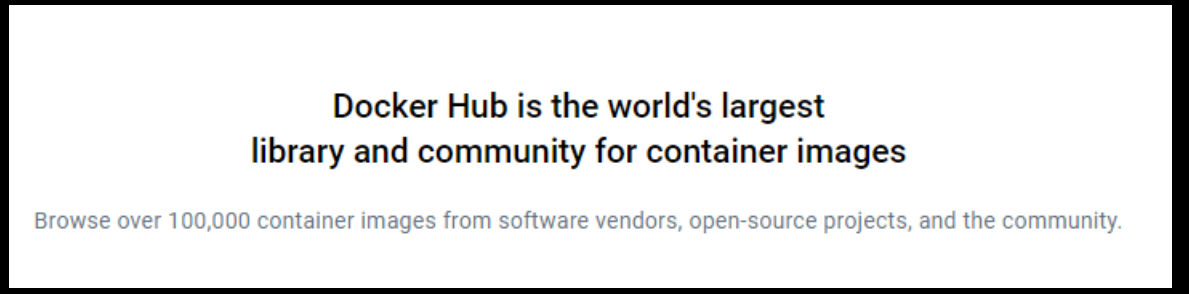
CONTENT DISCOVERY (INSTALLED AND LEAKED)

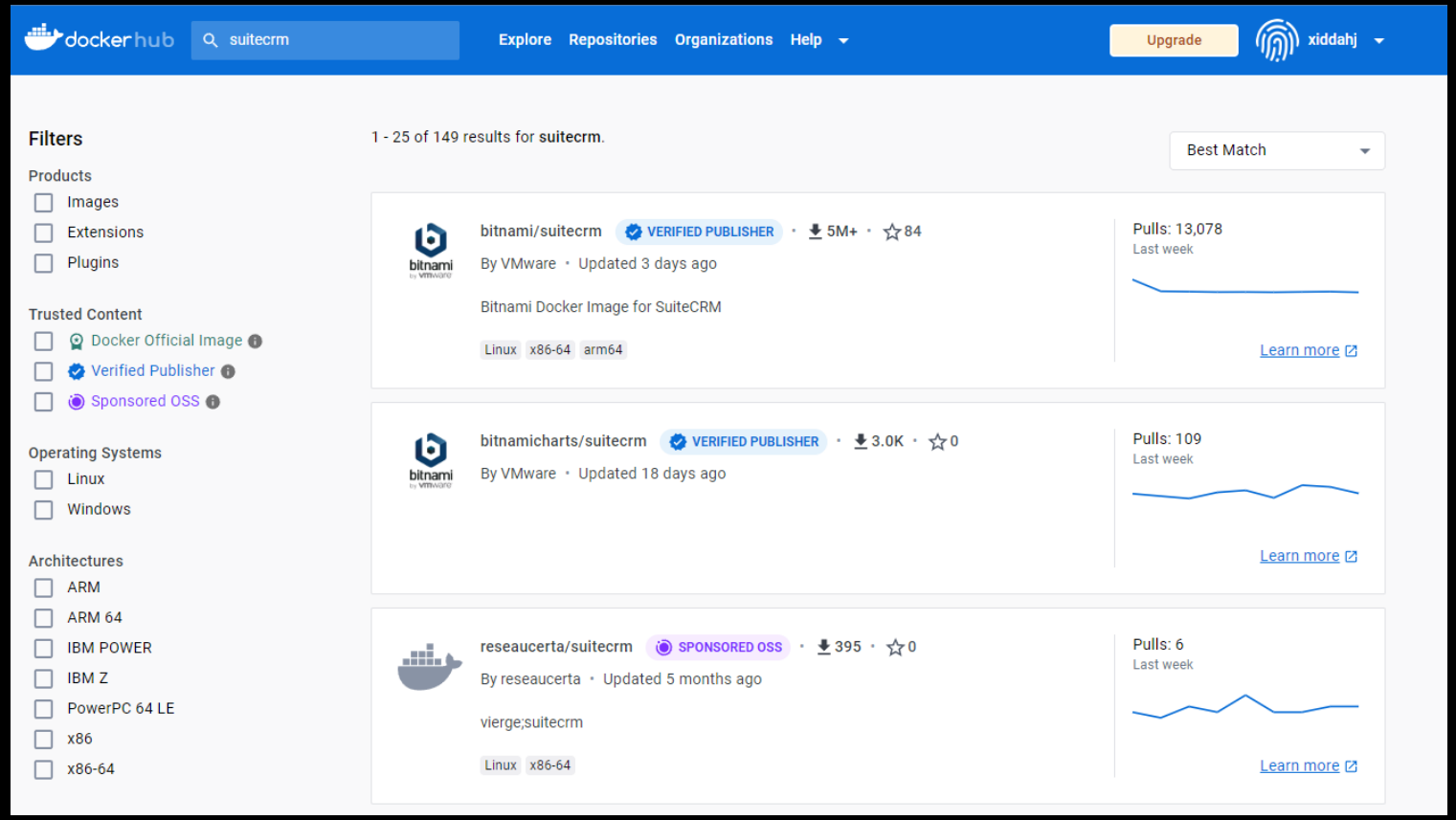
Lastly, you might identify a piece of software is paid or cots, but there is no demo available.

Many times, when installing applications like this, developers do it through dockerhub and accidentally end up posting the source code somewhere.

Searching through dockerhub can find you an installed instances of the software and because of the nature of docker hub many times you can get access to the complete source code.

Other times you can gain snippets of a piece of paid software through developers posting modifications or sections of it on GitHub.





CONTENT DISCOVERY (CUSTOM)

Sometimes using custom word lists to discover paths and routes can be beneficial.

This process we'll take contextual words related to our application and try to build a word list so that you can brute force paths and parameters based on that context.

For example if you're testing a bank application and you build one of these word lists, it may include the word “invest” or “transaction” or “deposit”.

There are a few tools to do this in the industry, the first of which was CEWL.

After much experimentation I have actually gone back to CEWL when I do this sort of content discovery.

You can also use another workflow that feeds GAU into a tool called wordlistgen.

```
echo bugcrowd.com | gau | wordlistgen | sort -u
```

```
cewl https://www.geeksforgeeks.org/
```

```
array
bound
Copy
URL
computer
articles
Round
Page
Self
Paced
Classes
Working
Professionals
Feed
different
operations
important
Properties
Bound
Theory
Preparation
System
PHP
Must
This
org
Comments
Sort
Fibonacci
```

CONTENT DISCOVERY (HISTORICAL)

Another method for content discovery is to try and search sources that archive URL data.

There exist several of these sources such as the wayback machine, alien vault, common crawl, and URL scan.

In the previous slide we used gau to do this. Last year one of our esteemed class members wrote a tool that utilize this technique but goes much deeper.

Grabbing these URL's from these sources is where many of the tools stop. WayMore by Xnl-h4ck3r will also download the archived responses and search through them for more links and extra information.

CONTENT DISCOVERY (TIP – RECURSION)

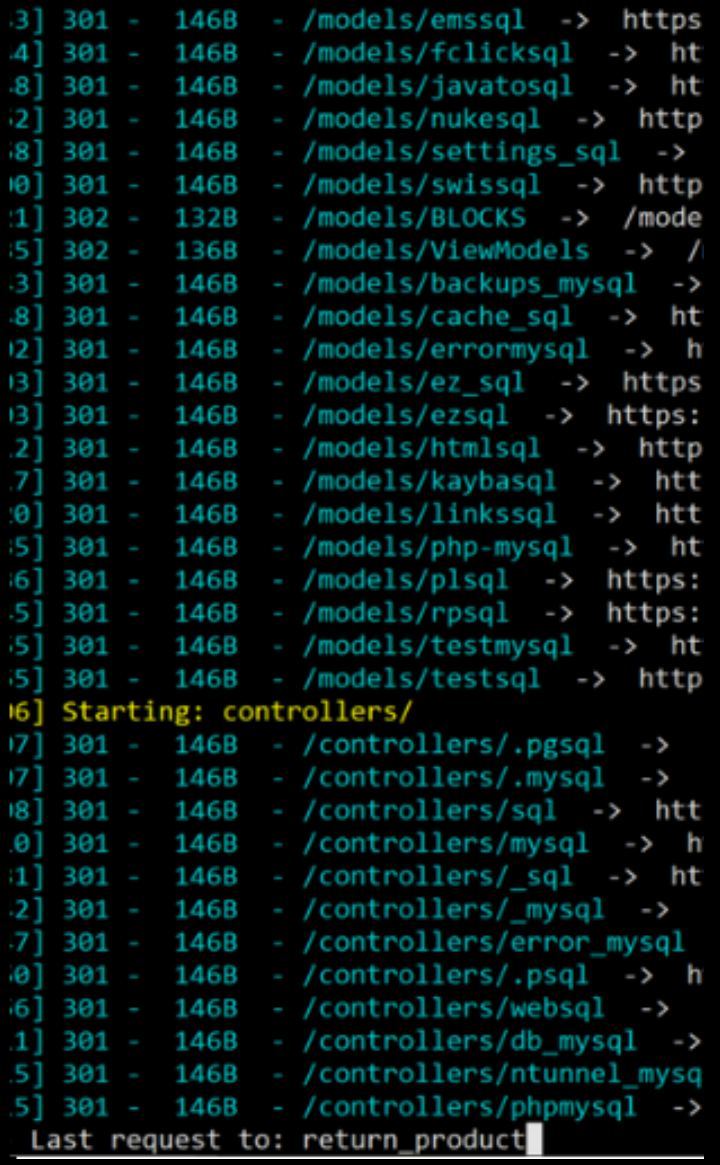
Bounty Tip

02

"Content discovery" is trying to guess sensitive paths and files that might exist in the application but are not linked anywhere.

Often when doing this part of web assessment you will run into 401 Not Authorized responses. It is beneficial to recursively brute force that path. Often resources past that path have not had the same access controls applied. In addition, 401 replies should be investigated with waybackmachine (https://archive.org/web/) to see if they ever did not have authentication applied and to garner clues about the application pathing.

https://someapp.com/admin/	401
https://someapp.com/admin/dashboard/	401
https://someapp.com/admin/dashboard/members	200

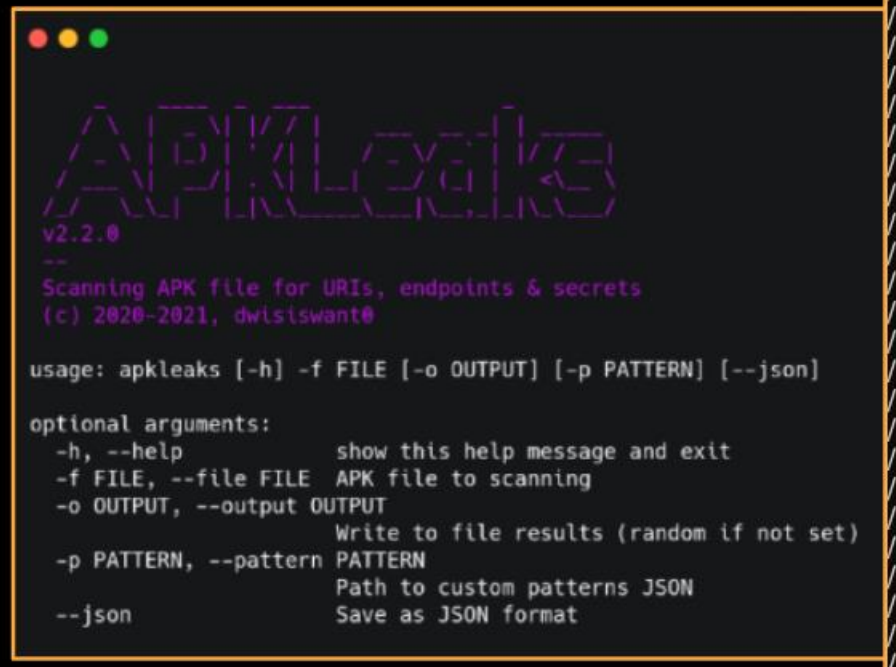


CONTENT DISCOVERY (TIP – MOBILE)

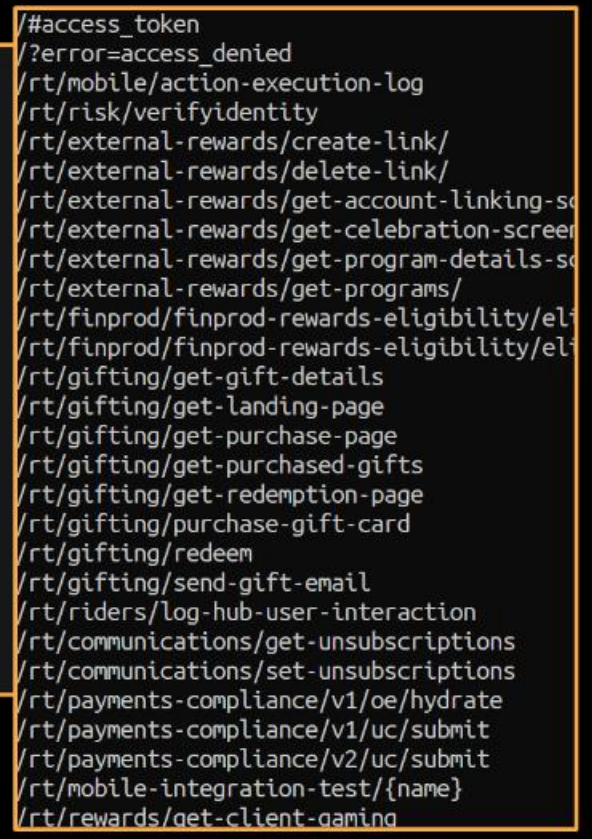
Often mobile application binaries can contain pathing for the same website we might be testing!

Usually this is an API hanging off our main domain.

You can use APKleaks to parse out paths from an APK file to get additional routes and API endpoints and parameters.



github.com/dwisiswant0/apkleaks



ubereats.com

JAVASCRIPT ANALYSIS

WHAT ARE WE AFTER?

Endpoints, parameters, routes, secrets, domains.

JAVASCRIPT ANALYSIS

One of the most important parts of assessing a web application, that ties into content discovery, is analyzing the sites JavaScript.

If the application is a heavy JavaScript framework than many of the routes or parameters will be defined in the JavaScript.

Many times API references will also be housed in JavaScript files.

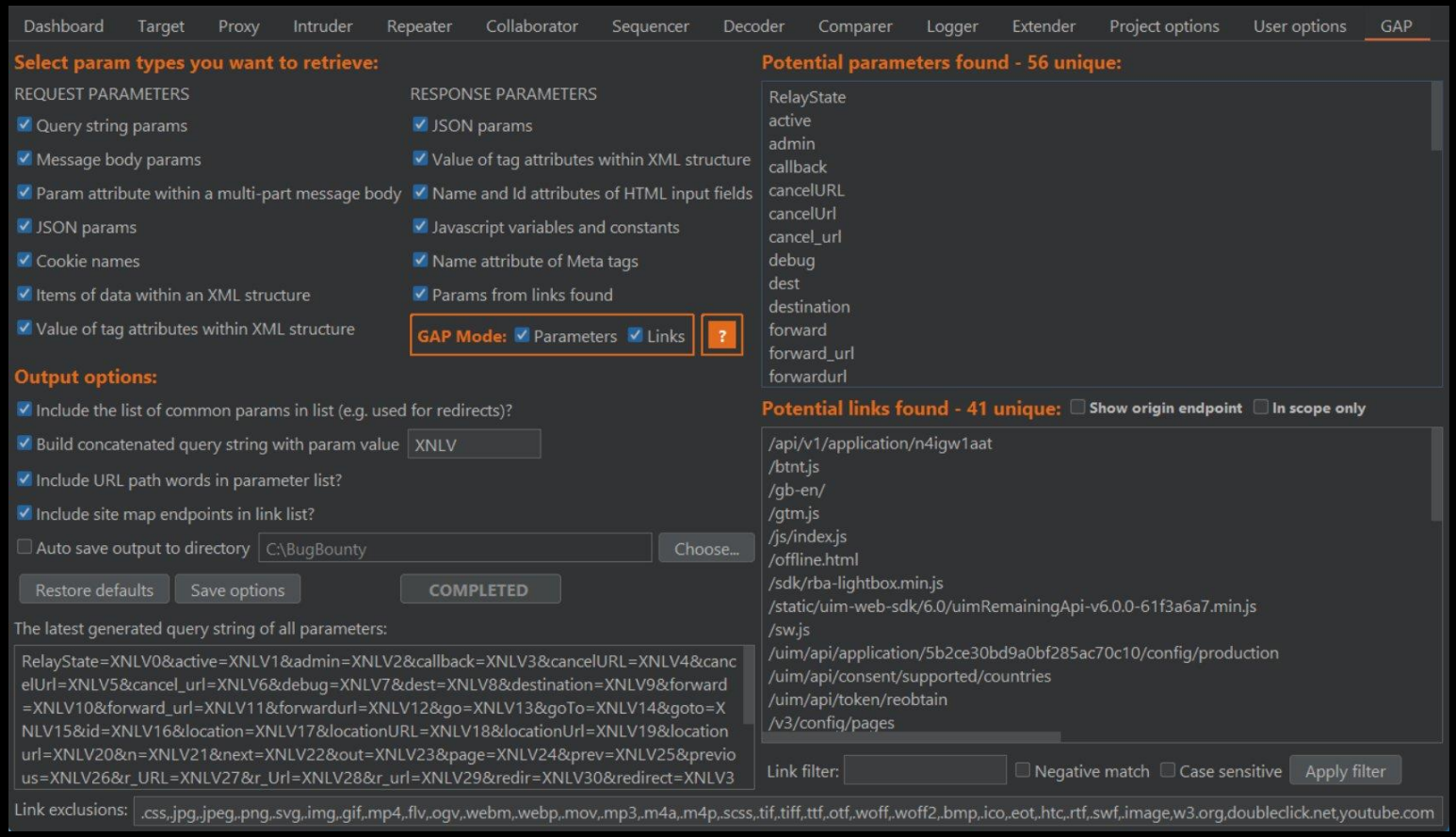
Additionally, in the worst case scenarios, developers can also store secrets in JavaScript thinking that no one will find them.

```
function App() {
  return (
    <Router>
    <div>
          <h2>React Router Step By Step Tutorial</h2>
          <nav>
          <ul>
            <li><Link to={'/'} > Home </Link></li>
            <li><Link to={'/contact'} >Contact</Link></li>
            <li><Link to={'/about'} >About</Link></li>
            <li><Link to={'/services'} >Services</Link></li>
          </ul>
          </nav>
          <Switch>
                <Route path = "/" exact component = {Home}></Route>
                <Route path = "/contact" component = {Contact}></Route>
                <Route path = "/about" component = {About}></Route>
                <Route path = "/services" component = {Services}></Route>
          </Switch>
    </div>
    </Router>
  );
}
```

JAVASCRIPT ANALYSIS (PATHS VIA GAP)

Again one of our esteemed class members, XNL, has also created a tool that helps us parse paths and routes out of JavaScript files.

Gap is a burp extension where you can right click on your entire scope and it will grab links , endpoints, and parameters from not only the JavaScript files but also inline JavaScript.

