PNPT

Table of contents

1. PNPT Live

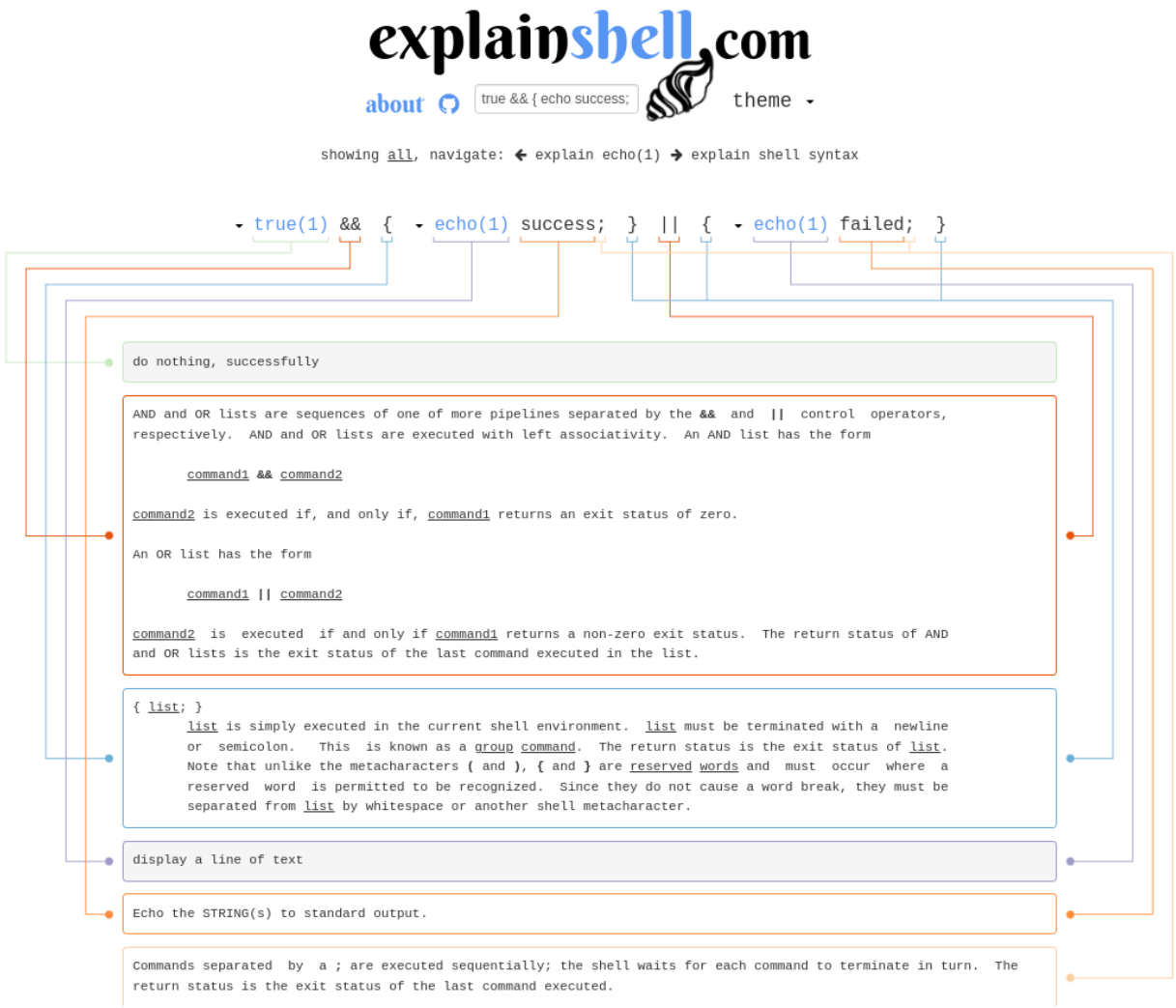
For full documentation visit academy.tcm-com.

2. Networking Refresher

2.1 Linux & Networking

2.1.1 Shell

explaineshell.com



2.1.2 commands mentined duriing lecture

ip a

ip n = arp -a

n=neighbours

ip r

r = route

python3 -m http.server 8080

git

pimpmykali.sh in github

2.1.3 Routing explained

Do you know what's less useful than a screen door on a submarine? A network with no routes. Routing has existed in tandem with networks since the genesis of time, or at least since the 1960's when the concept was invented. Routing as we know it today didn't exist until 1981 when a team of researchers developed the first multiprotocol router. Even though the technology is relatively new, it's highly integrated into our daily lives.

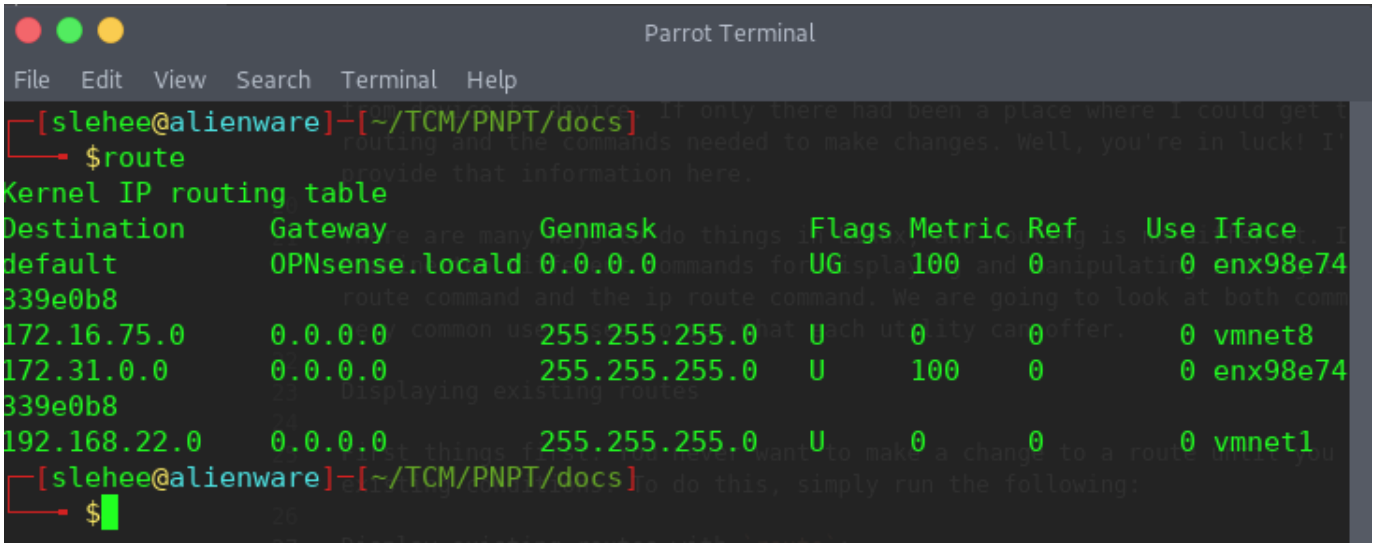
During my time as a support engineer, I found myself troubleshooting network connectivity issues more often than I care to remember. Many times, I would find myself having to engage a senior tech for help because I didn't fully understand the routing from device to device. If only there had been a place where I could get the basics of routing and the commands needed to make changes. Well, you're in luck! I'm going to provide that information here.

There are many ways to do things in Linux, and routing is no different. I want to examine two different commands for displaying and manipulating routing tables: the route command and the ip route command. We are going to look at both commands in some very common use cases to see what each utility can offer.

Displaying existing routes

First things first. You never want to make a change to a route until you verify the existing conditions. To do this, simply run the following:

Display existing routes with :



Display existing routes with ip:

```

```

For a basic listing, I prefer the old school route command, but your mileage may vary.

Adding new routes

At times, you need to add new routes between devices. To do this, use the examples below.

Using :

```

```

Using ip:

```

```

Removing routes

You can remove routes in a similar fashion.

Using route:

The syntax is the same as the add command, except we are using del instead of add.

Using :

Again, we are only altering the syntax slightly from the add command.