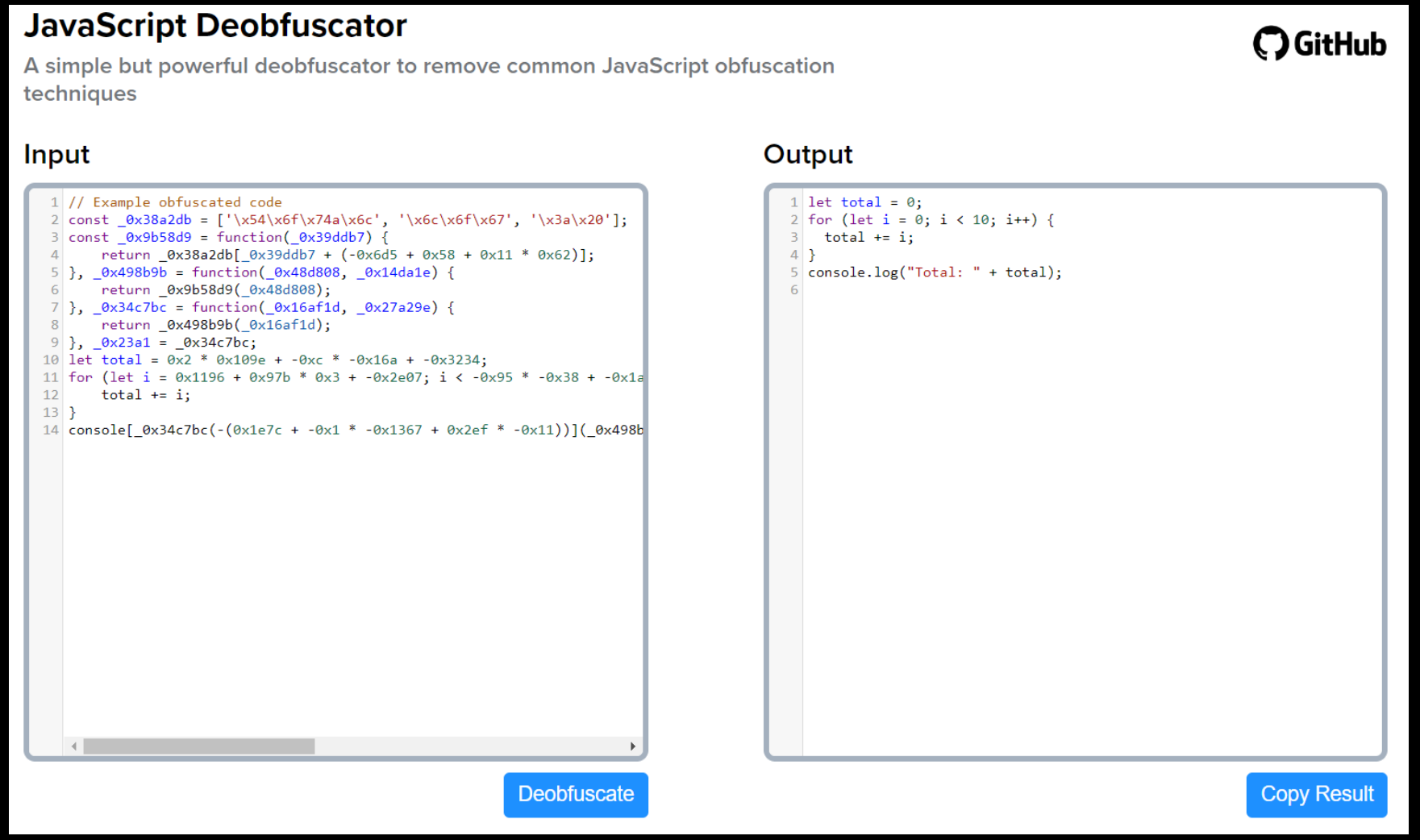


JAVASCRIPT ANALYSIS (BEAUTIFYING++)

If you run into heavily packed or obfuscated JS code, give https://deobfuscate.io a try.

http://deobfuscate.io:

- Unpacks arrays
- Simplifies expressions
- Beatifies the code
- and more



JAVASCRIPT ANALYSIS (MINING)

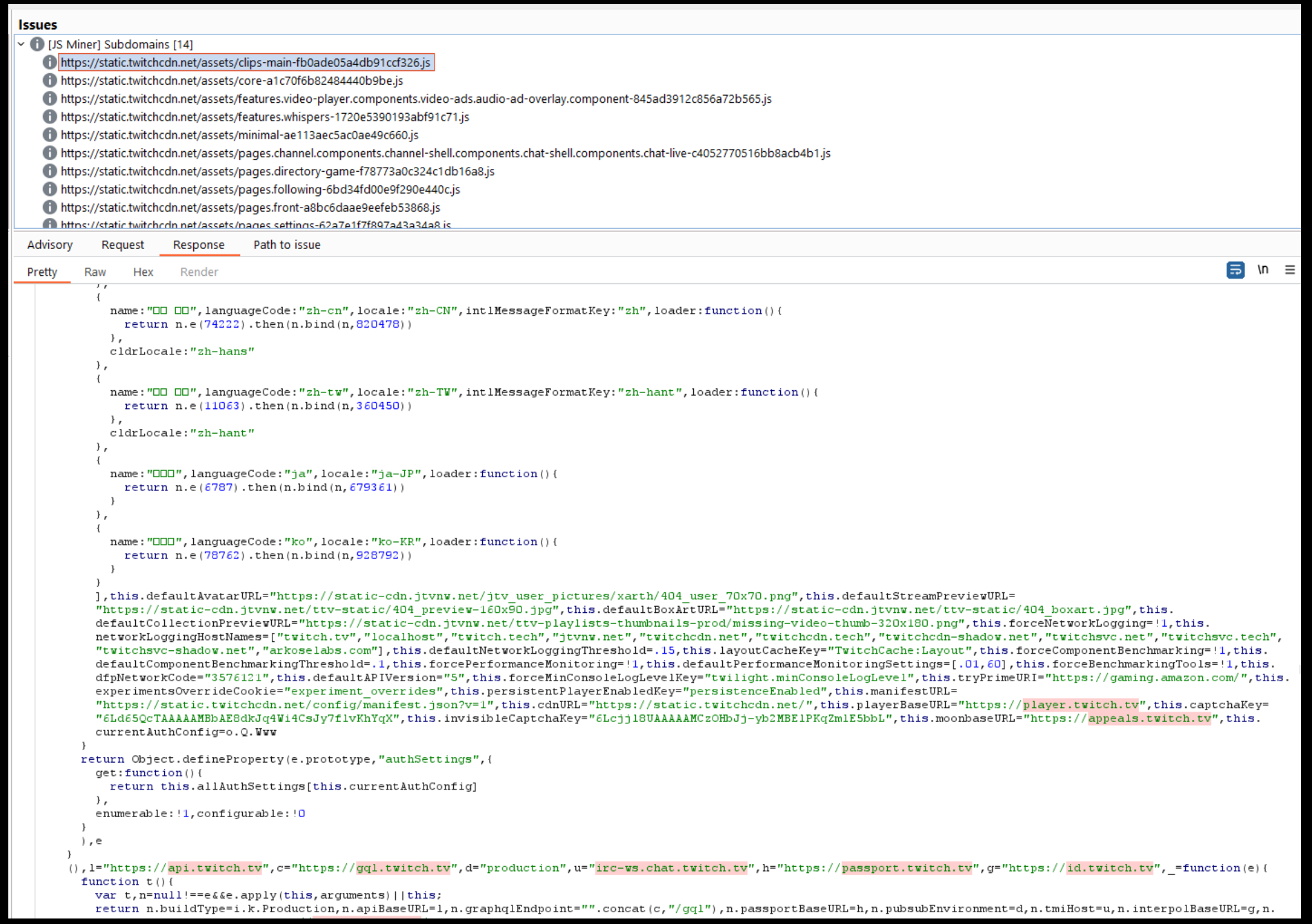
You should be looking for verbatim, hard-coded secrets in JS files.

JS Miner (Burp extension) adds passive scanning checks to alert you of these.

https://portswigger.net/bappstore/0ab7a94d8e11449daaf0fb3874312255b

🔥 Not just regex either, it uses a (Shannon) entropy function for things that might be interesting.

JAVASCRIPT ANALYSIS (MINING)

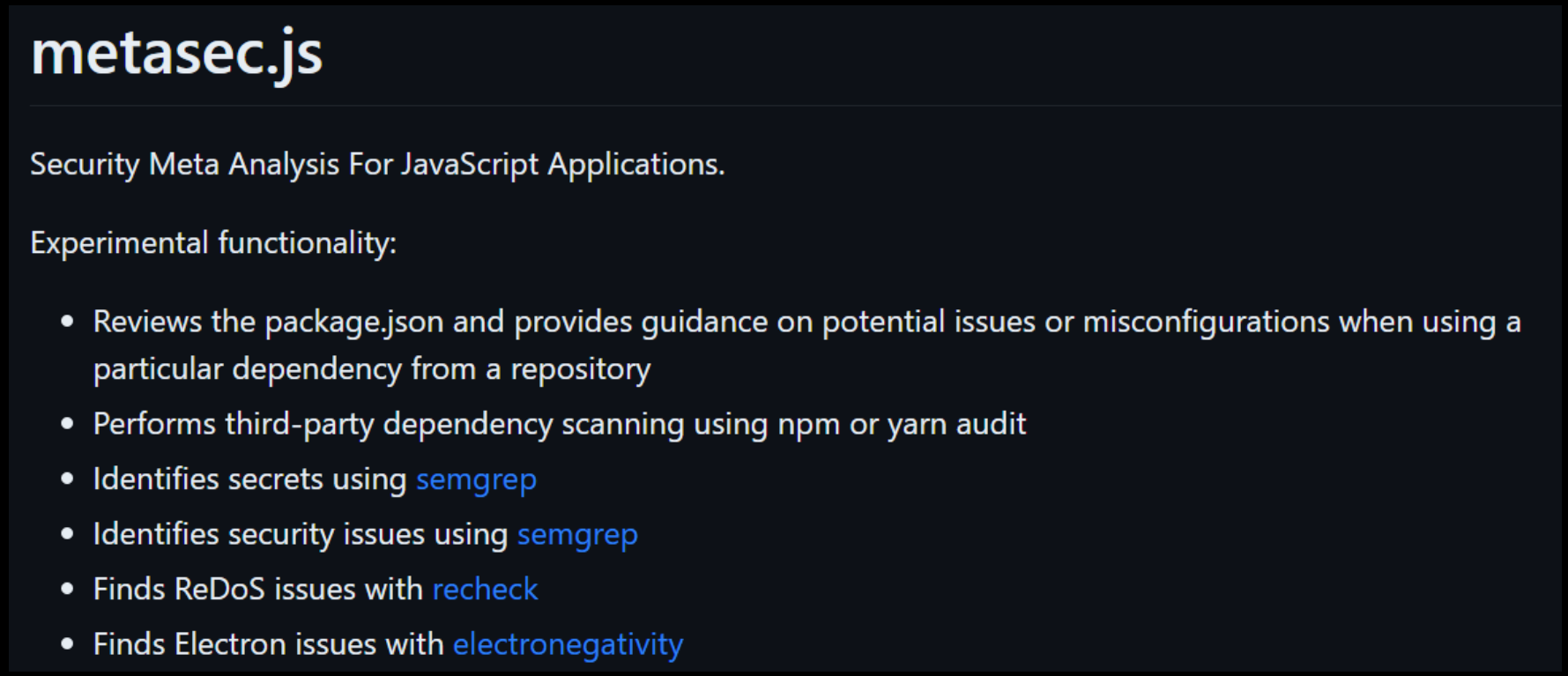


JAVASCRIPT ANALYSIS (SCANNING)

@LewisArdern made Metasec.js a while back which can be used on a downloaded JS file.

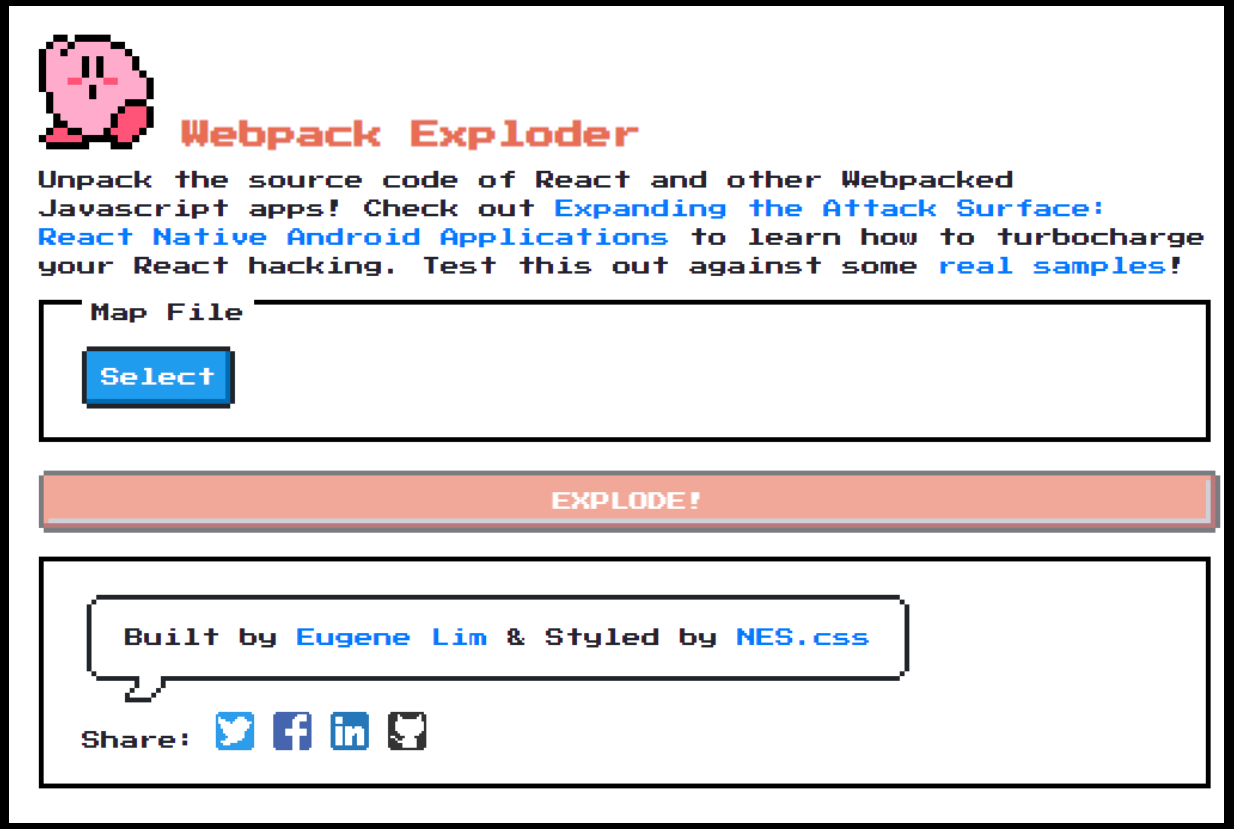
It uses the static source code security analysis engines of:

- 🔍 npm-audit
- 🔍 yarn-audit
- 🔍 semgrep for secrets and JS sec issues



JAVASCRIPT ANALYSIS (WEBPACKED)

Another resource for unpacking Webpacked JS files is Webpack Exploder by @spaceraccoonsec



The following strings can be grepped for in order to extract the Firebase API key from the `index.android.bundle` :

```
FIREBASE_API_KEY
FIREBASE_AUTH_DOMAIN
FIREBASE_DB_URL
FIREBASE_BUCKET
apiKey
```

For example:

```
❯ grep -rnis 'apiKey' index.android.bundle
... omitted for brevity ...

initializeApp({apiKey:"AIzaSyDokhX9fzFlJMfXjwbiiG-2fGDhi4kLPFI",
authDomain:"react-native-examples-bcc4d.firebaseapp.com",
databaseURL:"https://react-native-examples-bcc4d.firebaseio.com",
projectId:"react-native-examples-bcc4d",
storageBucket:"",
messagingSenderId:"928497342409"});

... omitted for brevity ...
```

In addition to finding Firebase credentials, the `index.android.bundle` file can also be analysed for API endpoints. In a React Native application I was reversing, I was able to find a number of API endpoints by browsing through the un-minified JavaScript in Chrome:

https://blog.assetnote.io/bug-bounty/2020/02/02/expanding-attack-surface-react-native/

JAVASCRIPT ANALYSIS (MORE)

Jsluice + Charlies thing

THE BIG QUESTIONS

BIG QUESTIONS (PASSING DATA)

The first question I asked myself when looking at an application is how does this app pass data?

Does it use a resource, parameter, value, format?

https://app.com/resource?parameter=value&param2=value

Or does it use a RESTful format?

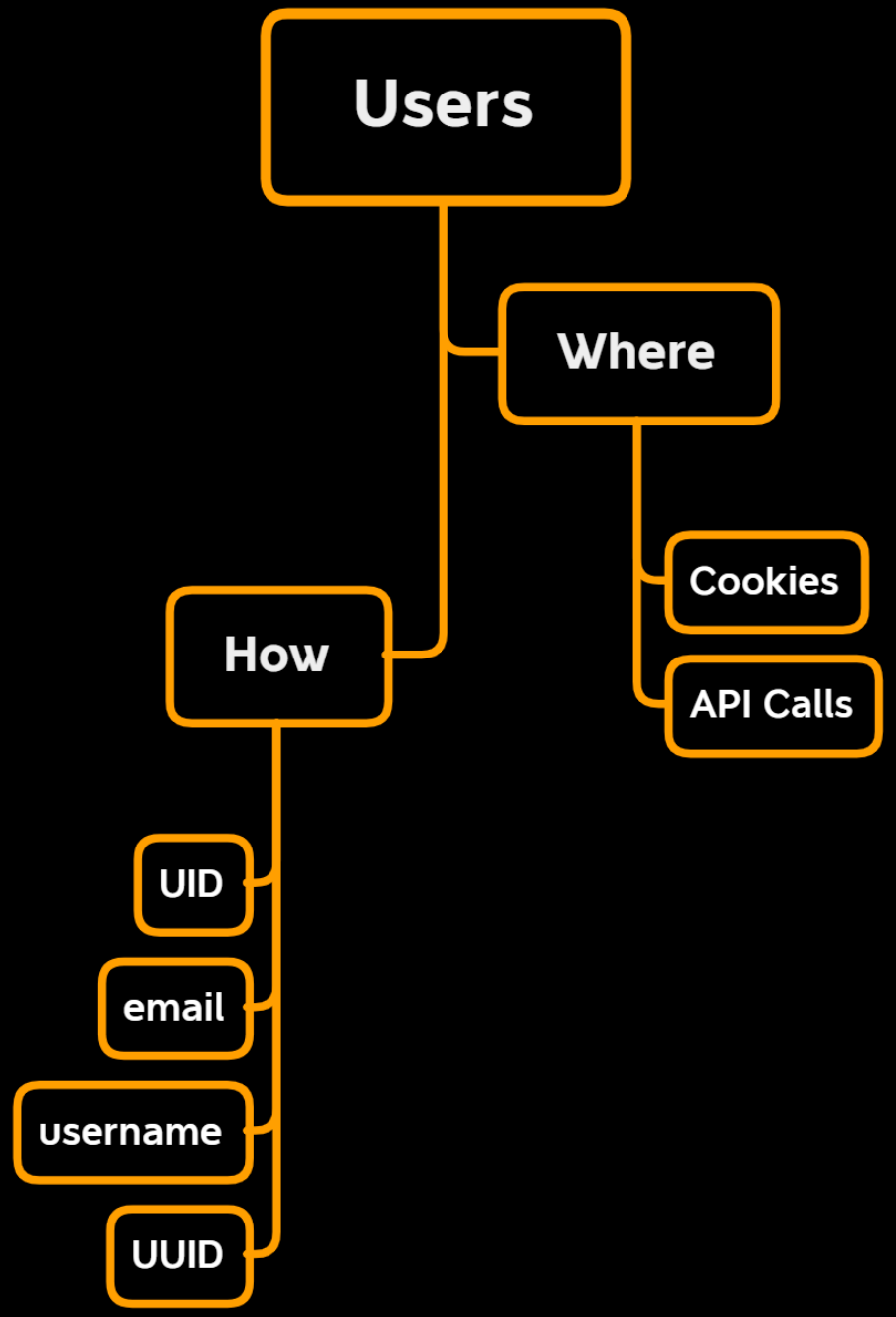
https://app.com/route/resource/sub-resource/...

Understanding this will be the cornerstone of how you test for vast categories of bugs. The bugs will be there, but if you're not familiar with where to inject your payloads, you will fail.

BIG QUESTIONS (USERS)

Next, I ask myself how/where does the app talk about users?

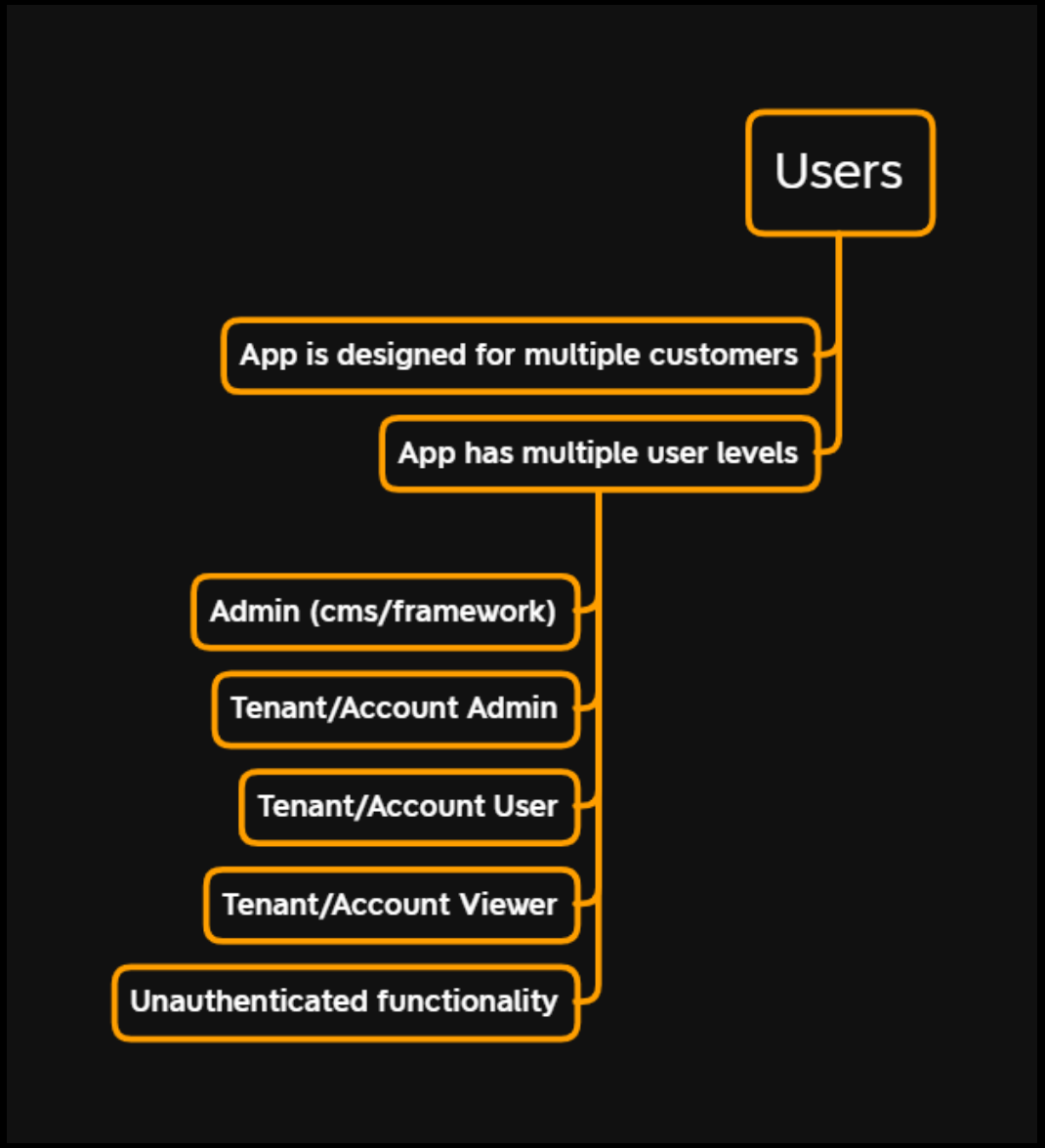
Understanding how users (yourself and other users) are referenced and where in the application is pivotal to finding several bug classes, most specifically Access, Authorization, Logic, and Information Disclosure bugs.



BIG QUESTIONS (USER LEVELS)

Does the site have multi-tenancy or different user levels?

This will also dictate how we test for authorization and access bugs.

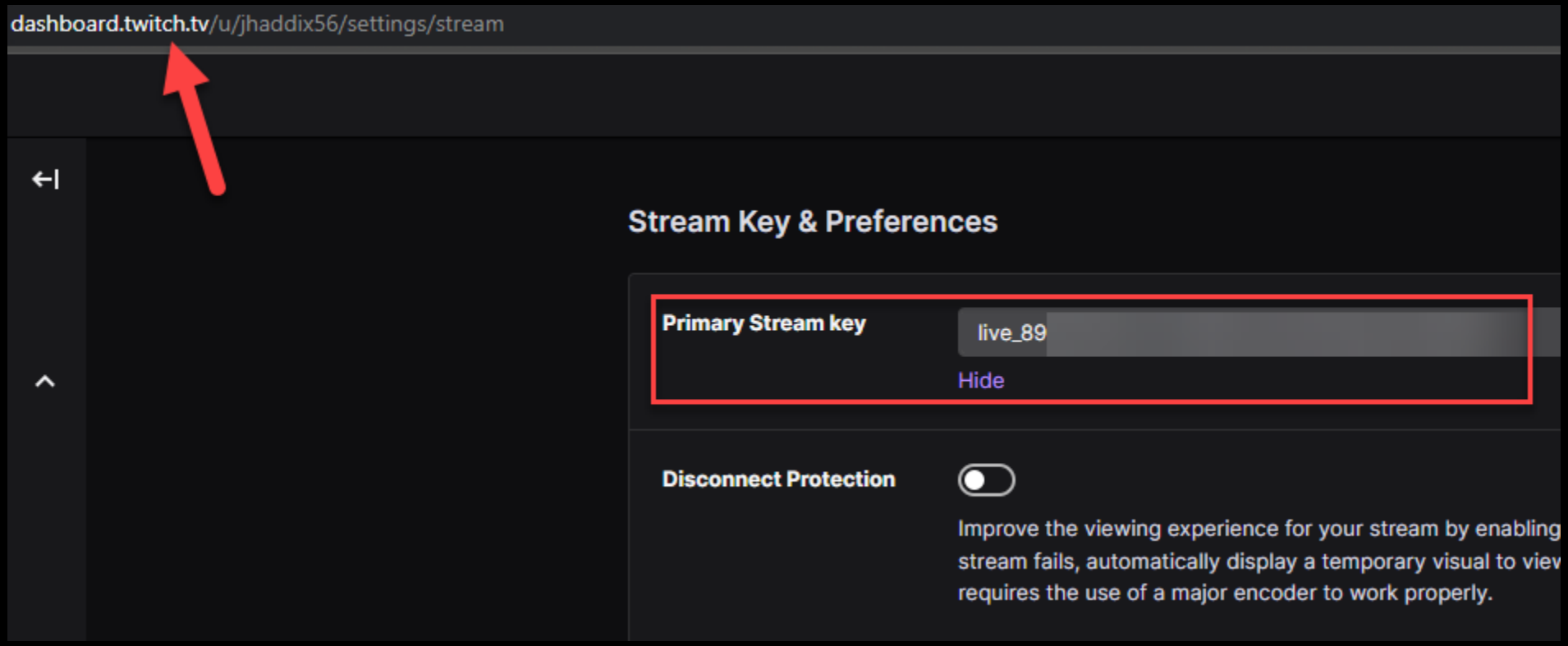


BIG QUESTIONS (THREAT MODEL)

Does the site have a unique threat model?

If the application houses more than the standard PII data, it's easy to forget to target that data in your testing.

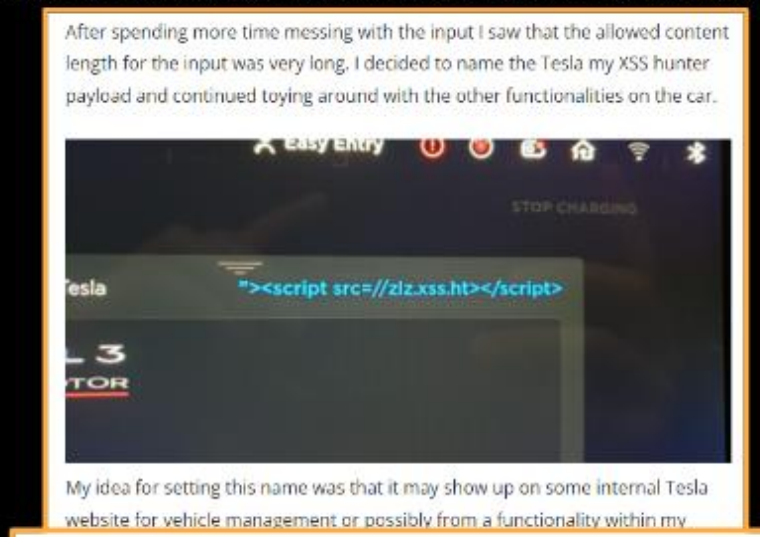
Examples: API keys, application data for doxing.



BIG QUESTIONS (SECURITY RESEARCH)

Has there been past security research & vulns?

https://samcurry.net/cracking-my-windshield-and-earning-10000-on-the-tesla-bug-bounty-program/



After spending more time messing with the input I saw that the allowed content length for the input was very long. I decided to name the Tesla my XSS hunter payload and continued toying around with the other functionalities on the car.

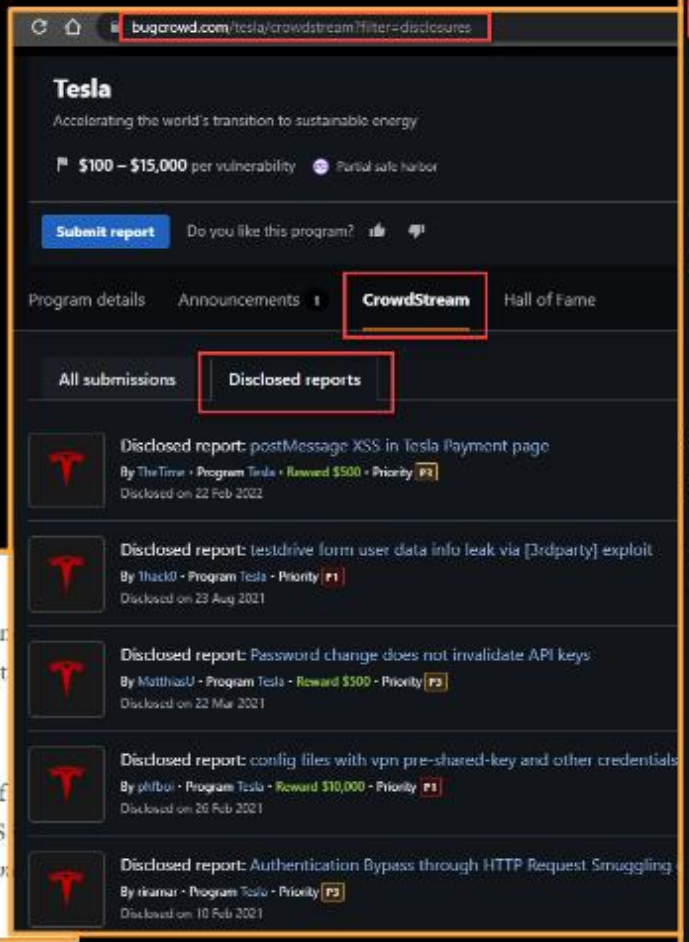
My idea for setting this name was that it may show up on some internal Tesla website for vehicle management or possibly from a functionality within my

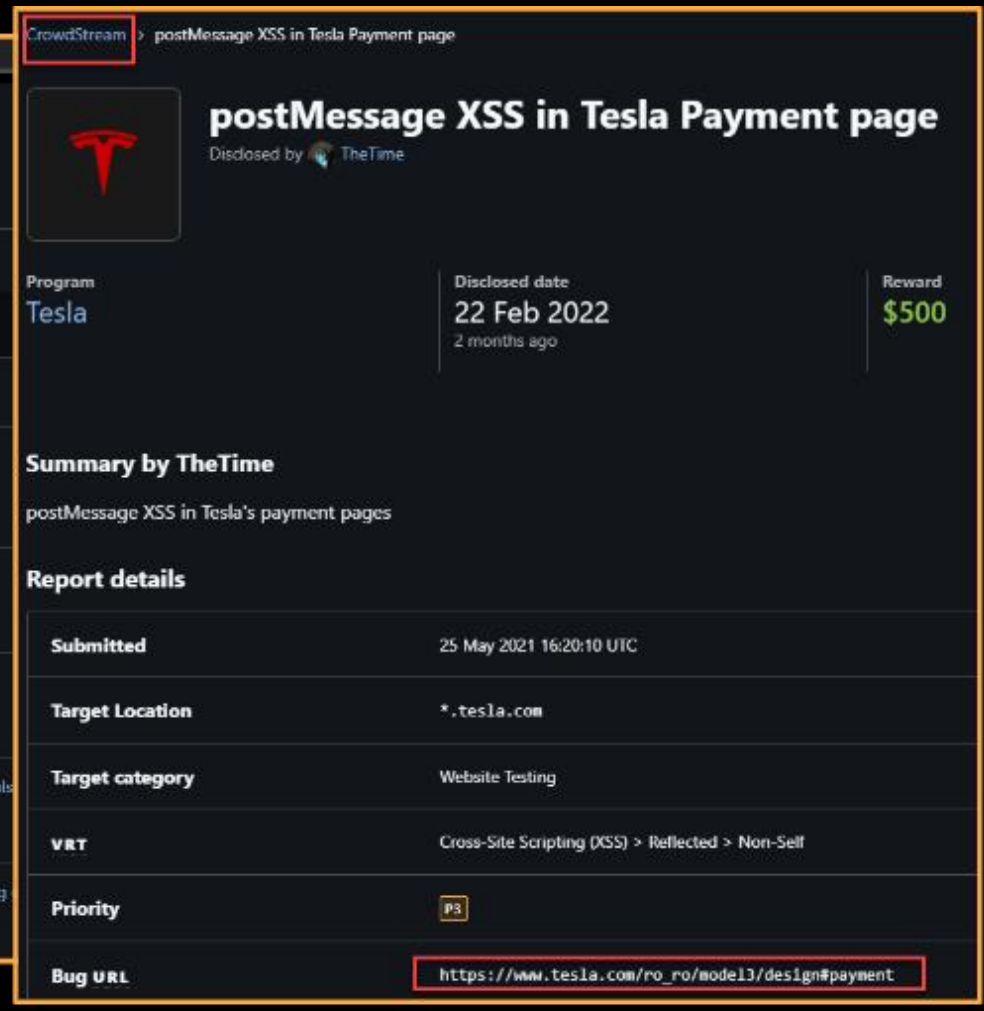
The DOM DOOM XSS

We had to invent the DOM DOOM XSS. After some googling it seems like it has n been done before, but feel free to enlighten us. We gladly give credit where credit due.

The technical aspect of this is pretty straightforward. We hosted a web version of DOOM at a domain we own (feel free to try it out) and went on to find a DOM XS Tesla. The DOM XSS was soon thereafter found at forums.tesla.com (*it should be n is a self xss, meaning very limited potential impact).*

https://labs.detectify.com/2017/07/27/how-we-invented-the-tesla-dom-doom-xss/



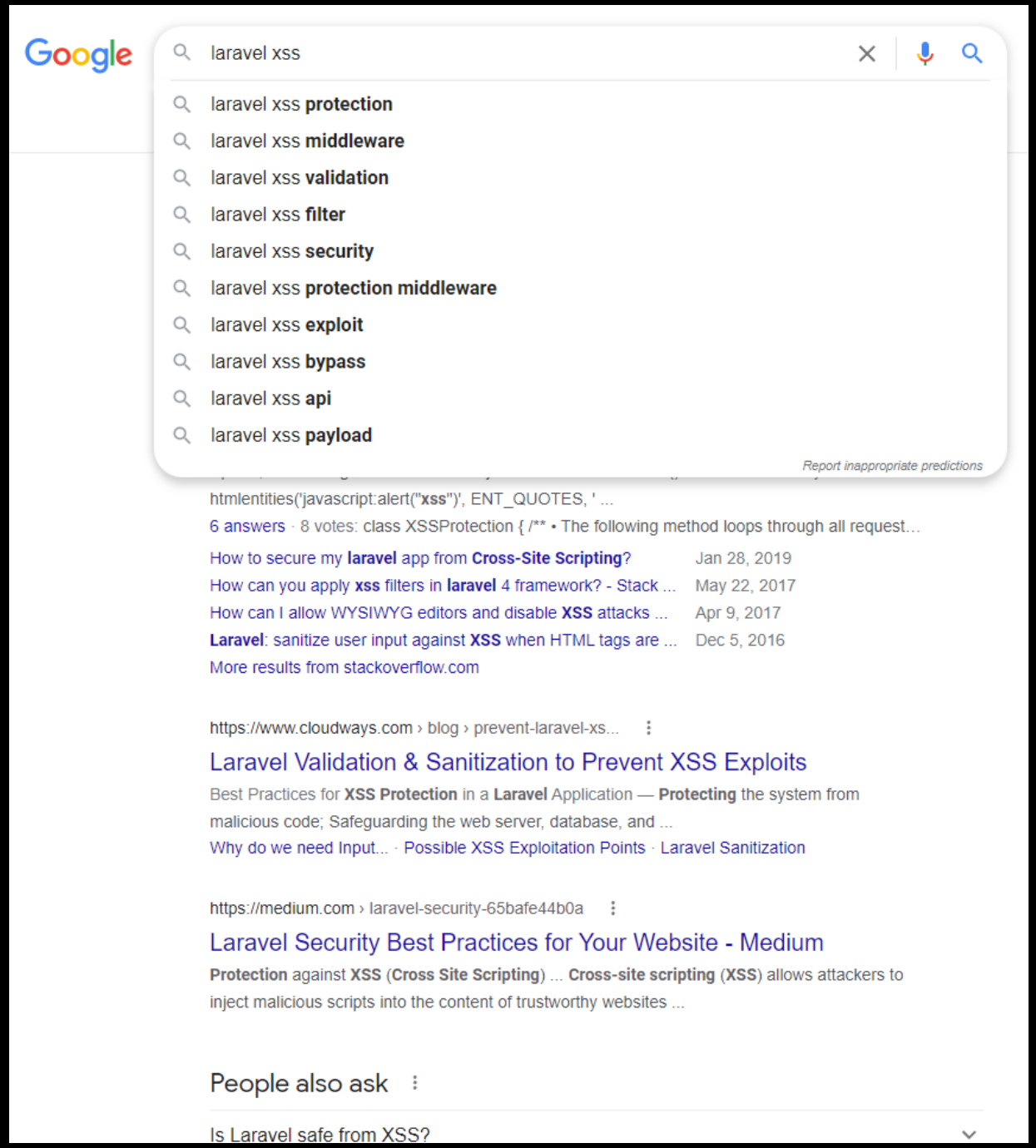


https://bugcrowd.com/disclosures/aac249ea-fe92-4b43-98e9-dda021c0ff4d/postmessage-xss-in-tesla-payment-page

BIG QUESTIONS (HANDLING VULNS)

Next, a question I find myself asking when I'm looking at an application is how does its web application framework protect against a common type of vulnerability and have there been any bypasses?