Adding a new default gateway

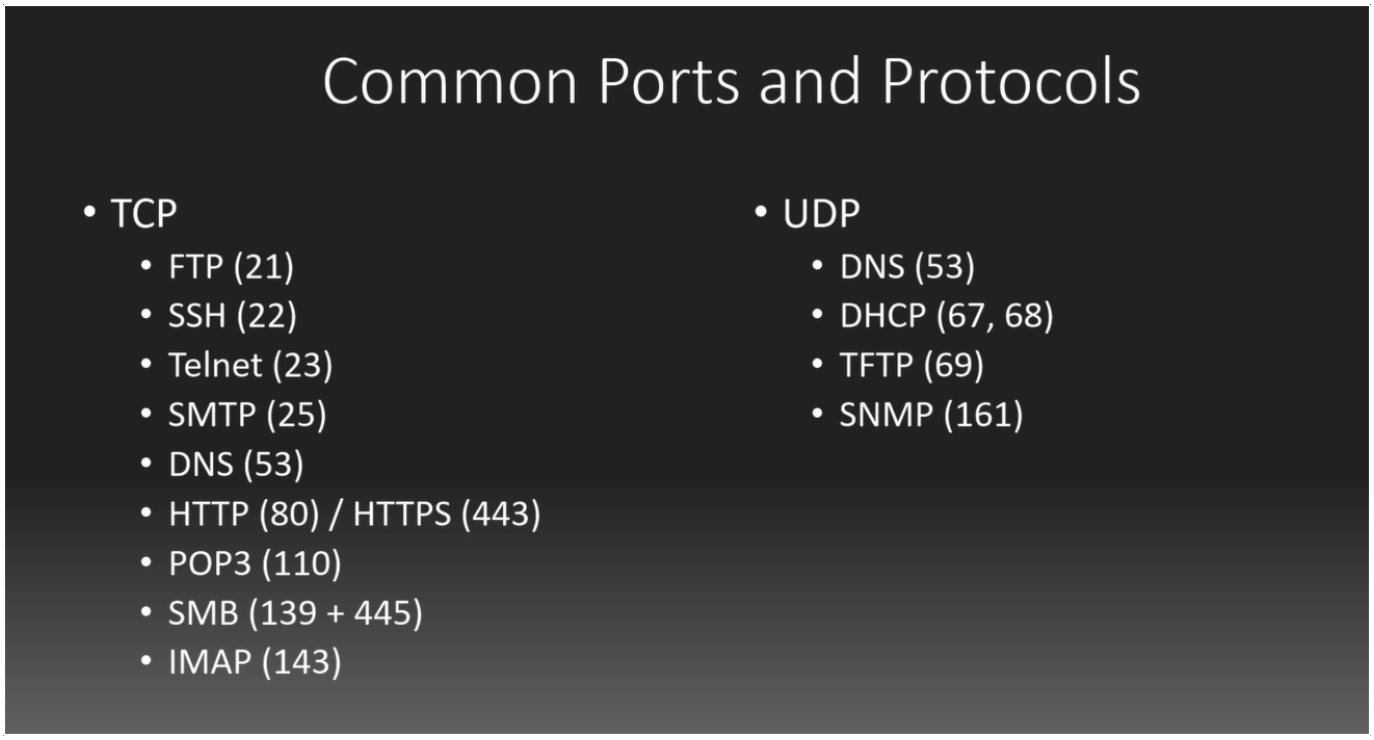
Another task you may need to accomplish is configuring traffic to flow to a gateway. To accomplish this, use the following commands.

Adding a new gateway with route:

Adding a new gateway with ip:

vim s
To verify that the new gateway is set, use the standard route command or ip route show.

2.1.4 Common Ports and Protocols



2.1.5 The OSI Model

lease o ot hrow ausage izza way

1. Physical - data, cables, cat6
2. Data - switching, mac addresses
3. Network - IP addresses, routing
4. Trasport -TCP/UDP
5. Session -session management
6. Presentation - WMV, JPEG, MOV
7. Application - HTTP, SMTP

2.2 Subneting

Seven Second Subnetting video

The Cyber Mentor's Subnetting Sheet

	Subnet x.0.0.0							
CIDR	/1	/2	/3	/4	/5	/6	/7	/8
Hosts	2,147,483,648	1,073,741,824	536,870,912	268,435,456	134,217,728	67,108,864	33,554,432	16,777,216
	Subnet 255.x.0.0							
CIDR	/9	/10	/11	/12	/13	/14	/15	/16
Hosts	8,388,608	4,194,304	2,097,152	1,048,576	524,288	262,144	131,072	65,536
	Subnet 255.255.x.0							
CIDR	/17	/18	/19	/20	/21	/22	/23	/24
Hosts	32,768	16,384	8,192	4,096	2,048	1,024	512	256
	Subnet 255.255.255.x							
CIDR	/25	/26	/27	/28	/29	/30	/31	/32
Hosts	128	64	32	16	8	4	2	1
Subnet Mask (Replace x)	128	192	224	240	248	252	254	255

Notes:
*Hosts double each increment of a CIDR
*Always subtract 2 from host total:
Network ID - First Address
Broadcast - Last Address

2.3 Linux 101 Stuff

2.3.1 Users and Priviliges

chmod numbers		
Number	Permissions	Totals
0	– – –	0+0+0
1	– – x	0+0+1
2	– w –	0+2+0
3	– w x	0+2+1
4	r – –	4+0+0
5	r – x	4+0+1
6	r w –	4+2+0
7	r w x	4+2+1

Linux101-Resources

3. Bash

3.1 Bash

Scripting with `bash`

```

```

```

```

- Other usefull ways to use it with

> **Note**
>
> You need to run the command in the same directory where pingsweep.sh file is located.

Scan the TCP port 80 for all active IPs in the ips.txt by executing the following line in the terminal:

This is how you interpret the above command: For “ip” in the “ips.txt” file, run nmap, and scan the port “-p” 80 at speed “-T4” for every IP “$ip” simultaneously “&” and finish “done”.

4. Python

4.1 Intro-To-Python

4.1.1 Intro-To-Python

This content is based on the book Think Python 2nd Edition by Allen B. Downey. More information about the book can be found here.

Lecture/discussion notes organized by chapter. Links for each of these are below.

Ch.2: Variables, expressions, and statements

Ch.3: Functions

Ch.4: Interface design

Ch.5: Conditionals and Recursion

Ch.6: Fruitful functions

Ch.7: Iteration

Ch.8: Strings

Ch.9: Word play

Ch.10: Lists

Ch. 11: Dictionaries

Ch. 12: Tuples

Math

```
#!/bin/python3

#Math
print(50 + 50) #add
print(50 - 50) #subtract
print(50 * 50) #multiply
print(50 / 50) #divide
print(50 + 50 - 50 * 50 / 50) #PEMDAS
print(50 ** 2) #exponents
print(50 % 6) #modulo
print(50 / 6)
print(50 // 6) #no leftovers
```

Variable and Methods

- Assignment statements

An assignment statement creates a new variable and gives it a value:

Python 3 has these keywords:

Keywords
False, class, finally, is, return
None, continue, for, lambda, try
True, def, from, nonlocal, while
and, del, global, not, with
as, elif, if, or, yield
assert, else, import, pass
break, except, in, raise

Adding a method

- The method is assigend to the object.

GLOSSARY

A name that refers to a value.

A statement that assigns a value to a variable.

A graphical representation of a set of variables and the values they refer to.

A reserved word that is used to parse a program; you cannot use keywords like , , and as variable names.

One of the values on which an operator operates.

A combination of variables, operators, and values that represents a single re-sult.

To simplify an expression by performing the operations in order to yield a single value.

A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements.

To run a statement and do what it says.

A way of using the Python interpreter by typing code at the prompt. A way of using the Python interpreter to read code from a script and run it.

A program stored in a file. order of operations: Rules governing the order in which expressions involving multiple operators and operands are evaluated.

To join two operands end-to-end.

Information in a program that is meant for other programmers (or anyone read- ing the source code) and has no effect on the execution of the program.