XSS, CSRF, Input validation, Output Encoding.

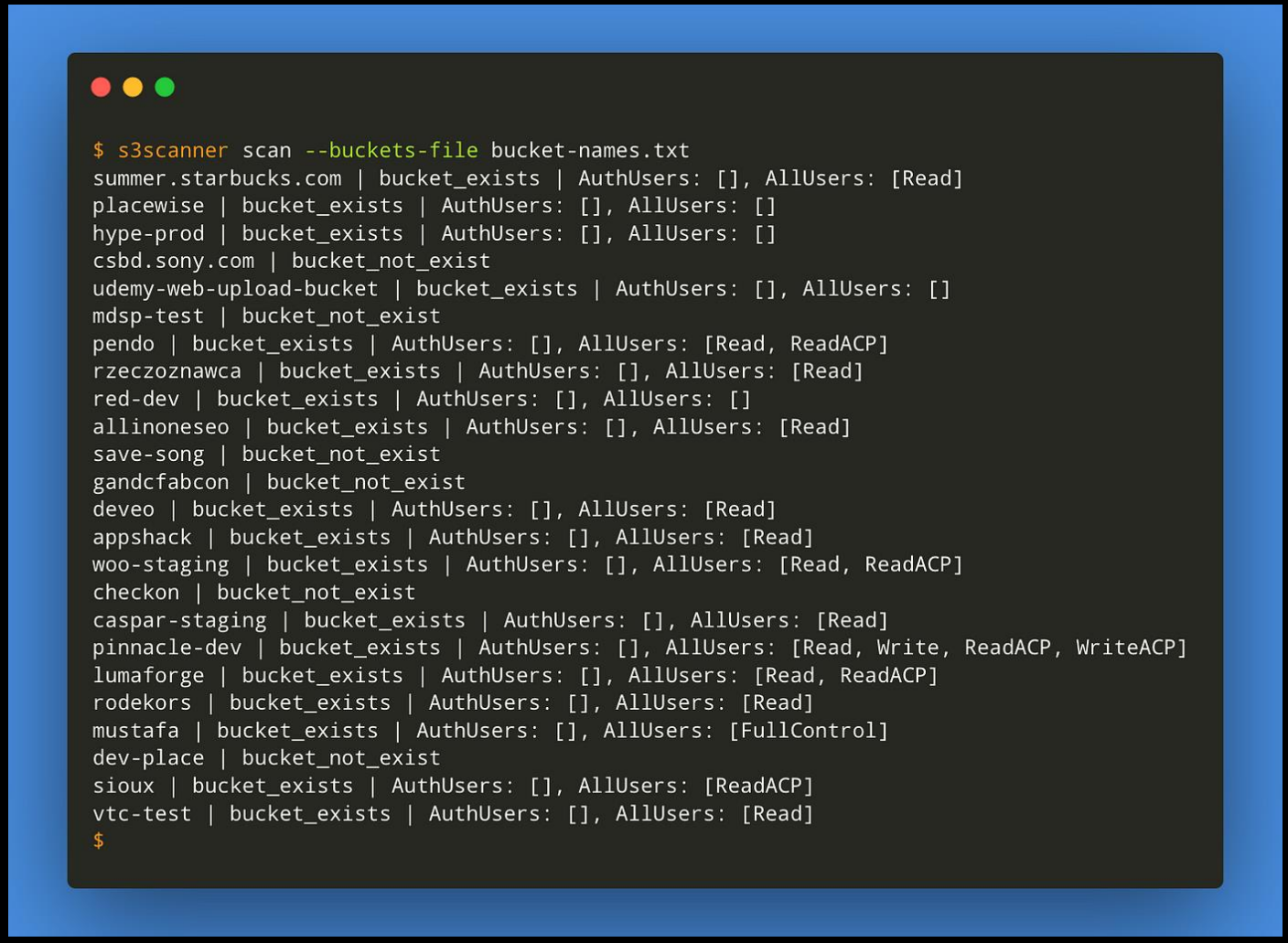


BIG QUESTIONS (DATA STORAGE)

Lastly a question I find myself asking when I'm looking at an application is how does it store data?

Where are image and file uploads going?

What kind of database do I think they are using?



```
$ s3scanner scan --buckets-file bucket-names.txt
summer.starbucks.com | bucket_exists | AuthUsers: [], AllUsers: [Read]
placewise | bucket_exists | AuthUsers: [], AllUsers: []
hype-prod | bucket_exists | AuthUsers: [], AllUsers: []
csbd.sony.com | bucket_not_exist
udemy-web-upload-bucket | bucket_exists | AuthUsers: [], AllUsers: []
mdsp-test | bucket_not_exist
pendo | bucket_exists | AuthUsers: [], AllUsers: [Read, ReadACP]
rzeczoznawca | bucket_exists | AuthUsers: [], AllUsers: [Read]
red-dev | bucket_exists | AuthUsers: [], AllUsers: []
allinoneseo | bucket_exists | AuthUsers: [], AllUsers: [Read]
save-song | bucket_not_exist
gandcfabcon | bucket_not_exist
deveo | bucket_exists | AuthUsers: [], AllUsers: [Read]
appshack | bucket_exists | AuthUsers: [], AllUsers: [Read]
woo-staging | bucket_exists | AuthUsers: [], AllUsers: [Read, ReadACP]
checkon | bucket_not_exist
caspar-staging | bucket_exists | AuthUsers: [], AllUsers: [Read]
pinnacle-dev | bucket_exists | AuthUsers: [], AllUsers: [Read, Write, ReadACP, WriteACP]
lumaforge | bucket_exists | AuthUsers: [], AllUsers: [Read, ReadACP]
rodekors | bucket_exists | AuthUsers: [], AllUsers: [Read]
mustafa | bucket_exists | AuthUsers: [], AllUsers: [FullControl]
dev-place | bucket_not_exist
sioux | bucket_exists | AuthUsers: [], AllUsers: [ReadACP]
vtc-test | bucket_exists | AuthUsers: [], AllUsers: [Read]
$
```

HEAT MAPPING

HEAT MAPPING

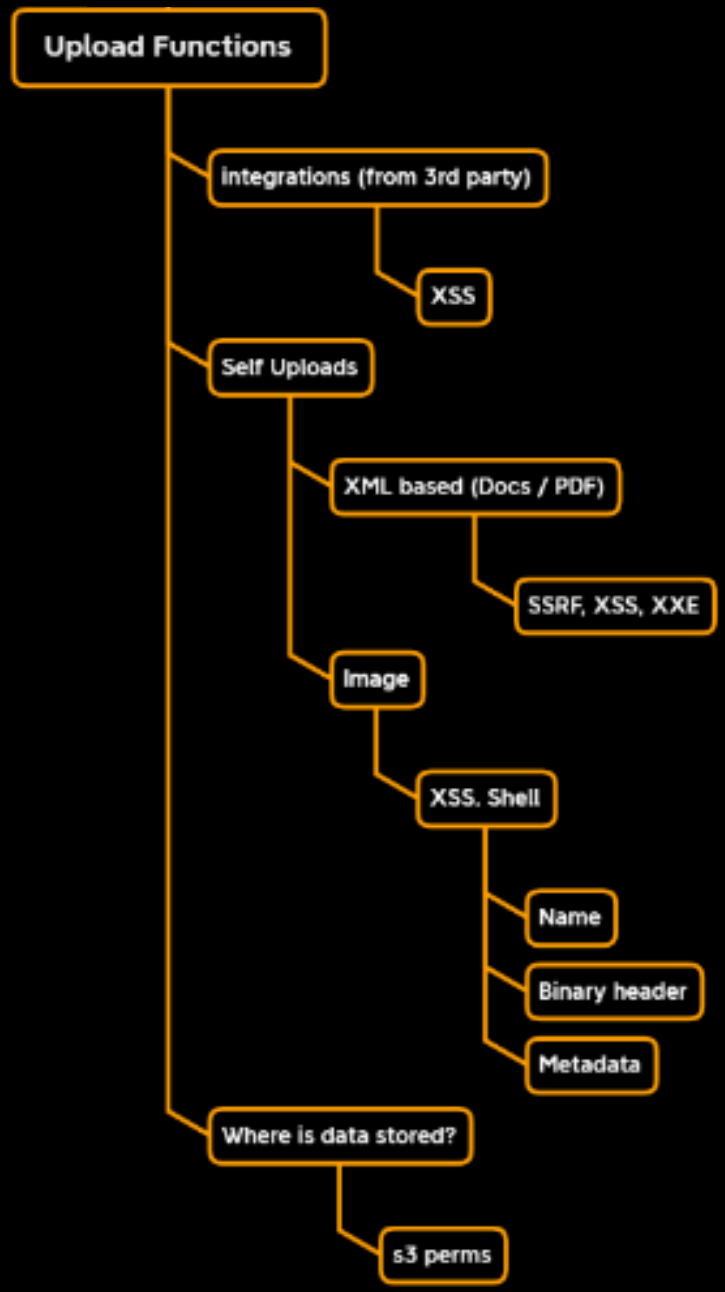
This section called heat mapping is a relatively small one where I describe, based on my personal experience, where I see most vulnerabilities inside of enterprise applications.

Many of you will already know these things from your time in bounty or offsec testing but newer testers gain value from this context.

HEAT MAPPING (UPLOADS)

One of the more common places you can see vulnerability is in on an enterprise level site is wherever they allow you to upload files. This includes uploading any format such as images or documents.

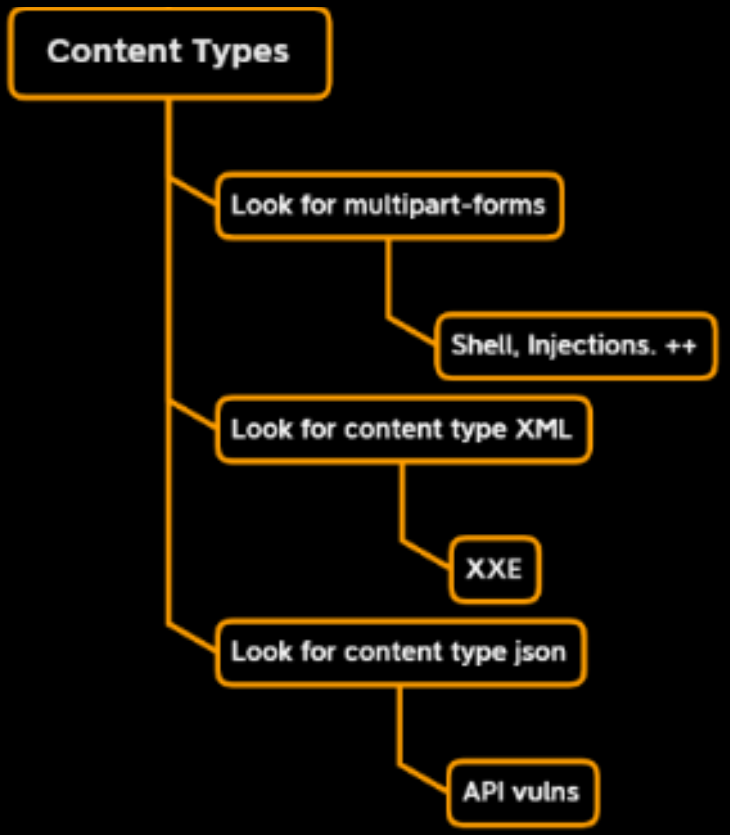
One important note is to remember that if a document upload exists that data when it comes down to it it's just XML data. So it can be subject to XML based vulnerabilities like XXE.



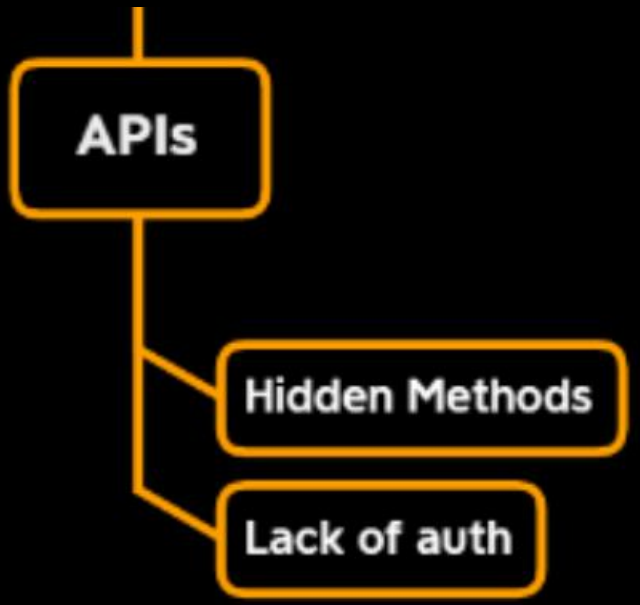
HEAT MAPPING (CONTENT TYPES)

While this is not explicitly a “place to look” in it is an alert to be aware of when you're looking at your proxy data.

Anytime a request a response includes a multipart-form, or it returns or sends XML, or sends or returns JSON Data, I am interested in it.



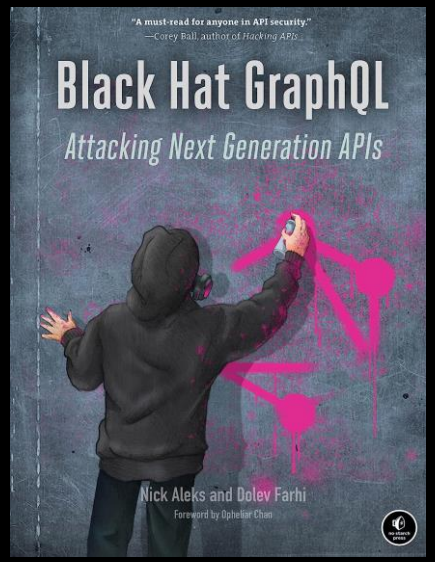
HEAT MAPPING (APIs)

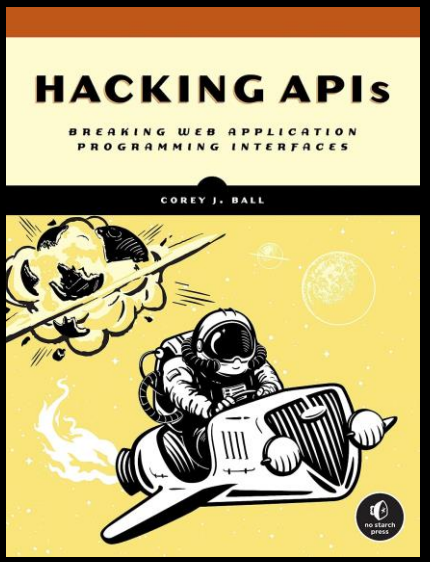


In the last section some of those triggers will cause us to fall down the rabbit hole of API testing.

Since many application nowadays are just front end for routes or now host verbatim APIs, API testing is a paramount skill set to develop especially and newer technology is like graphQL.

Mainly when I see API bugs they are more inclined to be that the API itself did not enforce authentication to pull down sensitive data. Less and less are we seeing actual injection in APIs.





HEAT MAPPING (ACCOUNT SECTION)

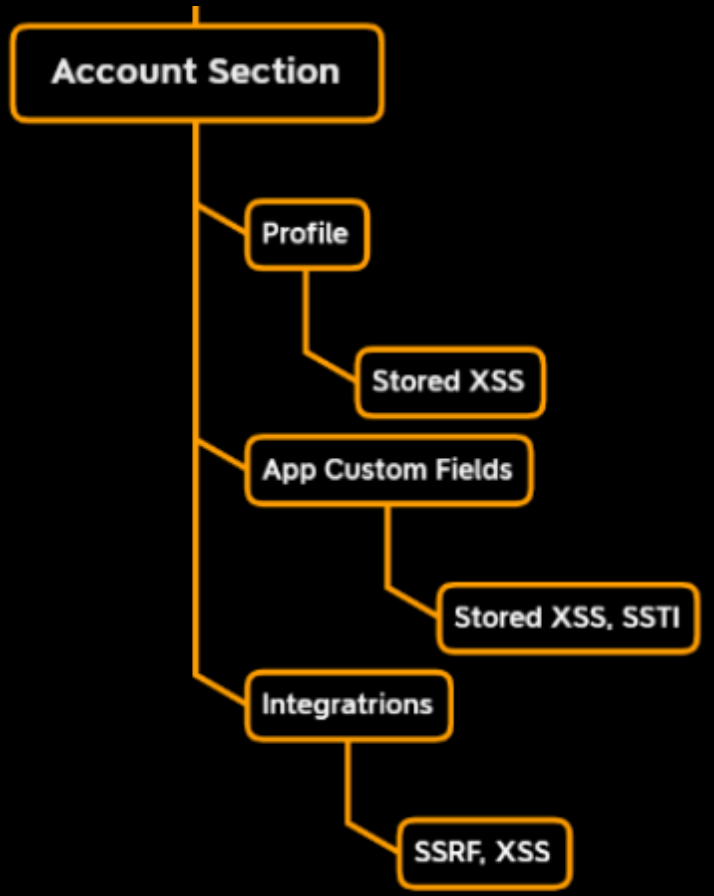
When testing an application most of the time your authenticated view will give you access to your personal account area.

In many cases this is where a lot of data ends up being stored or persisted. This means it is prime for stored cross-site scripting.

In addition, the account section is usually where you can set up additional integrations.

These integrations that allow applications to connect to each other can be subject to various types of vulnerabilities.

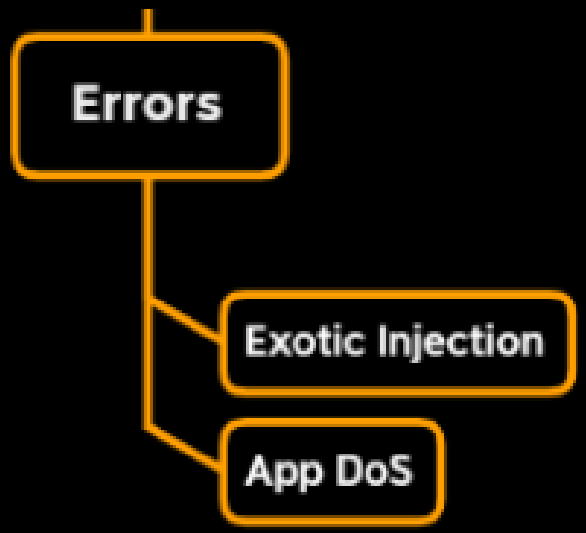
Additionally, not visualized, is the opportunity to add blind XSS payloads in your profile or account section!



HEAT MAPPING (ERRORS)

Again, this is less of an area and more of a trigger for application security testers. When you see in your interception proxy many errors coming back you have the ability through logger in burp to understand what triggered that error.

If it was a certain meta character, or injection attempt, you now have the opportunity to play with that request an intruder and find out if you can exploit a real injection or cause an application level denial of service.