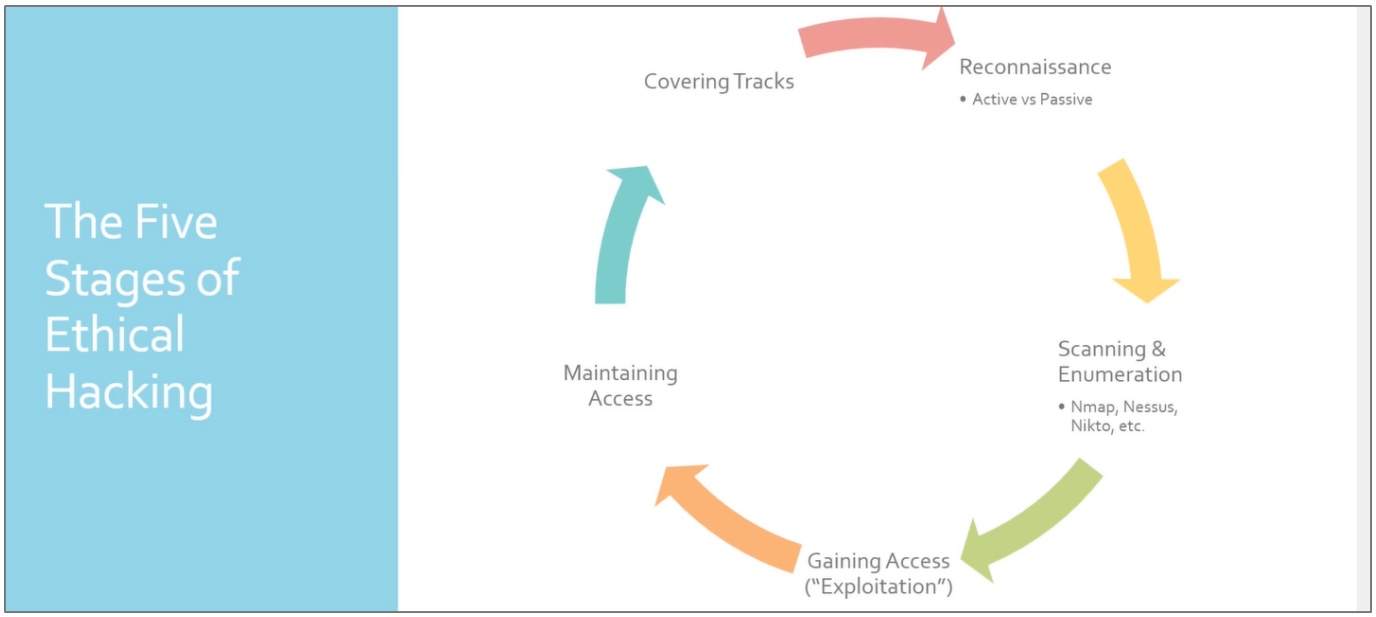
An error in a program that makes it impossible to parse (and therefore im- possible to interpret).

An error that is detected while the program is running.

The meaning of a program.

An error in a program that makes it do something other than what the programmer intended

5. 5-stages



- Reconnaisance (Activate vs Passive)
- Scanning and Enumaration (nmap, nessus)
- Gaining Access (Exploitation)
- Maitaning Access
- Covering tracs (cleaning up)

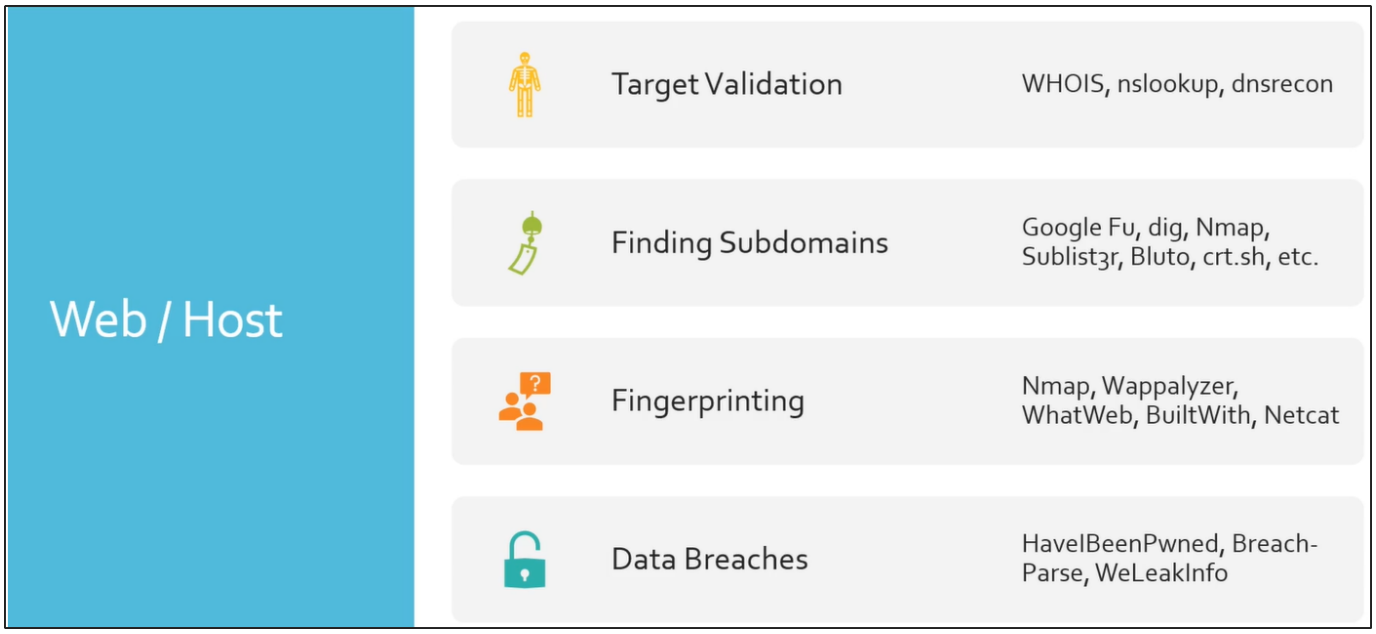
5.1 Information Gathering

5.1.1 Passive Recon

Types: Physical/Social

- Location information:
- satellite images, drone recon
- building layout
- Job Information
- Employees (names, jobtitle, phone number, etc)
- Pictures (badges photoes, desk photos, computer, etc)

5.1.2 Web/Host



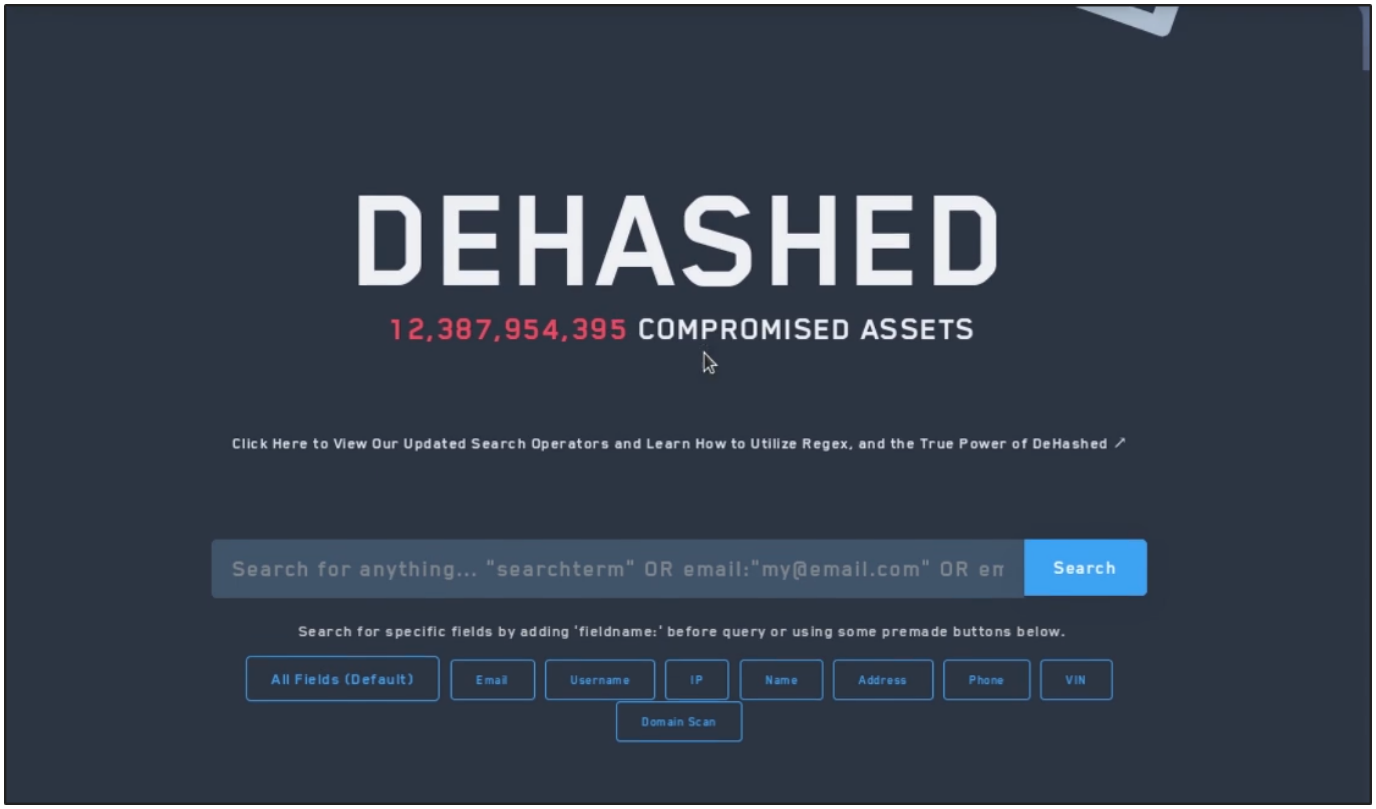
Bugcrowd for programs and targets

5.1.3 Discoverig Email Addresses

Hunter for email discovery and verify or Phonebook for chrome as an extesnion another one verifyemail