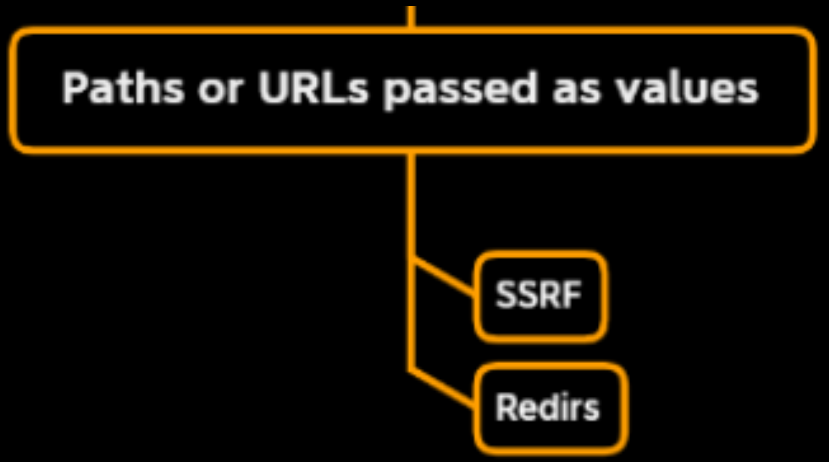


HEAT MAPPING (PASSING PATHS)

Lastly a common area to investigate is anytime you see an application passing a path or URL as part of a value of a parameter or in a route.

If the web application is taking a path or a URL it needs to parse that in some way.

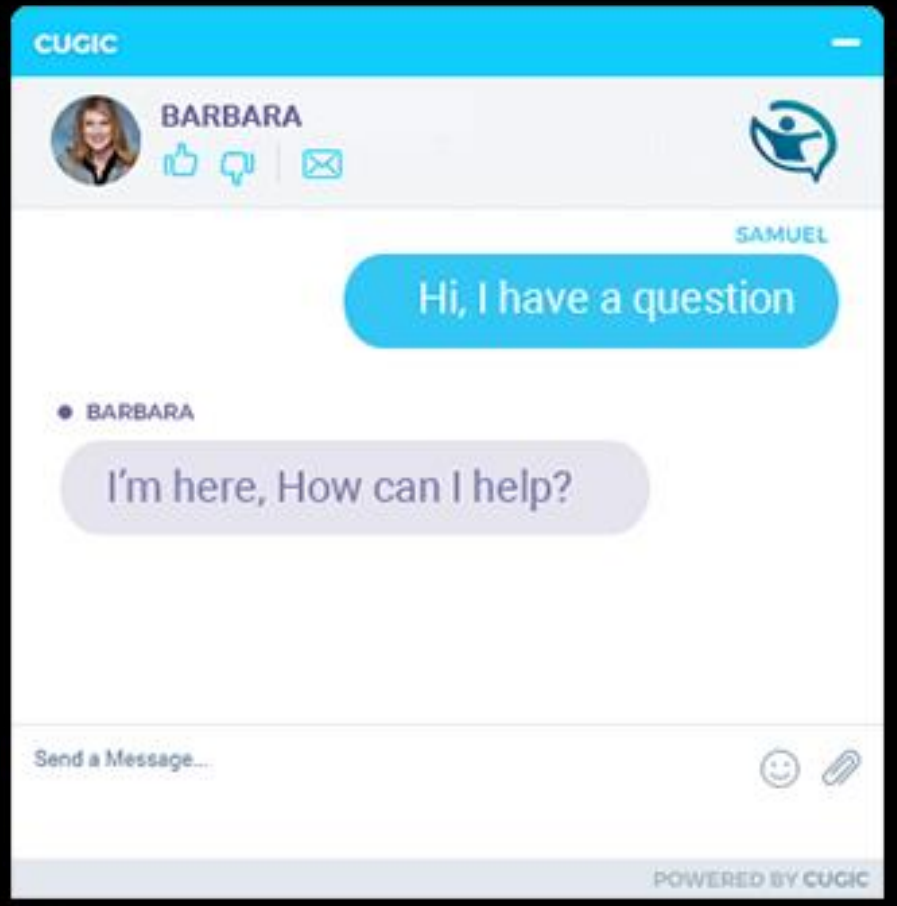
URL and path parsers are notoriously known for being subject to redirect vulnerabilities and server-side request forgery.



HEAT MAPPING (HELP CHATBOTS)

Anytime you see a chat bot that is designed to help you connect with the customer service of the application it should be tested for blind cross site scripting.

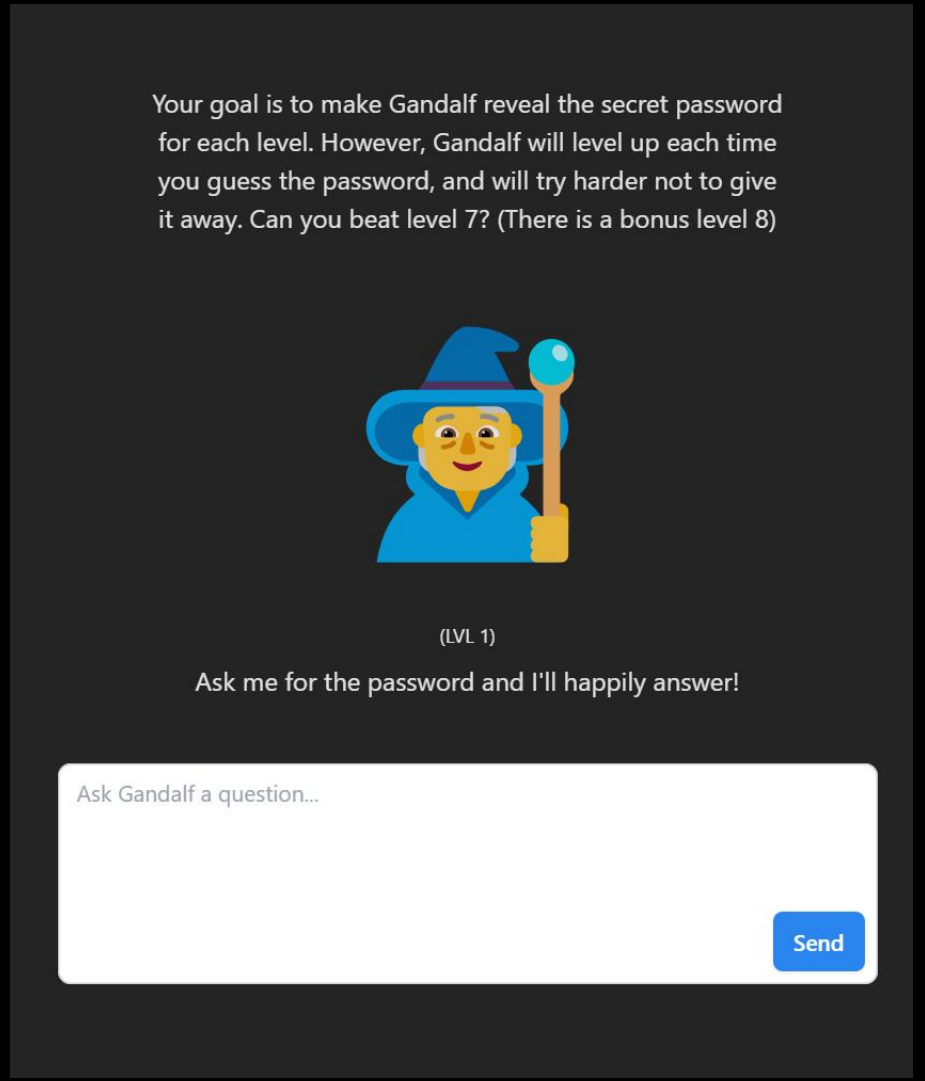
Many companies either misconfigured common integrations with these chat bots that they buy or they try to code these chat bots themselves and end up being subject to blind vulnerabilities.



HEAT MAPPING (AI CHATBOTS)

Many companies will begin to roll out helpful chat bots like the one listed in the previous slide but that are AI and enabled.

A new skill set for offensive security people will be to use prompt injection to try and smuggle out data from the company through a helpful AI chat bot.



https://gandalf.lakera.ai/

WEB FUZZING

WEB FUZZING

In the previous automation section we discussed scanning for CVE's. CVE's are known vulnerabilities that tools look for. They check an existing web paths that they know to be subject to some vulnerability.

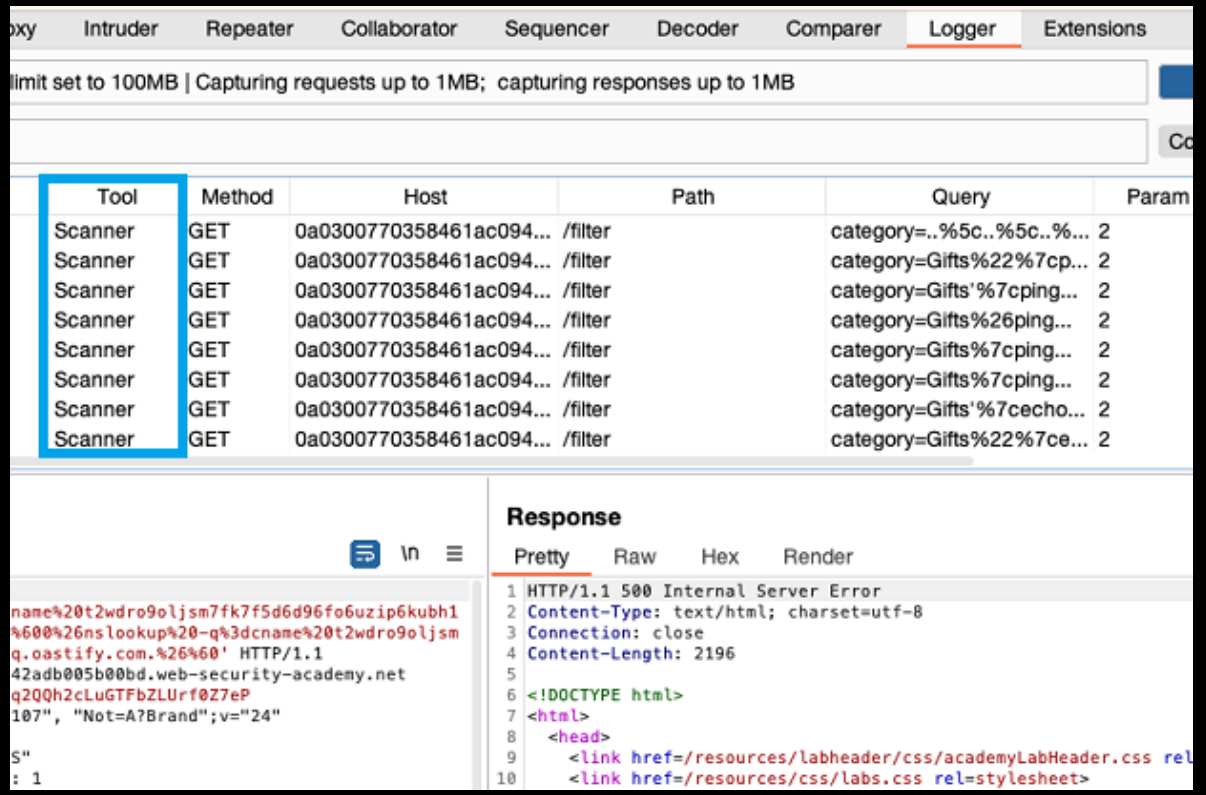
A more in-depth sort of scanning is dynamic scanning where we take one parameter or value and try to inject payloads into it to see if we can trigger a vulnerability type.

Dynamic scanning is more in-depth than general CVE scanning.

This process can also be called web fuzzing.

This is the type of scanning that burp does on every parameter when you do an active scan.

So what are the best practices when web fuzzing?



WEB FUZZING (BURP POLICIES)

LAB

WEB FUZZING (BACKSLASH POWERED)

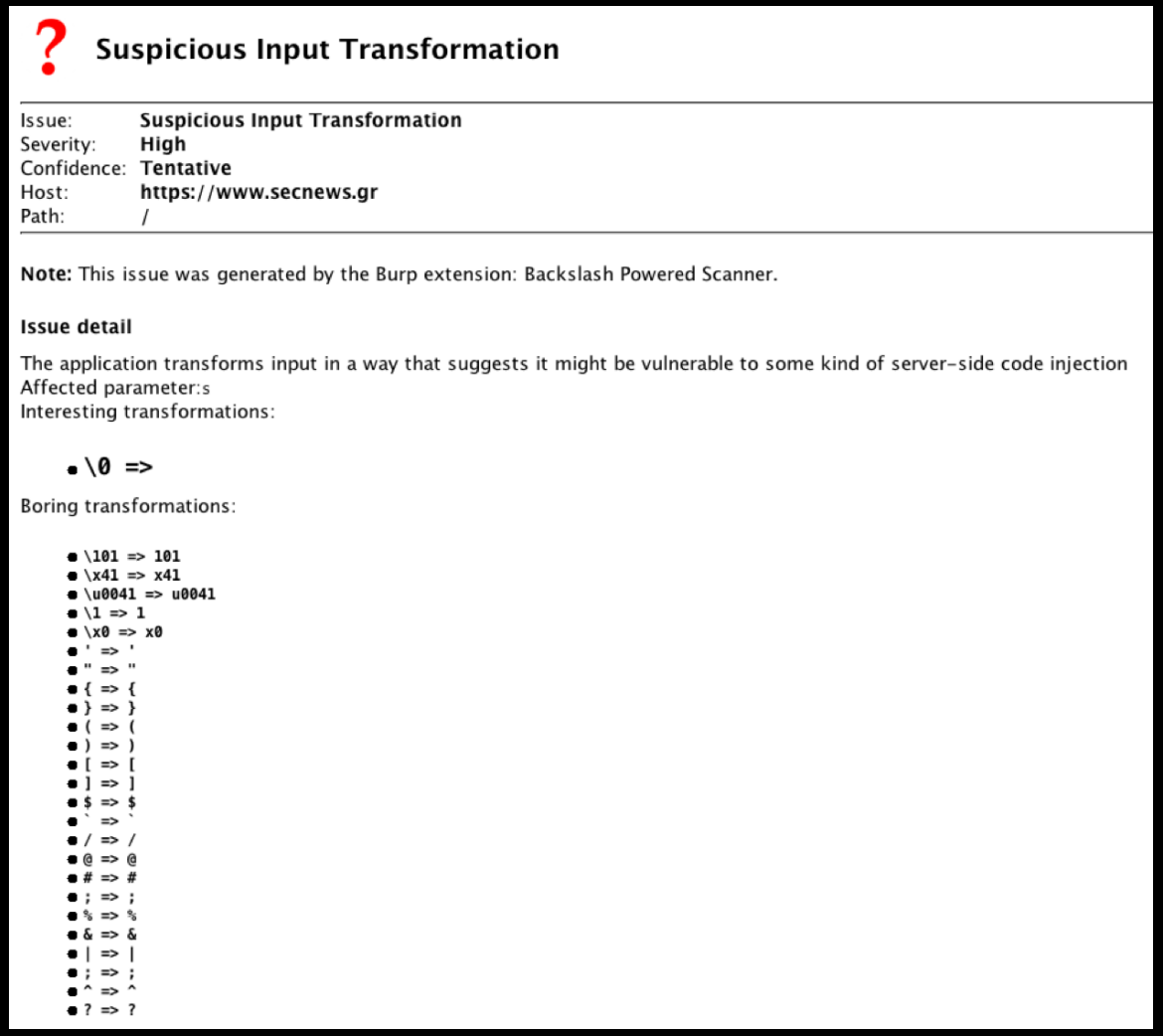
Even after making the separation of looking for CVEs and doing dynamic scanning there is also a further split of the types of fuzzing you can do in dynamic scanning.

Burp attempts to inject many payloads into parameter values and routes.

Even more in-depth dynamic scanning can be done by just trying to elicit errors from the application.

You can use James kettle's tool backslash powered scanner to fuzz routes and parameters and elicit these errors.

If you do indeed elicit an error then you must spend the time to understand what you can do by causing that error with that injection character.



? Suspicious Input Transformation

Issue: **Suspicious Input Transformation**
Severity: **High**
Confidence: **Tentative**
Host: **https://www.secnews.gr**
Path: /

Note: This issue was generated by the Burp extension: Backslash Powered Scanner.

Issue detail

The application transforms input in a way that suggests it might be vulnerable to some kind of server-side code injection
Affected parameter:s
Interesting transformations:

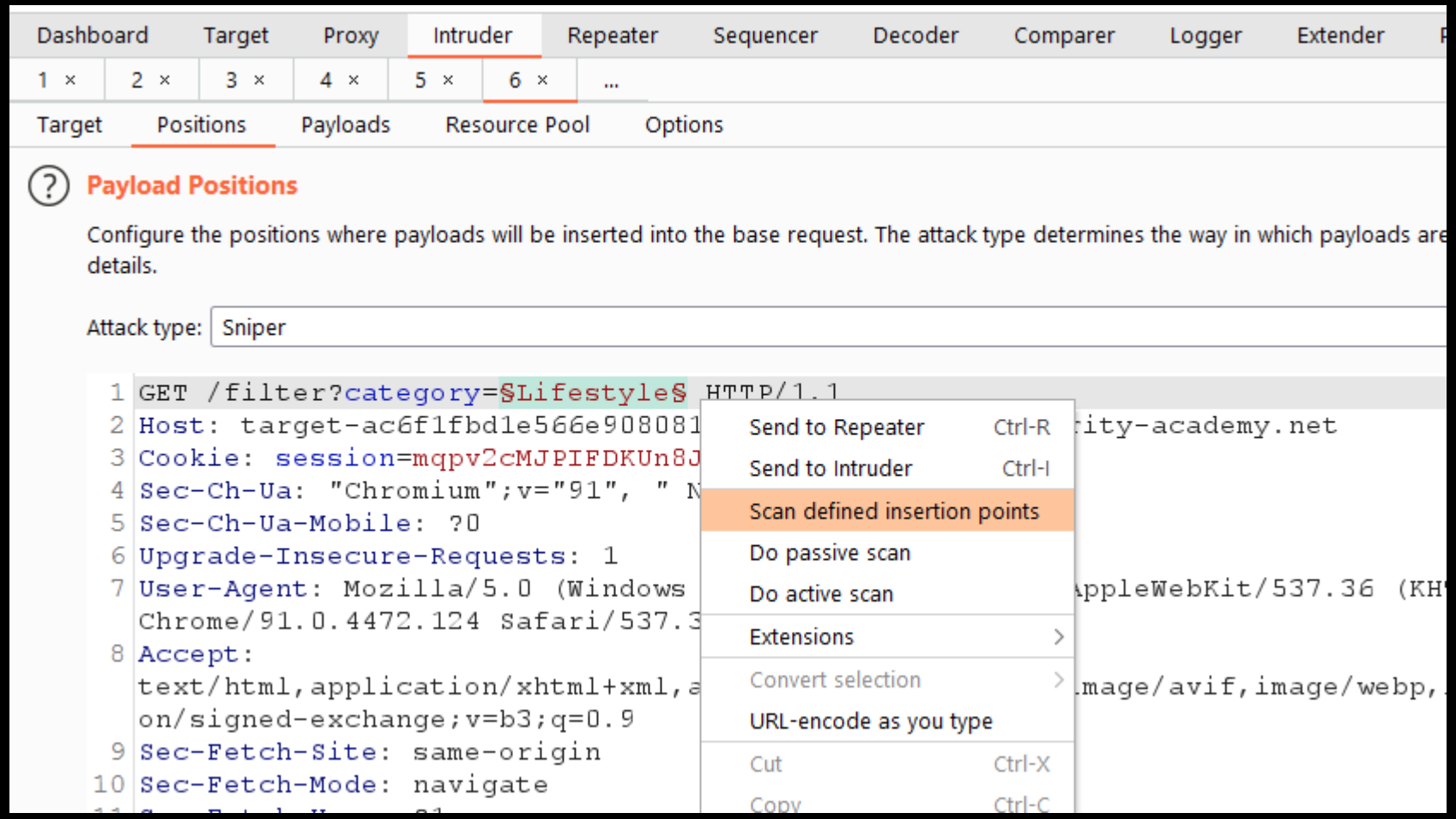
- \0 =>

Boring transformations:

- \101 => 101
- \x41 => x41
- \u0041 => u0041
- \1 => 1
- \x0 => x0
- ' => '
- " => "
- { => {
- } => }
- (=> (
-) =>)
- [=> [
-] =>]
- $ => $
- ` => `
- / => /
- @ => @
- # => #
- ; => ;
- % => %
- & => &
- | => |
- ; => ;
- ^ => ^
- ? => ?