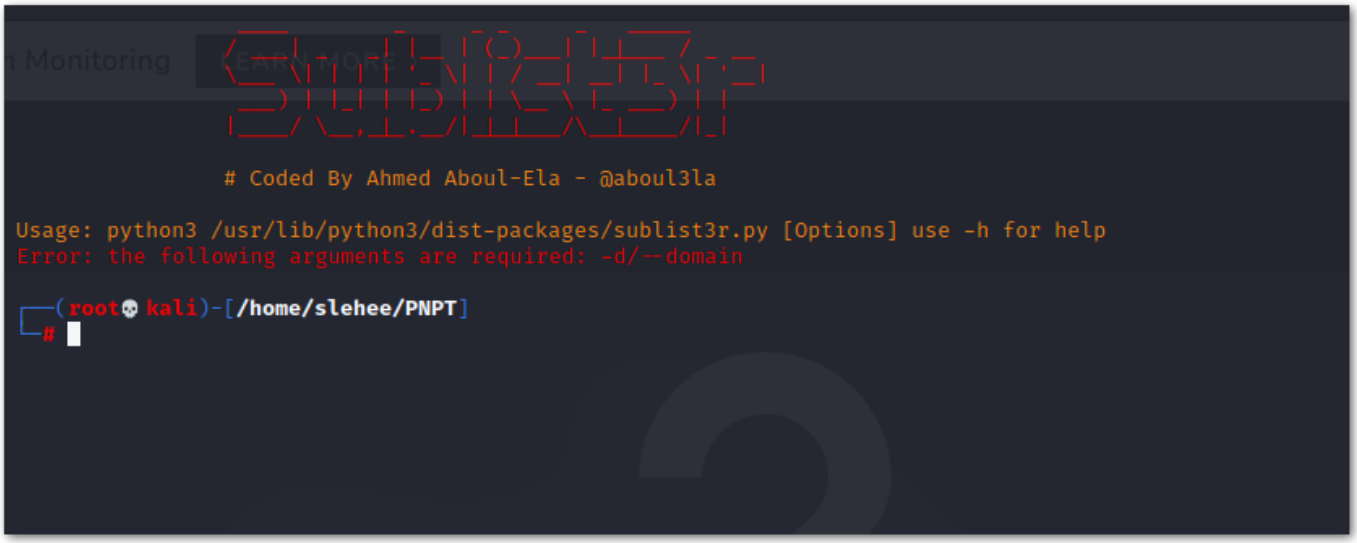
5.1.4 Hunting breached credentials

Dehashed

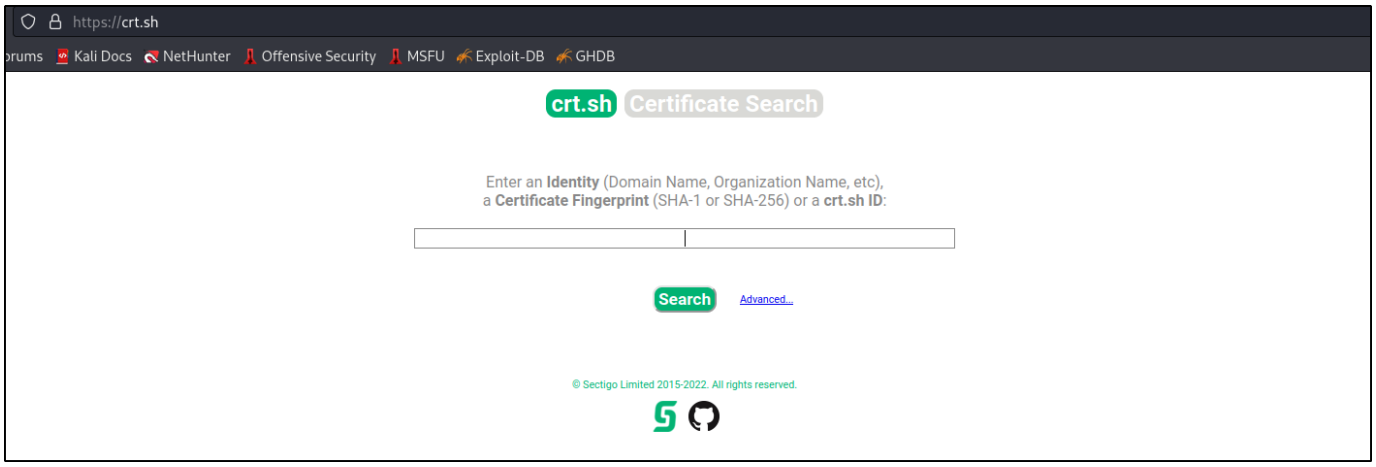


5.1.5 Hunting subdomains

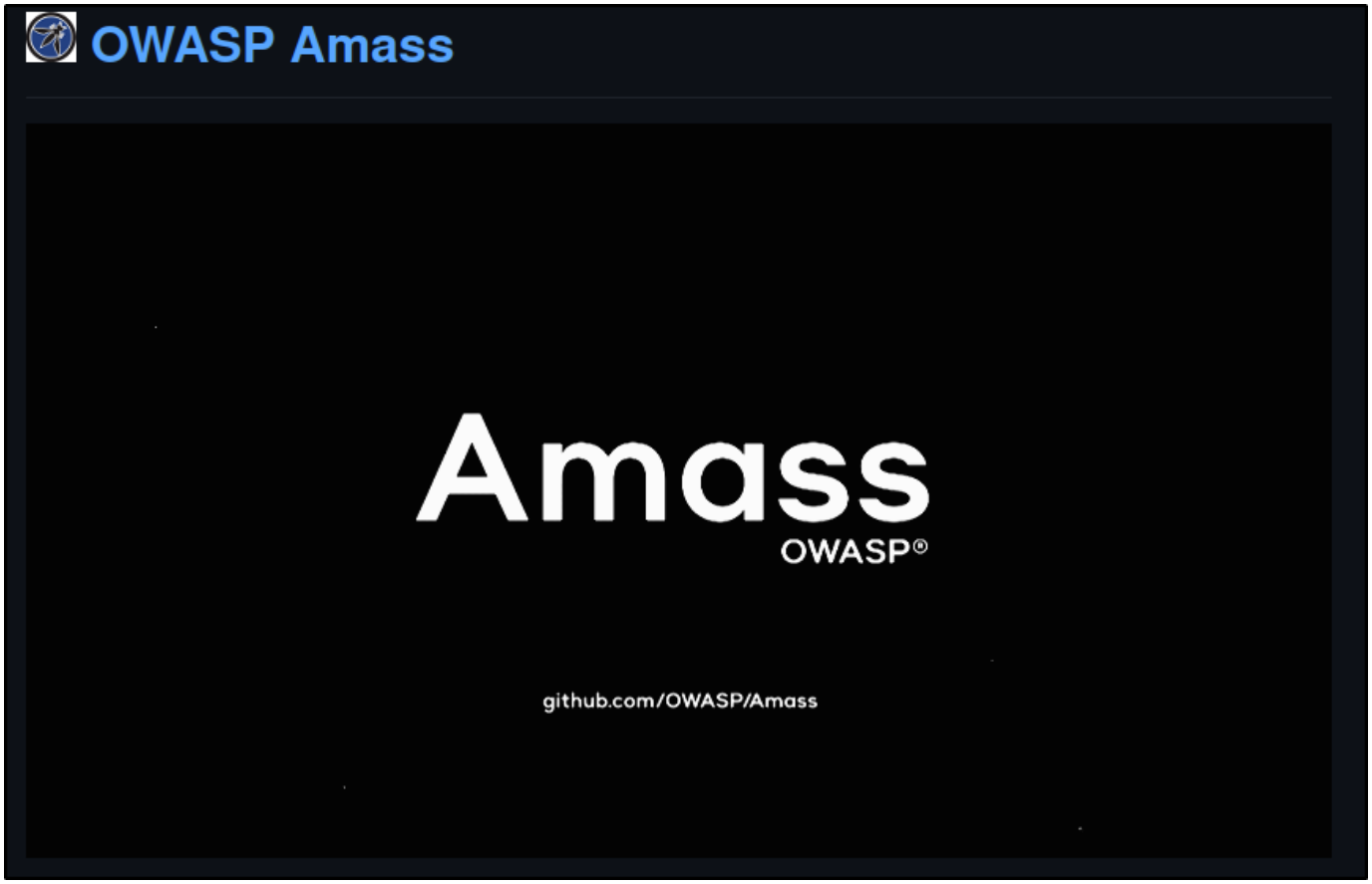
- Get subdomains with wublist3r



Search by certificate with

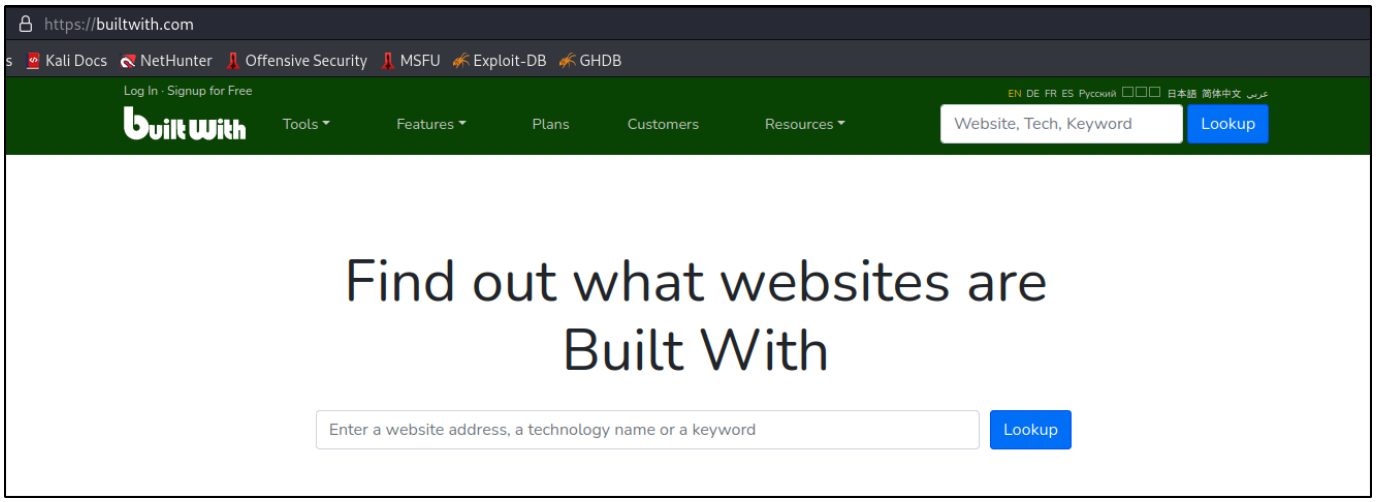


• The go to tool is OWASP-AMASS



5.1.6 Identify built with

Check builtwith and for firefox on kali



5.1.7 Installing Tor Browser on Kali Linux

6. Scaninng & Enumeration

6.1 Starting with Kioptrix

ptrix Download

Vulnhub for more VM's to test.

6.1.1 Scaning with Nmap

> **Tip**
>
> Arp scan: arp-scan -l
>
> Netdiscover: netdiscover -r 192.168.57.0/24
>
> r -range

nmap –help

nmap vuln script location

nmap –scripts script-name -v ip

nmap -sV -A –script vuln -p 80,21 192.168.57.7

nmpa -T4 (speed 1-5) -p- all ports -A evertyhing

└─# nmap -T4 -p- -A 192.168.57.4 (kioptrix machine)

> **Tip**
>
> Script wih python: nmap scan all ports then results scan with -A (all)

Nmap results on Kioptrix:

Methodology:

Start with 80;443;139

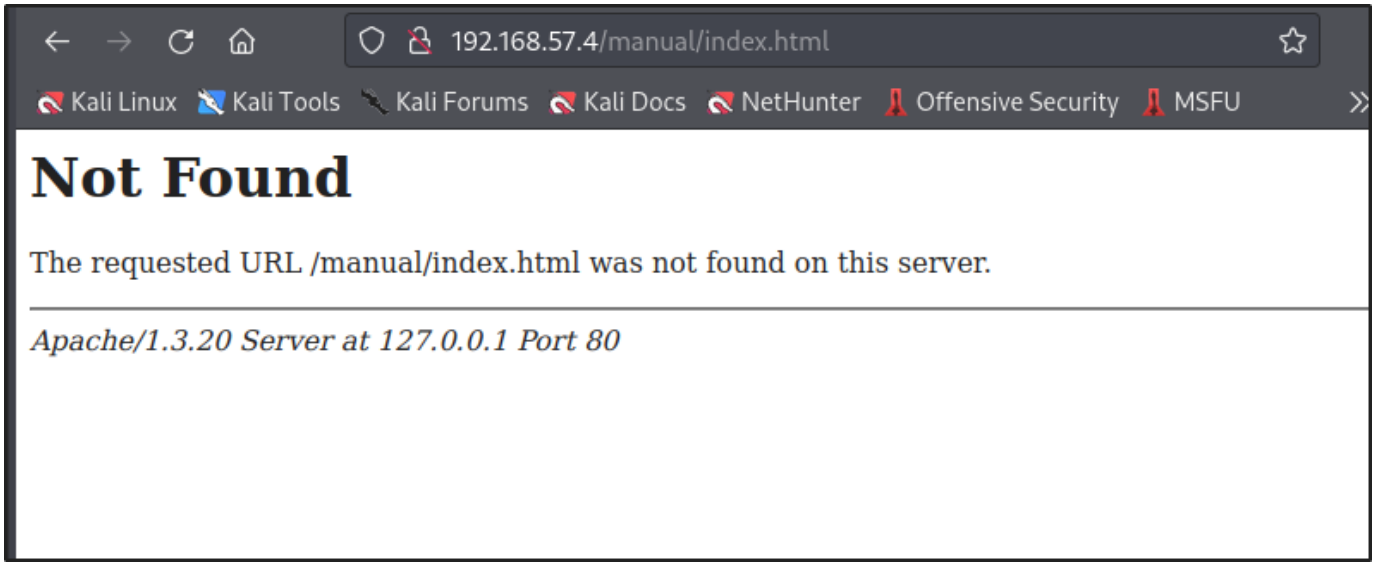
Steps:

Ports: 80 and 443 first.

Tips

Default Apache webpage. Client hygiene

Information Disclosure:



6.1.2 Use of Nikto

Good beginer tool

Tip

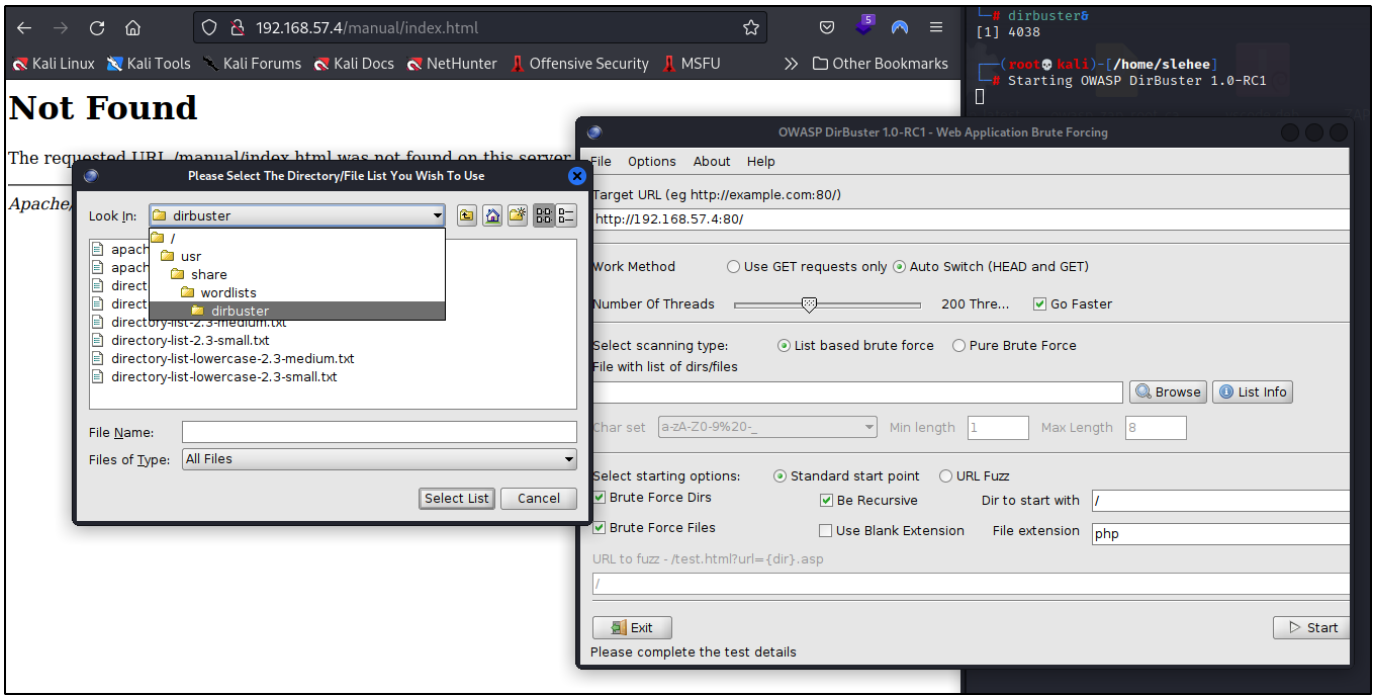
Good WAF would block it

Example to exploit:

6.1.3 Dirbuster

Alternatives

- dirbuster
- dirb
- gobuster

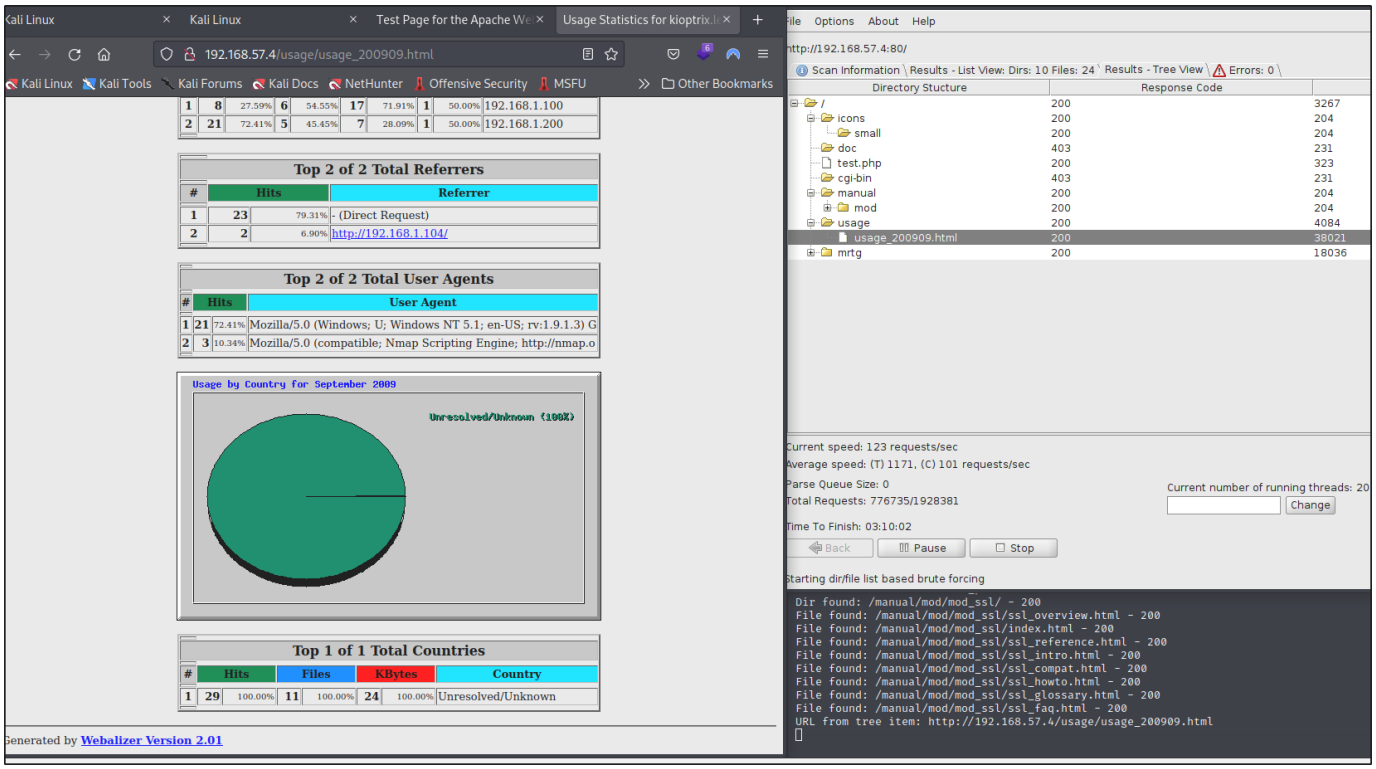


Use: php,txt,zip,pdf,docx as extension if needed

Wordlist:

Checking one othe results:

Another information disclosure:



Webaizer Version 2.01 :http://192.168.57.4/usage/usage_200909.html

6.1.4 Enumerating SMB

Using Metasploit: Auxiliary –> Scaning and Enumeration modules

search smb_version

use auxiliary/scanner/smb/smb_version

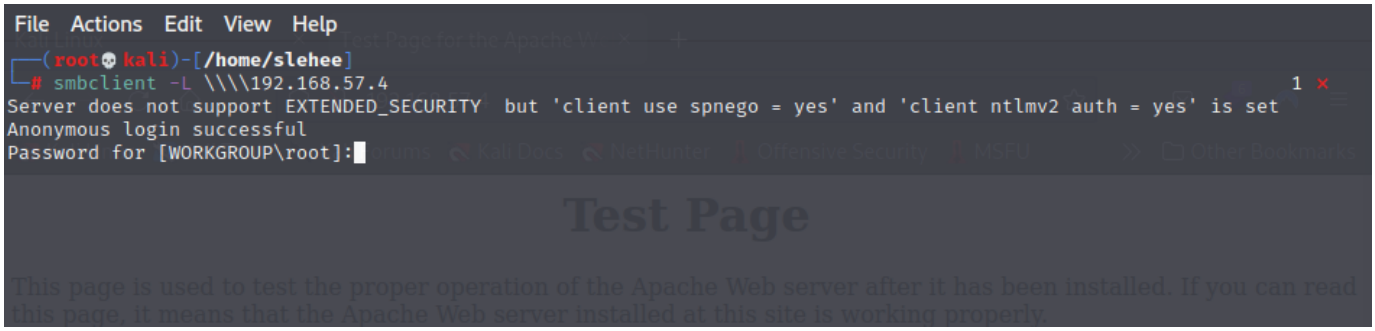
info

set

run

• Using to check anonimus access

Syntax: smbclient -L \\IP



6.1.5 Enumerating SSH

Looking fro a banner to see if any data is exposed or not.