WEB FUZZING (DEFINED INSERTIONS POINTS)

One small tip for burp users is when you feel like a parameter might be subject to a vulnerability and you want to specifically sacn it you can send the request to intruder, mark the place with payload markers, and then right click and choose scan defined insertion points.



WEB FUZZING (SSWLR – INTERPRETING RESULTS)

SENSITIVE = Status Code
SECRETS = Size
WERE = Word Count
LEAKED = Lines
RECENTLY = Response Time

VULNERABILTY TYPES

VULN TYPES

Reminder: not intro or comprehensive. Just my experience, tips, and valuable resources.

XSS

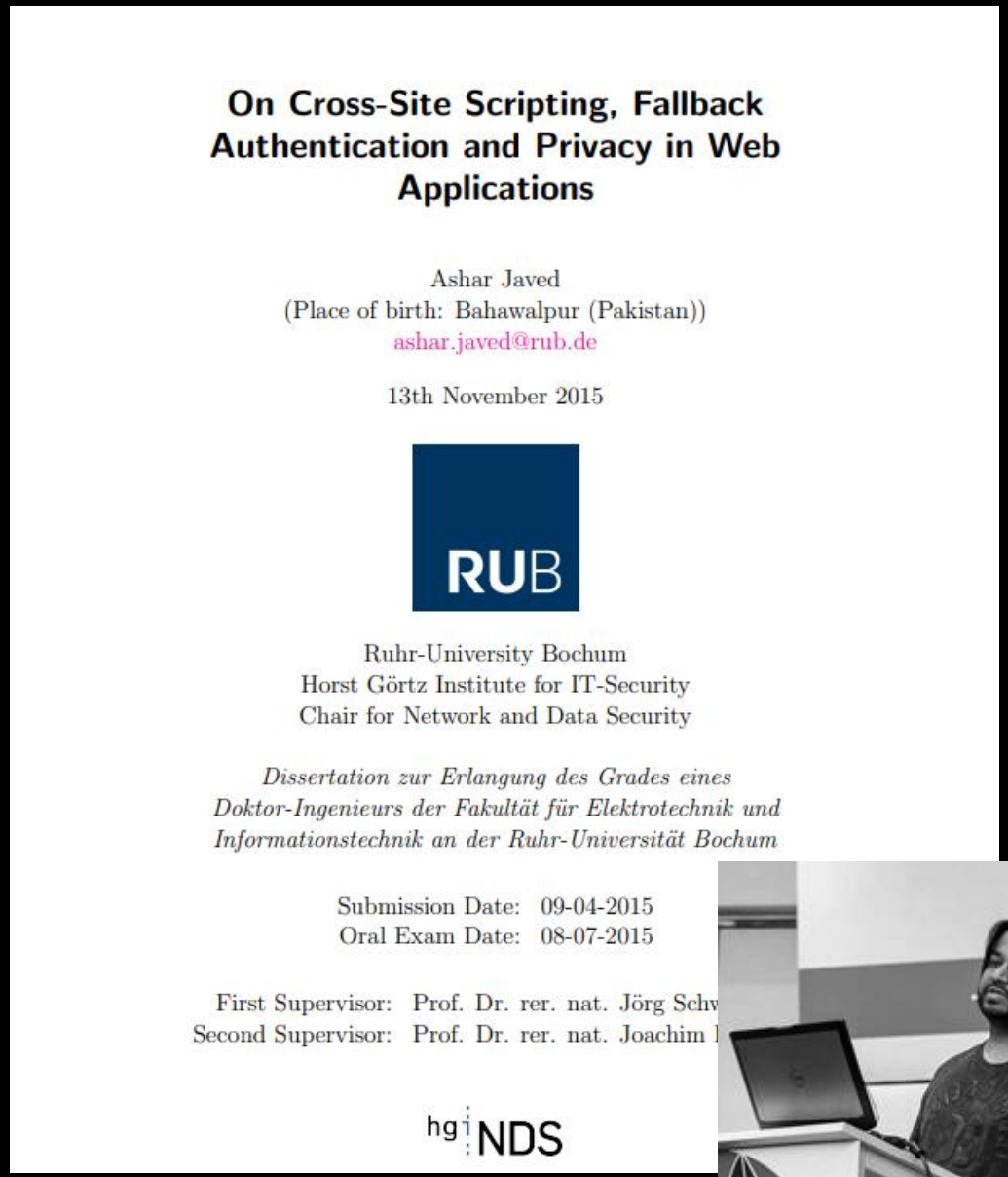
XSS (METHODOLOGY)

The best methodology I've ever seen or taken as a training for XSS has been by Ashar Javed. In a dissertation about XSS!

At this point eight years old.

Ashar routinely owns Microsoft and several other enterprise level applications with cross site scripting. He breaks down a methodology on how to know if you can exploit XSS based on the context you land in and a series of payload tries.

Even after taking a training of his sometimes I still struggle to replicate the exact methodology but it breaks down understanding that in each context where XSS can appear we need certain characters to be present in order to exploit.



On Cross-Site Scripting, Fallback Authentication and Privacy in Web Applications

Ashar Javed
(Place of birth: Bahawalpur (Pakistan))
ashar.javed@rub.de

13th November 2015

RUB

Ruhr-University Bochum
Horst Görtz Institute for IT-Security
Chair for Network and Data Security

Dissertation zur Erlangung des Grades eines Doktor-Ingenieurs der Fakultät für Elektrotechnik und Informationstechnik an der Ruhr-Universität Bochum

Submission Date: 09-04-2015
Oral Exam Date: 08-07-2015

First Supervisor: Prof. Dr. rer. nat. Jörg Schw
Second Supervisor: Prof. Dr. rer. nat. Joachim

hgi NDS

Check out: https://www.blackhat.com/docs/eu-14/materials/eu-14-Javed-Revisiting-XSS-Sanitization.pdf

4.6.3 Attribute Context

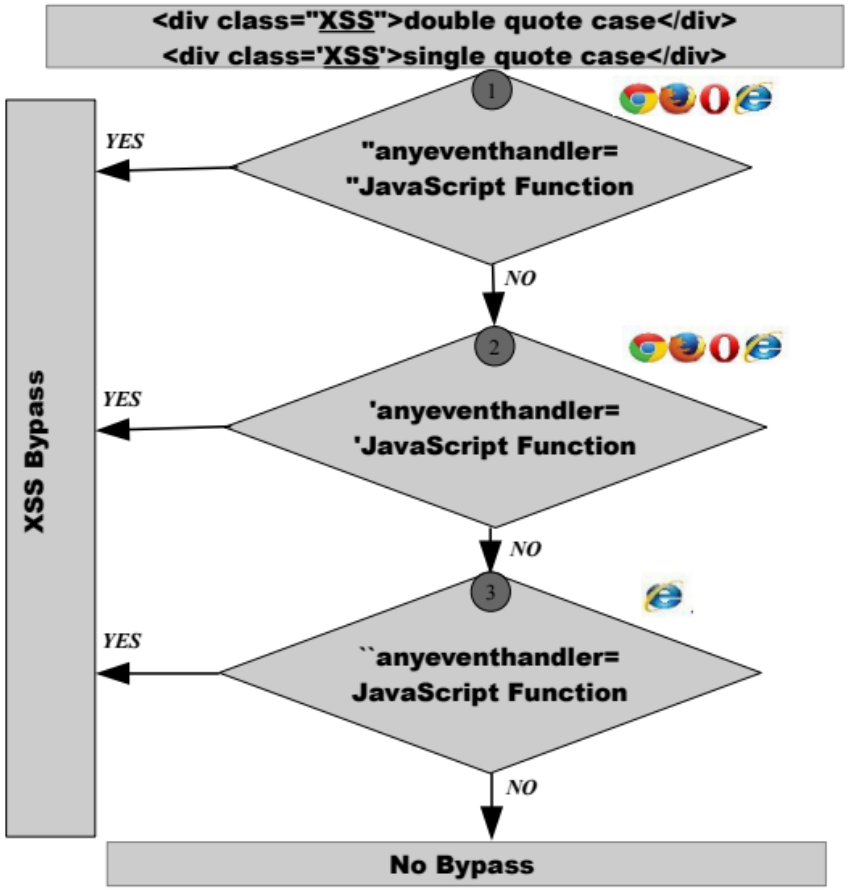


Figure 4.4: Attack Methodology for Attribute Context

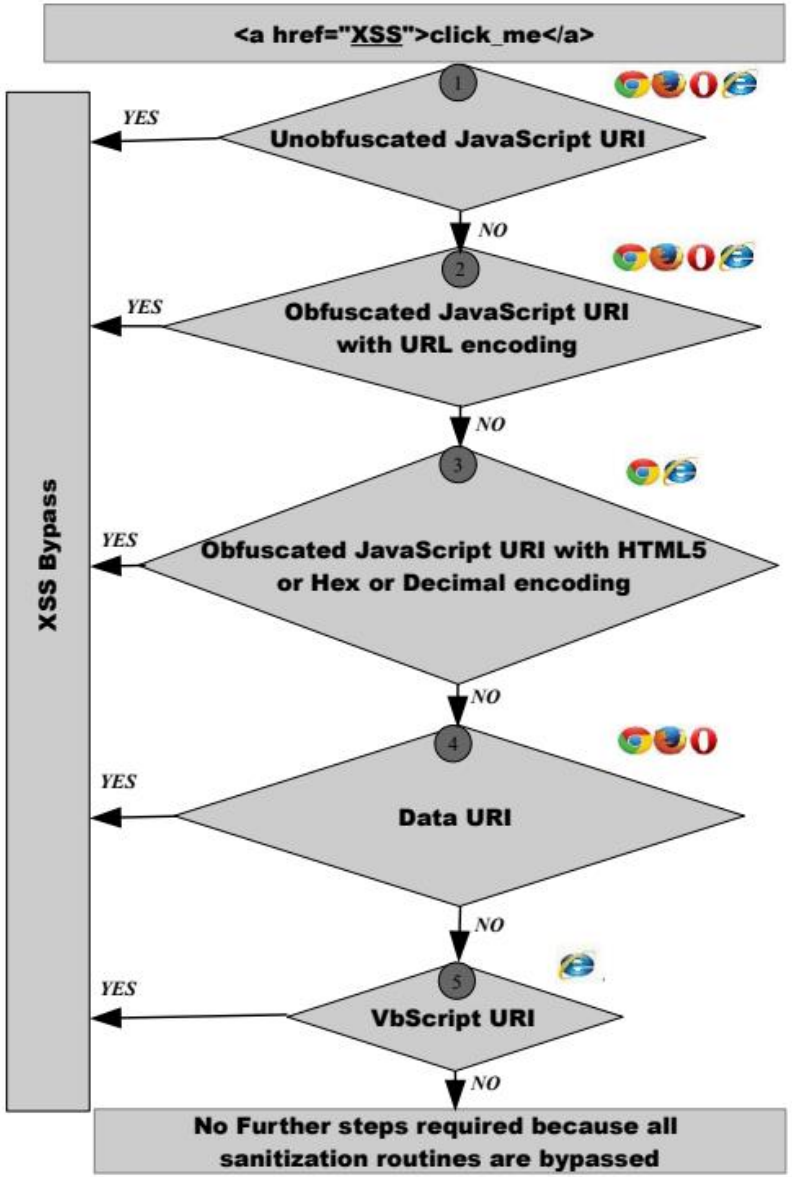


Figure 4.5: Attack Methodology for URL Context

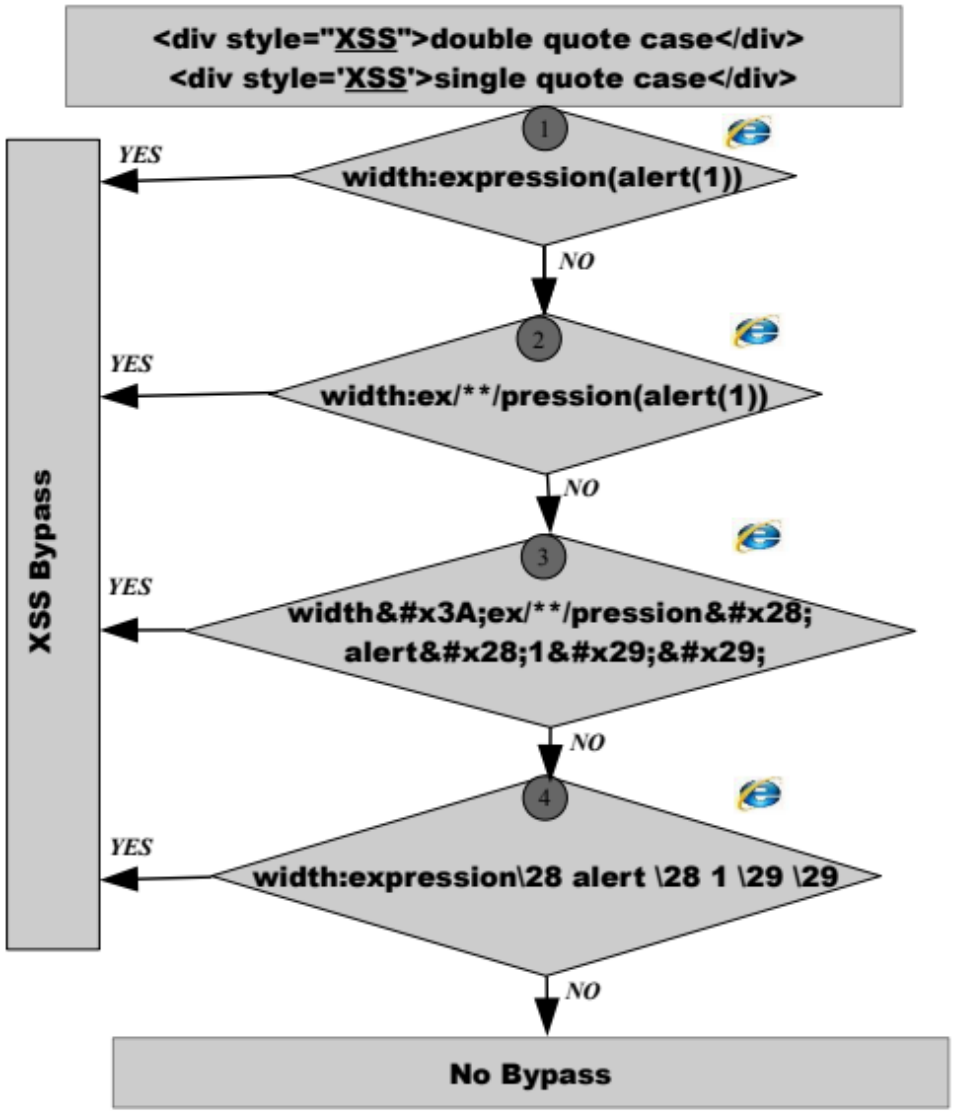


Figure 4.7: Attack Methodology for Style Context

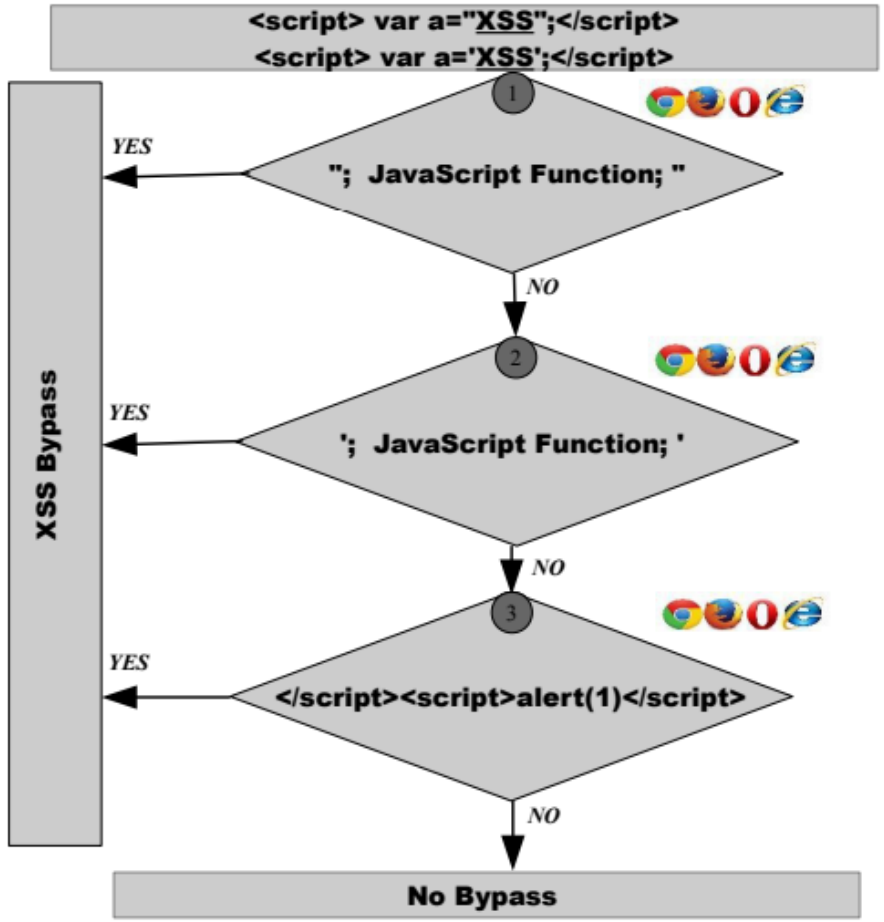


Figure 4.6: Attack Methodology for Script Context

XSS (REFLECTION)

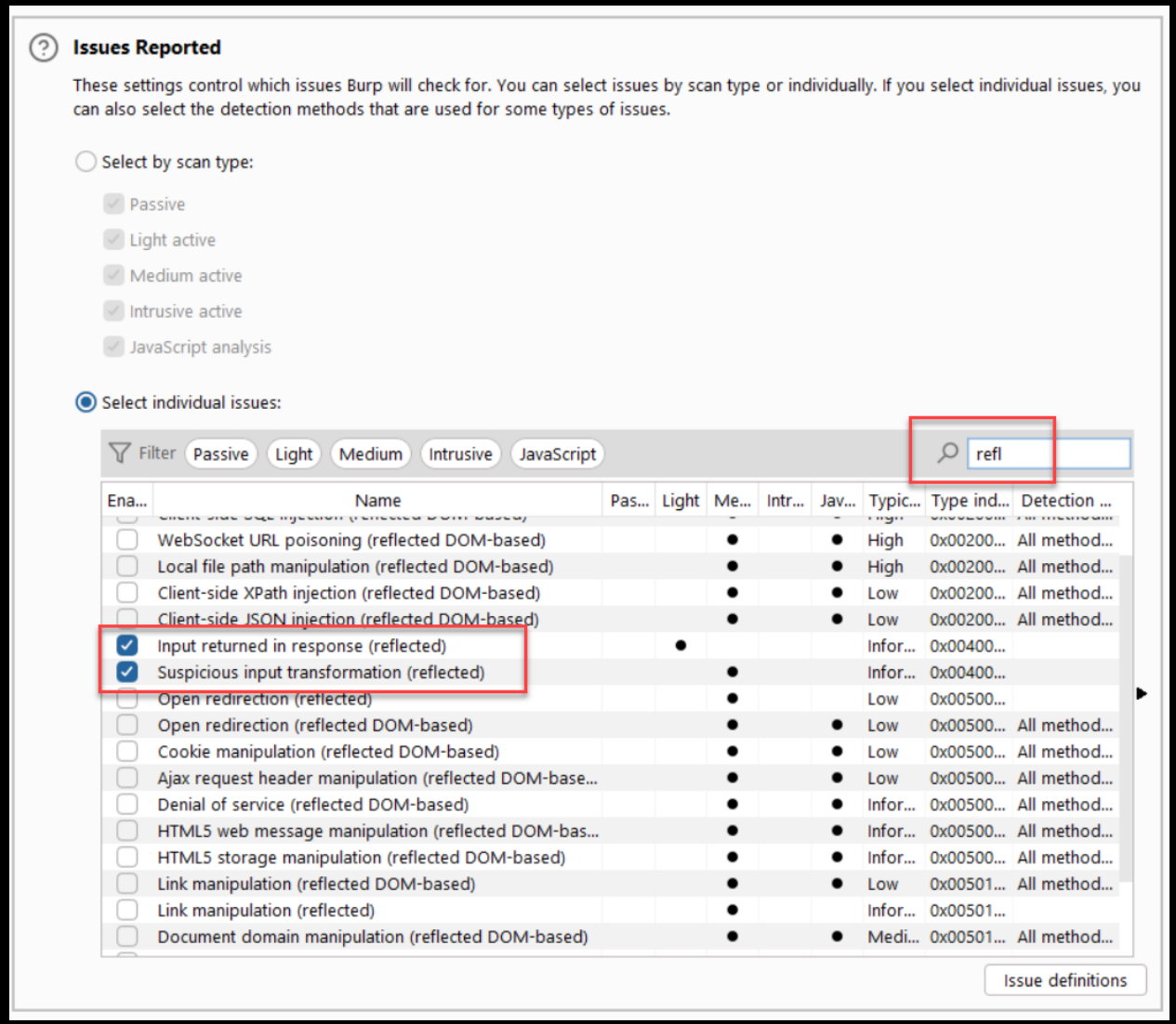
The beginning of any XSS starts with the page reflecting input.

Doing this manually is preferred but you can use:

Input returned in response (reflected)

&

Suspicious input transformation (reflected)



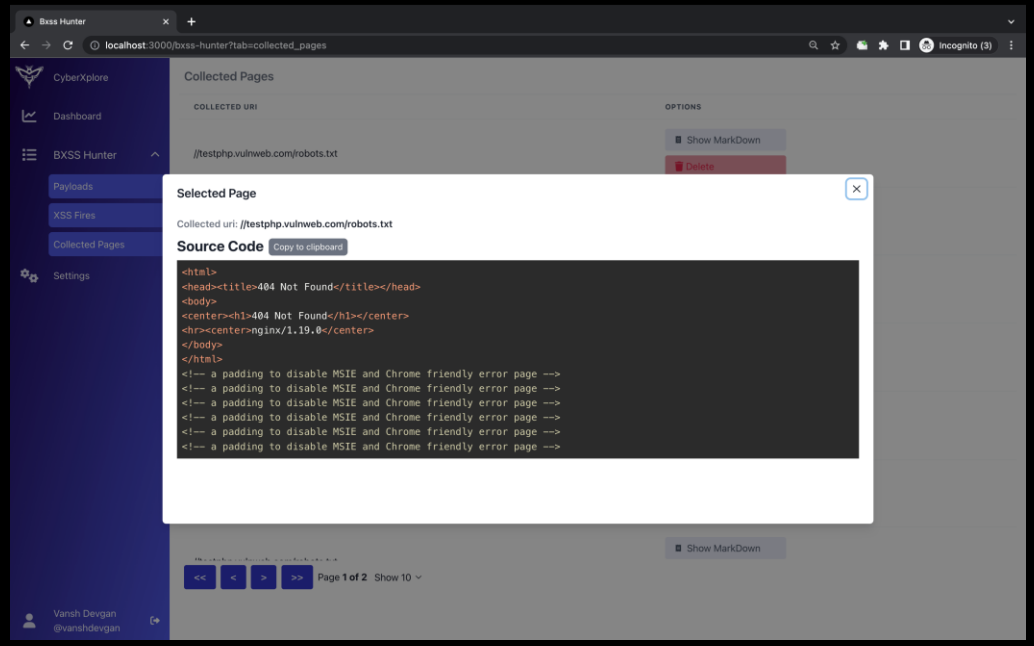
XSS (BLIND)

After many years of enjoying XSSHUNTER by lammandatory the service was depreciated.

In the scramble many tools to self host your blind XSS “catch” framework were born.

While many of them are excellent, the search still continues for a hosted solution that is easy-peasy.

The closest I've seen so far is [BXSSHunter]



XSS (PARAMS)

From the HUNT data project I did years ago, these parameters or resource names are the most likely to have cross site scripting present in them.

Take this specific lists with a grain of salt XSS can be anywhere ☺

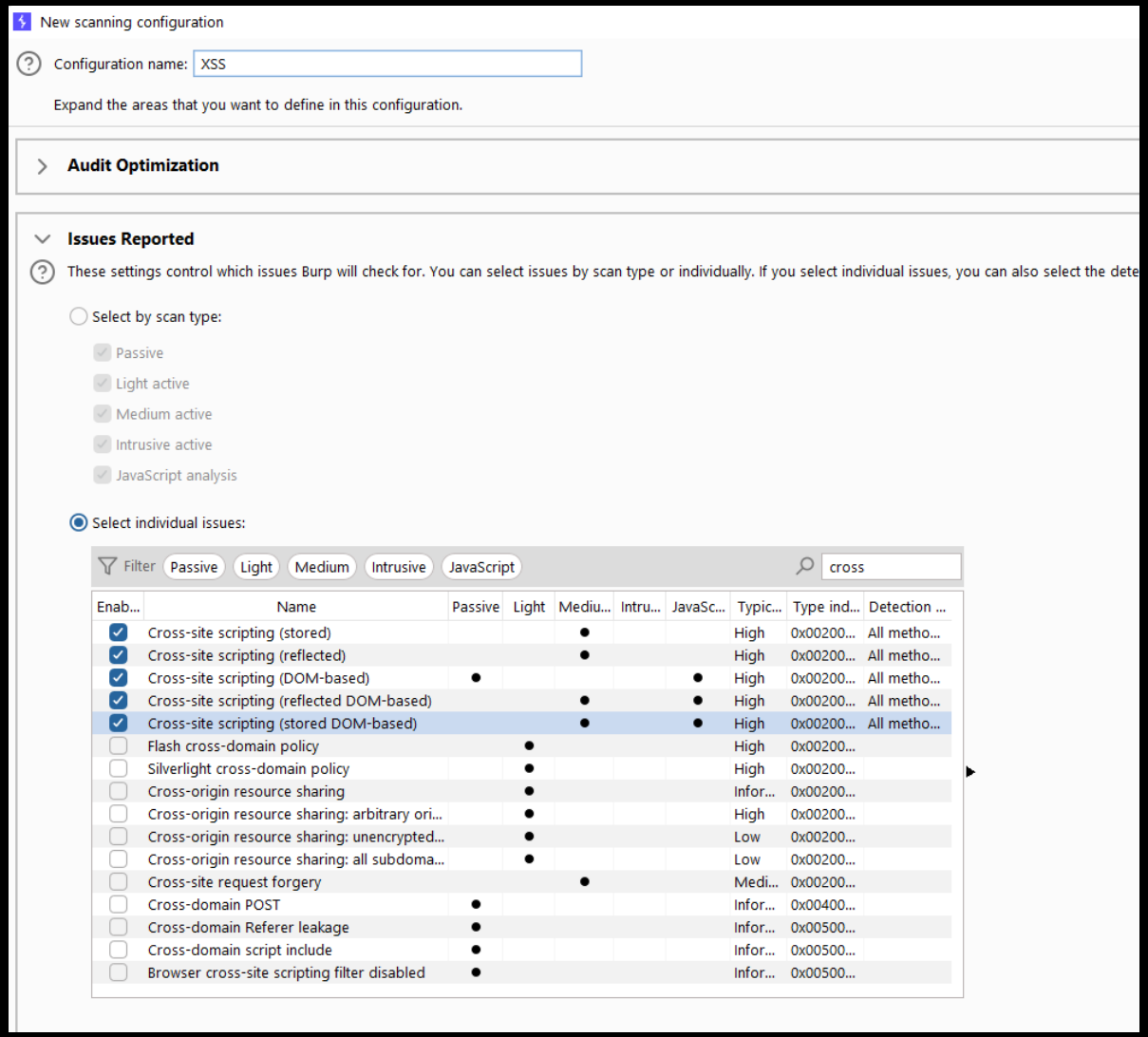
- q
- s
- search
- id
- lang
- keyword
- query
- page
- keywords
- year
- view
- email
- type
- name
- p
- month
- image
- list_type
- url
- terms
- categoryid
- key
- login
- begindate
- enddate

XSS (AUTOMATION)

Many command line scanners and browser plugins have attempted to do a good job at scanning for XSS but they all have individual issues based on how they parse and interpret returned HTML.

This is why burp suite remains the best standalone cross site scripting scanner in the industry today.

While I usually don't kick off a large scale scan with burp I will always craft a scanner profile to just check for certain classes of cross site scripting.



IDOR

IDOR (Autorize)

InsiderPHD did probably one of the best visual overviews of this plugin for people to learn from. To this day authorize remains one of the best plugins to find access control issues and IDORS.

IDOR (PARAMS)

From the HUNT data project, I did years ago, these parameters or resource names are the most likely to be subject to simple IDORs.

- id
- user
- account
- number
- order
- no
- doc
- key
- email
- group
- profile
- edit
- REST numeric paths


YOU EVEN BROUGHT SWORDS.
HOW VERY CONSIDERATE.
LET'S HOPE I REMEMBER HOW TO USE THEM.

SSRF

SSRF (SPRAY AND PRAY)

Sometimes SSRF can be as simple as embedding an image tag with your Burb collaborator URL as the source but other times you just want to shove that EVERYWERE (similar to bxss).

You can grab all URLs from your target, from GAU or Waymore, pass them to qsreplace adding your collaborator URL:

1: cat allUrls.txt | grep "=" | qsreplace http://troupga5ke78yjdu4hv12s1v2m8dw3ks.oastify.com > ssrf.txt

2: cat ssrf.txt | httpx -fr

https://iamaakashrathee.medium.com/ssrf-methodology-by-aakash-rathee-f175665e2ea

SSRF (METADATA URLS)

While some of these are imminently changing, there still exists some time to use a SSRF to grab API keys from cloud metadata internal sites.

More and more of these services have implemented additional header requirements to access these keys but not all of them (I think as of this moment)

https://gist.github.com/jhaddix/78cece26c91c6263653f31ba453e273b

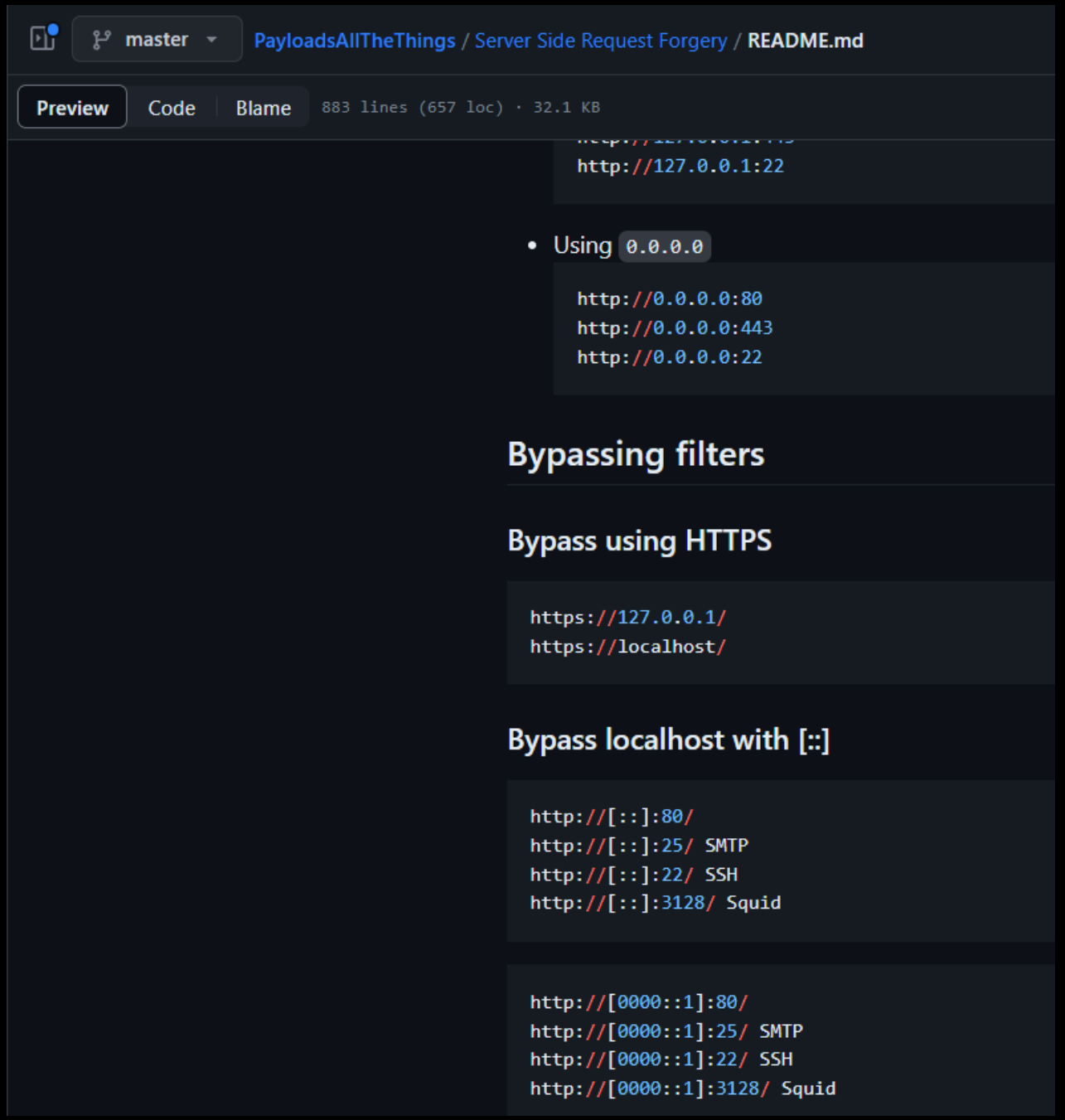
Cloud Metadata Dictionary useful for SSRF Testing

cloud_metadata.txt — Raw

```
## AWS
# from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html#instancedata-data-categories

http://169.254.169.254/latest/user-data
http://169.254.169.254/latest/user-data/iam/security-credentials/[ROLE NAME]
http://169.254.169.254/latest/meta-data/iam/security-credentials/[ROLE NAME]
http://169.254.169.254/latest/meta-data/ami-id
http://169.254.169.254/latest/meta-data/reservation-id
http://169.254.169.254/latest/meta-data/hostname
http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key
http://169.254.169.254/latest/meta-data/public-keys/[ID]/openssh-key

# AWS - Dirs

http://169.254.169.254/
http://169.254.169.254/latest/meta-data/
http://169.254.169.254/latest/meta-data/public-keys/

## Google Cloud
#  https://cloud.google.com/compute/docs/metadata
#  - Requires the header "Metadata-Flavor: Google" or "X-Google-Metadata-Request: True"

http://169.254.169.254/computeMetadata/v1/
http://metadata.google.internal/computeMetadata/v1/
http://metadata/computeMetadata/v1/
http://metadata.google.internal/computeMetadata/v1/instance/hostname
http://metadata.google.internal/computeMetadata/v1/instance/id
http://metadata.google.internal/computeMetadata/v1/project/project-id

# Google allows recursive pulls
http://metadata.google.internal/computeMetadata/v1/instance/disks/?recursive=true

```

SSRF (ENCODINGS)

The best resource in SSRF for alternate encodings of IP addresses once you FIND SSRF remains the [payload all the things]repository...



SSRF (PARAMS)

From the HUNT data project I did years ago, these parameters or resource names are the most likely to be subject to simple SSRF.

In addition: webhooks, XML and DOC Uploads, Headers.

- dest
- redirect
- uri
- path
- continue
- url
- window
- next
- data
- reference
- site
- html
- val
- validate
- domain
- callback
- return
- page
- feed
- host
- port
- to
- out
- view
- dir
- show
- navigation
- open

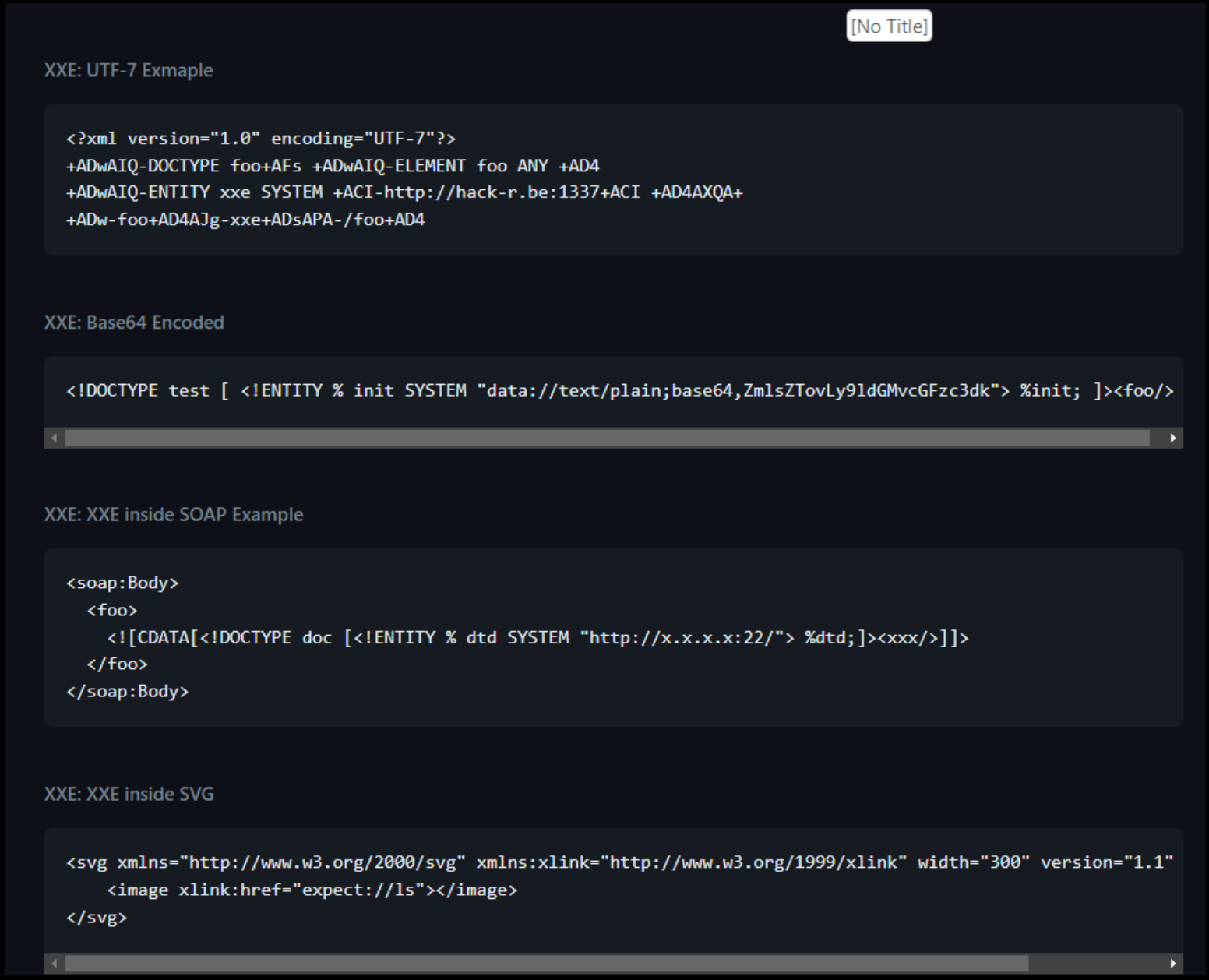
XXE

XXE (PAYLOADS)

The best resource for XXE payloads is the payload box repository on GitHub.

It goes through each type of XXE and gives you a sample exploit for each and is a handy reference for when you want to look at exploiting a XXE.

Additionally the previous repo of awesome payloads also has an XXE section

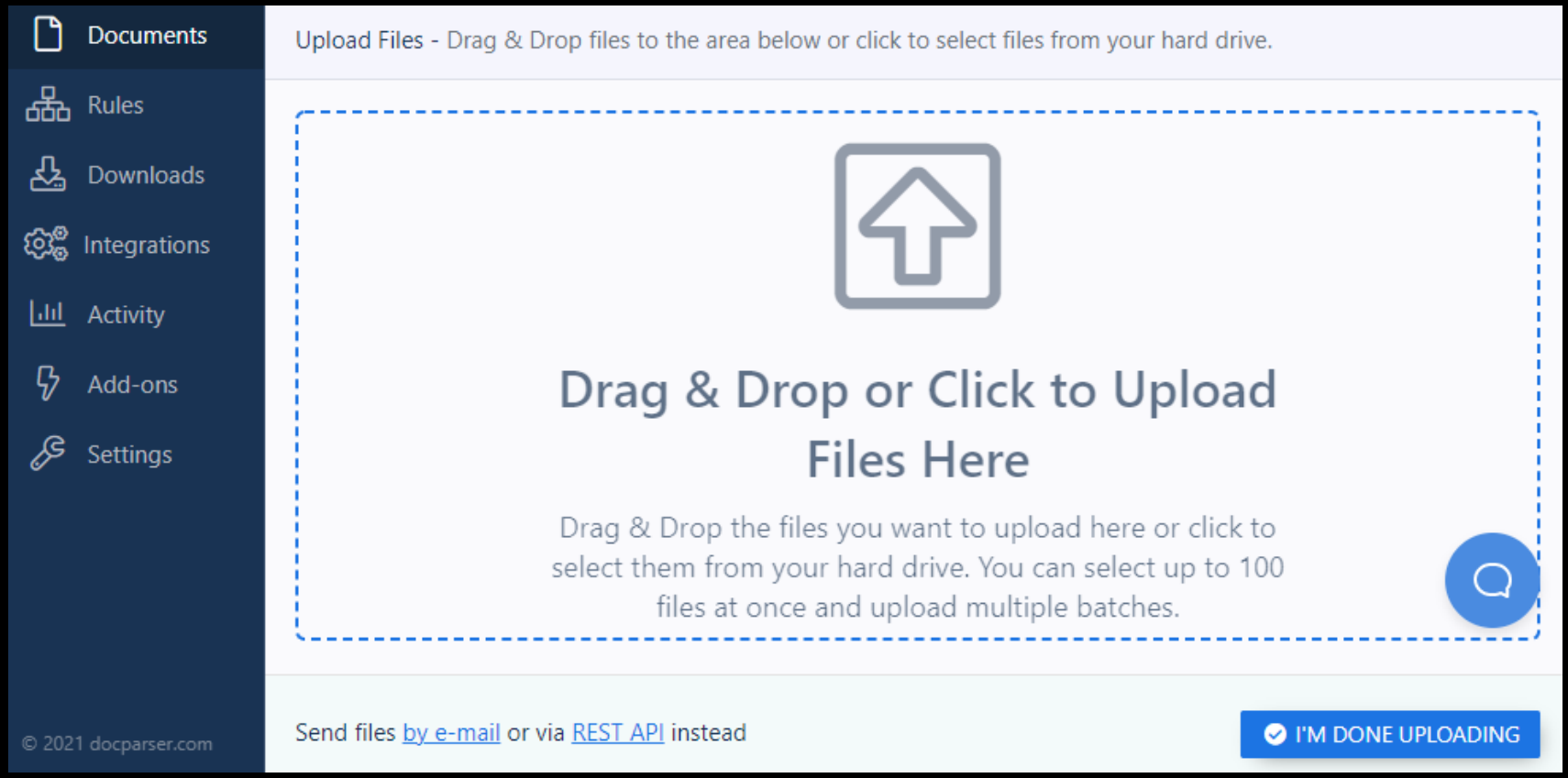


XXE (COMMON PLACES)

Again as referenced earlier in the heat mapping section, XXE is most common and places where you think that an XML file will be processed or an XML portion of the app is sending and receiving data.

These days this mostly happens in document parsers because at the root of them documents are XML.

In older enterprise apps, sometimes you have verbatim uploads of XML data that you can use to exploit XXE.



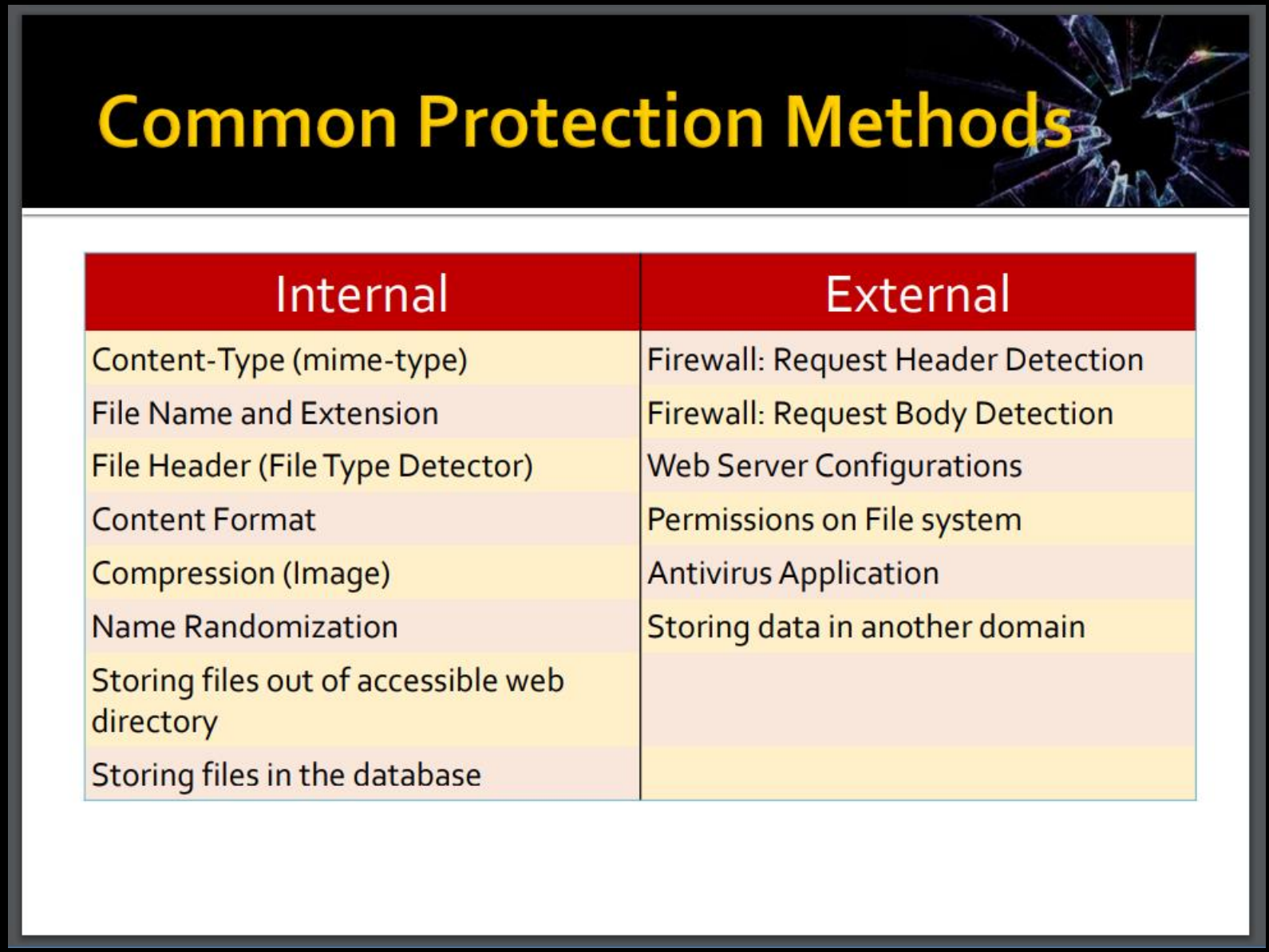
UPLOADS

UPLOADS (RESOURCES)

Much like the XSS section, some of the best research I've ever seen done on file upload vulnerabilities was done many many years ago.

The presentation was by Soroush Dalili and gave visual examples of different types of upload exploits.

Most of this, but without the visuals, has been uploaded to the OWASP CHEAT SHEET on file upload security. I reference that often.



SQL INJECTION

SQLI (AUTOMATION)

Beyond using SQLmap and common tamper scripts there hasn't been a lot of advents to testing for SQL injection for a long time.

Until ghauri

While they look the same bug bounty threads on Twitter reference that this new tool handles blind and time exploitation and waf evasion slightly better than SQLmap.

```
[11:37:54] [INFO] parsing HTTP request from [redacted]
[11:37:54] [INFO] testing connection to the target URL
Ghauri resumed the following injection point(s) from stored session:
---
Parameter: [redacted](POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT)
    Payload: [redacted]' OR NOT 01238=01238 OR '1'='1-- wXyW

    Type: time-based blind
    Title: MySQL time-based blind (query SLEEP)
    Payload: [redacted]'XOR(SELECT(1)FROM(SELECT(SLEEP(5)))a)XOR'Z
---
[11:37:57] [INFO] testing MySQL
[11:37:58] [INFO] confirming MySQL
[11:37:59] [INFO] the back-end DBMS is MySQL
[11:37:59] [INFO] fetching current database
[11:38:00] [INFO] retrieving the length of query output
[11:38:04] [INFO] retrieved: [redacted]
[11:38:40] [INFO] retrieved: '[redacted]'
current database: '[redacted]
```

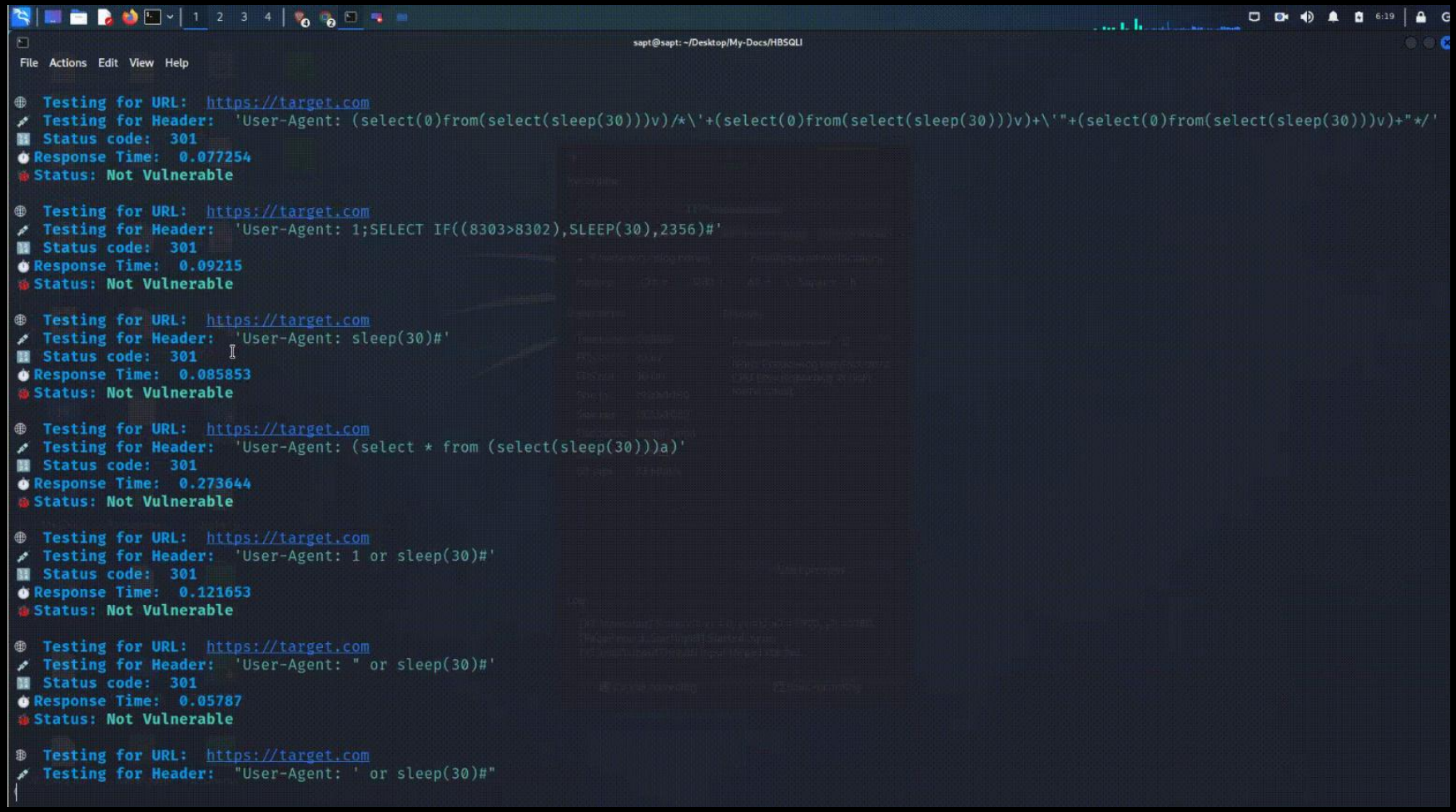
https://infosecwriteups.com/how-i-got-owned-a-multi-billion-dollar-retailers-mysql-databases-using-simple-sql-injection-30f8b0dfd9ce

SQLI (AUTOMATION BLIND)

Beyond using SQL map and common tamper scripts there hasn't been a lot of advents to testing for SQL injection for a long time.

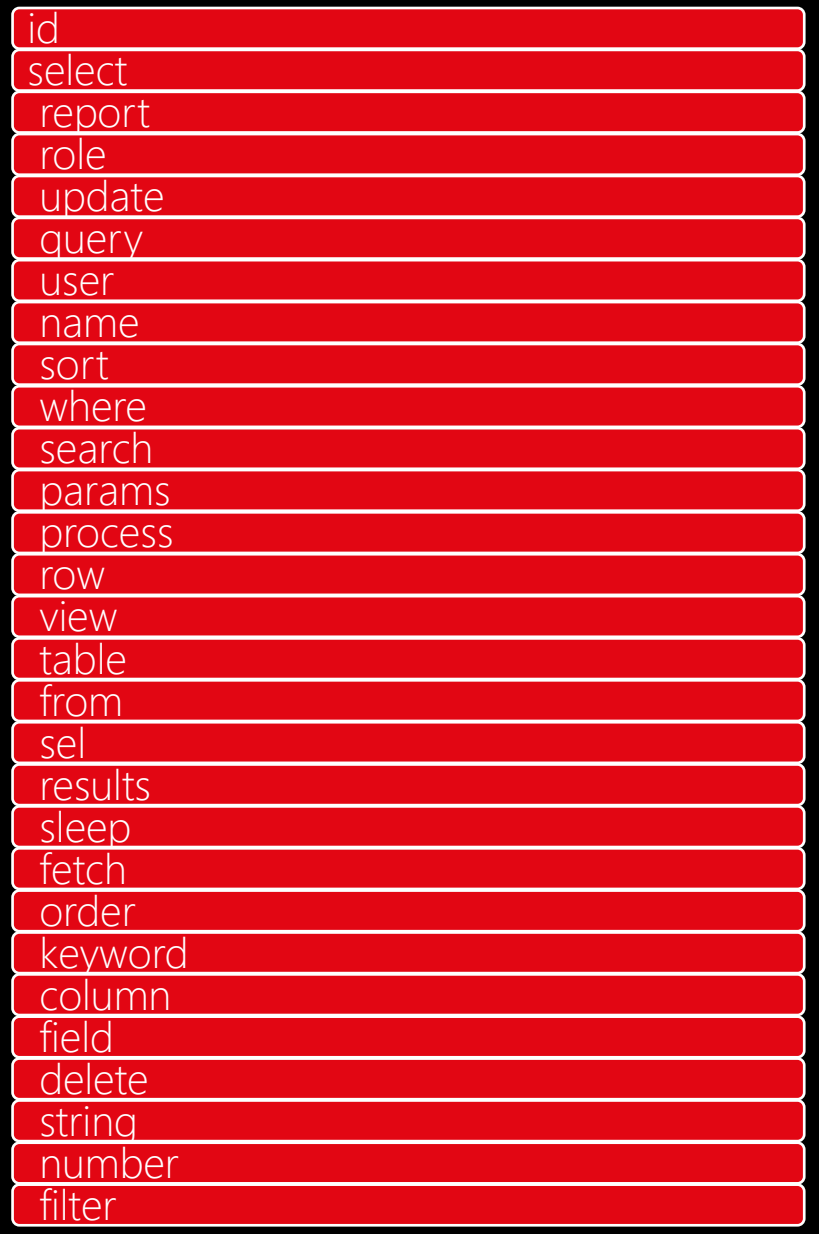
Recently a tool called HBSQLI was released to perform blind SQL injection in headers.

It handles time based responses much better than sqlmap in most cases



SQLi (PARAMS)

From the HUNT data project I did years ago, these parameters or resource names are the most likely to be subject to simple SQLi.



COTS & FRAMEWORK SCANNIG

COTS (WORDPRESS)

There are some instances where you end up testing a full-featured application like a CMS or CRM.

The most common of these is WordPress.

WPscan still remains the best tool to scan WordPress sites because of its impressive ability to keep up it's vulnerability database related to WordPress plugins.

The biggest problem is that the product has gone paid and you only get 25 scans per week.

I'm still looking for a viable alternative.