6.1.6 Research Potential Vulnerabilities

Google

On terminal (dont be to specific) ex: Samba 2

💀

💀

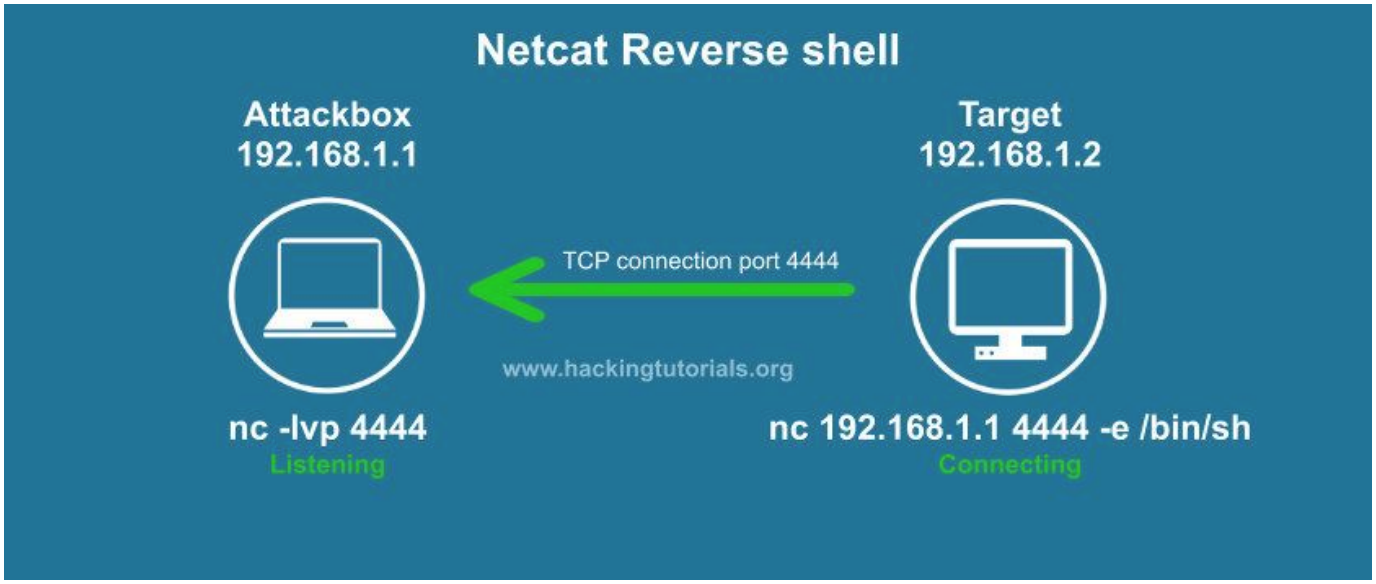
💀

💀

7. Exploitation Basics

7.0.1 Reverse Shell vs Bind Shell

A very popular usage of Netcat and probably the most common use from penetration testing perspective are reverse shells and bind shells. A reverse shell is a shell initiated from the target host back to the attack box which is in a listening state to pick up the shell. A bind shell is setup on the target host and binds to a specific port to listens for an incoming connection from the attack box. In malicious software a bind shell is often revered to as a backdoor.



GitHub examples of reverse shells.

Netcat reverse shell example

- Setup a Netcat listener.
- Connect to the Netcat listener from the target host.
- Issue commands on the target host from the attack box.

First we setup a Netcat listener on the attack box which is listening on port 4444 with the following command:

```
```

Than we issue the following command on the target host to connect to our attack box (remember we have remote code execution on this box):

For Linux:

```
```

For Windows:

```
```

On the attack box we now have a bash shell on the target host and we have full control over this box in the context of the account which initiated the reverse shell

Reverse shell without Netcat on the target host

One major downside on the shown example is that you need Netcat on that target host which is very often not the case in real world scenario's. In some cases Netcat is present, or we have a way to install it, but in many cases we need to use alternatives ways to connect back to the attack box. Let's have a look at a few alternative ways to setup a reverse shell.

Bash reverse shell

With can also use Bash to initiate a reverse shell from the target host to the attack box by using the following command:

Perl reverse shell

If Perl is present on that remote host we can also initiate a reverse shell using Perl. Run the following command on the target host to setup the reverse shell:

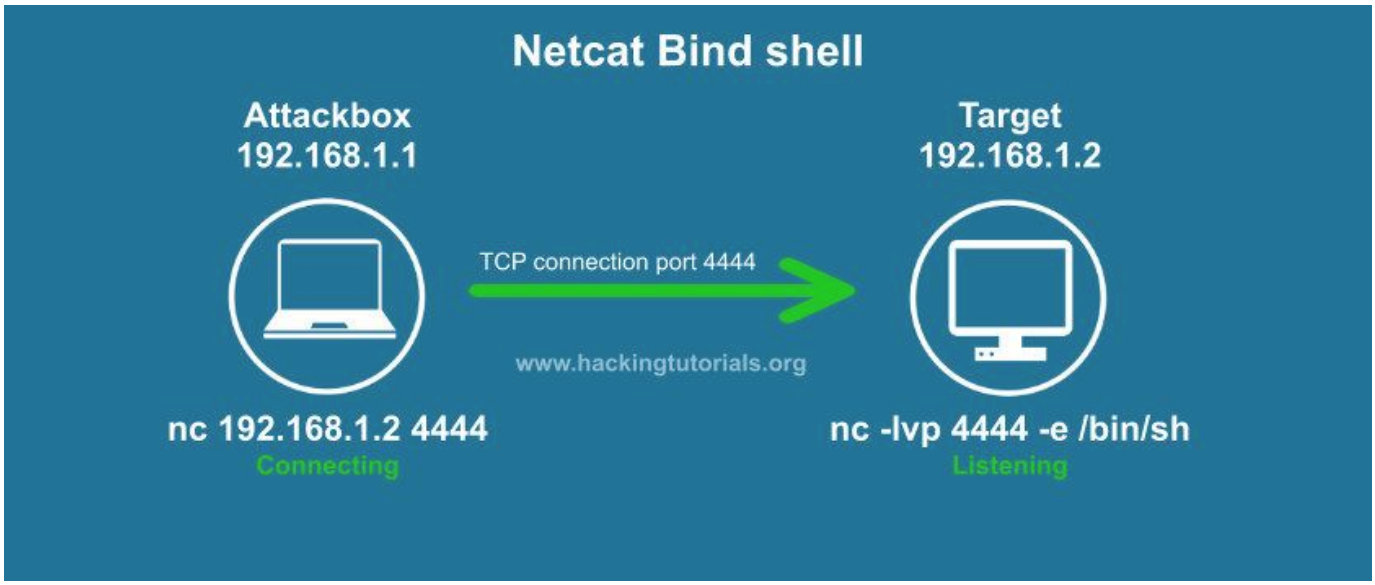
PHP reverse shell

When PHP is present on the compromised host, which is often the case on webservers, it is a great alternative to Netcat, Perl and Bash. Let's run the following code to use PHP for the reverse shell to the attack box:

Python reverse shell

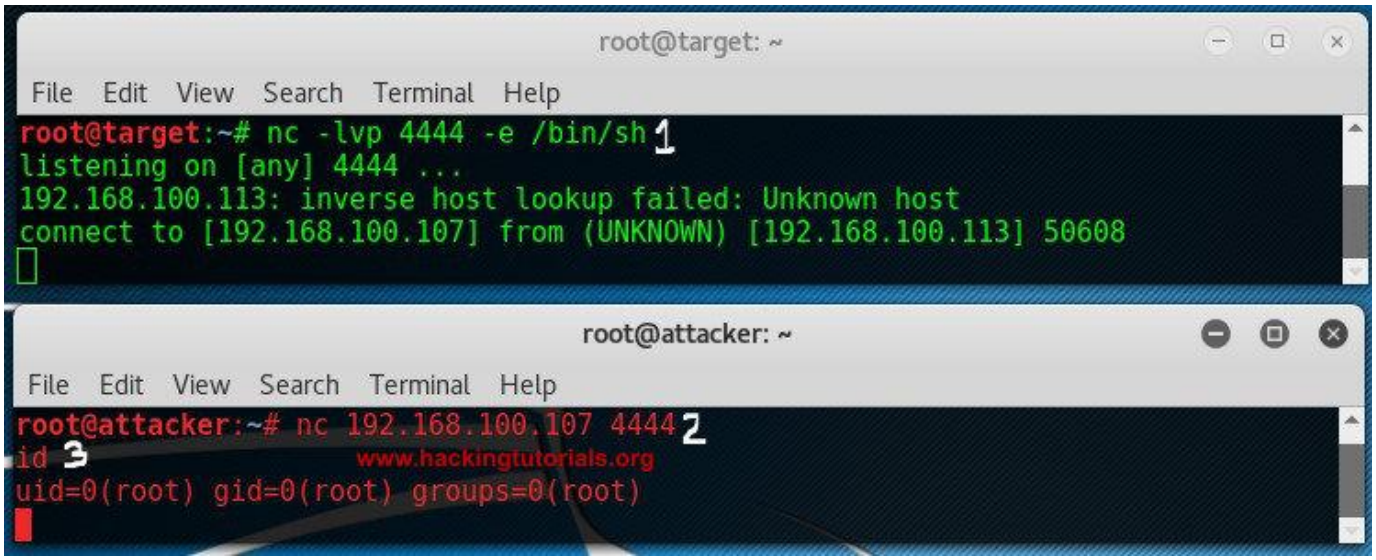
Netcat Bind Shell

As we've mentioned earlier in this Hacking with Netcat tutorial a bind shell is a shell that binds to a specific port on the target host to listen for incoming connections. Let's have a look at the visualization of a bind Netcat shell:



Netcat Bind shell example

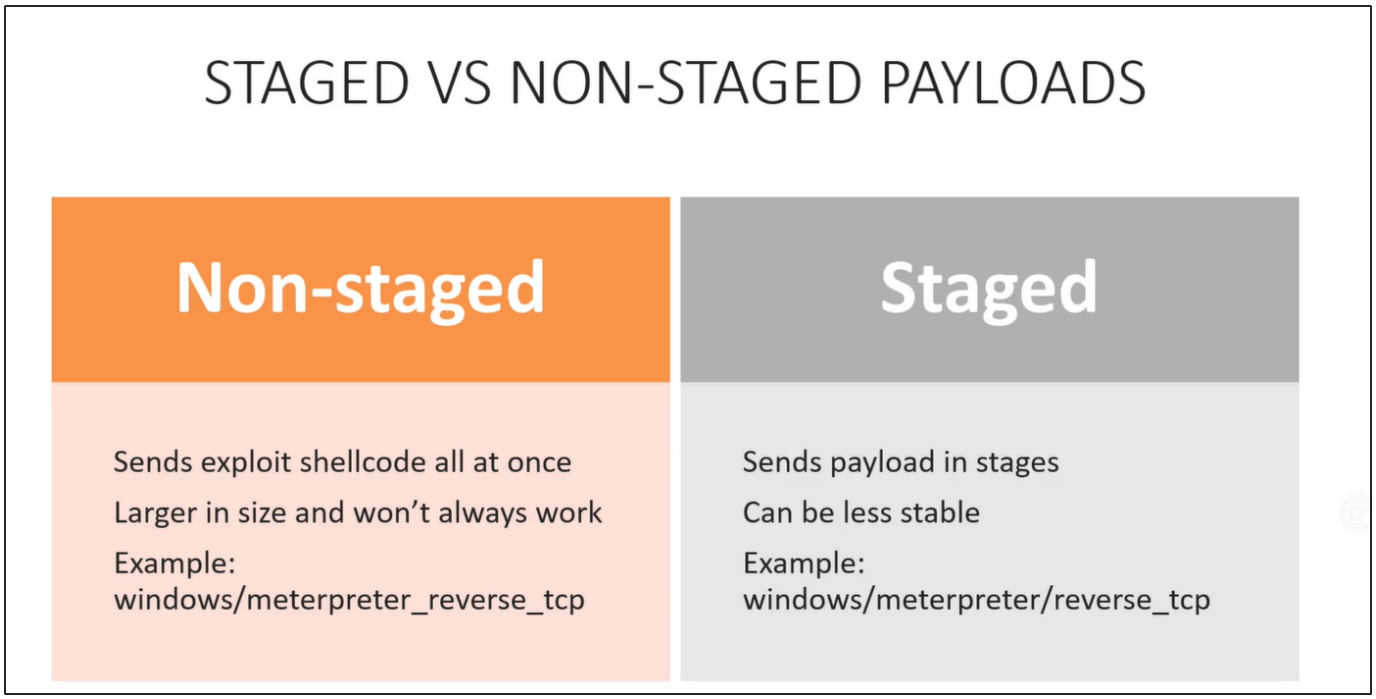
Let's see how this looks on the console:



Note

Reverse shell = Victime connects to us. Bind shell = we connect to the victime.

7.0.2 Staged vs Non-Staged Payloads



Important

Pay attention on the example: meterpreter_reverse_tcp (all in one line)=>Non-staged. Staged=>meterpreter/reverse_tcp staged

7.0.3 Gaining Root with Metasploit

-
-
-
-
- or
- linux/x86/

/

7.0.4 Manual Exploitation 80/443

7.0.5 Brute Force Attacks

syntax

- -l (user)
- -P (passwd list)

- metasploit search ssh_login

7.0.6 Credential Stuffing

Injecting breached account credentials in hopes of account takeover

Burp intruder /Sniper for password spraying and Pitchfork for user and pass.

8. Boug Bounty

8.1 Bounty101

8.1.1 What is Bug Bounty Hunting?

Lately Bug Bounty Hunting has become quite a buzzword. But what exactly is it and how can somebody start?

Bug bounty hunting 101

While bug bounty hunting can be proven highly lucrative, and it certainly has been for some people, there are also different reasons that people choose this professional path. First of all, being the boss of your own self gives you a lot of freedom. You do not have to be hired and your skills are the only thing that matters,so nobody is going to judge you based on your looks, personality etc. There are people that started their cybersecurity journey late and do not have a computer science degree. Working as a freelance bounty hunter allows a massive amount of flexibility for people that can not work on a 9-5. Also, these platforms allow people from less wealthy countries to have much higher earnings in comparison to having a regular job.

Just reading through these bug reports can be a fun learning experience for most hacking enthusiasts. Some of them are really complex and can give you a headache just by reading them but not all of them. In 2016 a researcher disclosed a bug to facebook that could allow him to reset the password and take control of any account. When you would request to change your password, Facebook would send a 6-digit PIN in either your phone or mail that you had to submit. You had a limited number of tries to get this password right before you get locked out. What the researcher found out, was that the lockout mechanism was not implemented on beta.facebook.com and mbasic.beta.facebook.com. This is a very clever hack but it does not sound that complicated, probably a lot of readers could replicate this if this was still applicable. So next time somebody asks you “Can you hack a Facebook account for me?” after learning that you are a hacker, you can reply “If I ever find a way I will probably report it for thousands of dollars, sorry”.

THE BUG HUNTER’S METHODOLOGY (TBHM) ON GITHUB

You too can become a bug bounty hunter!

8.2 Web Applications

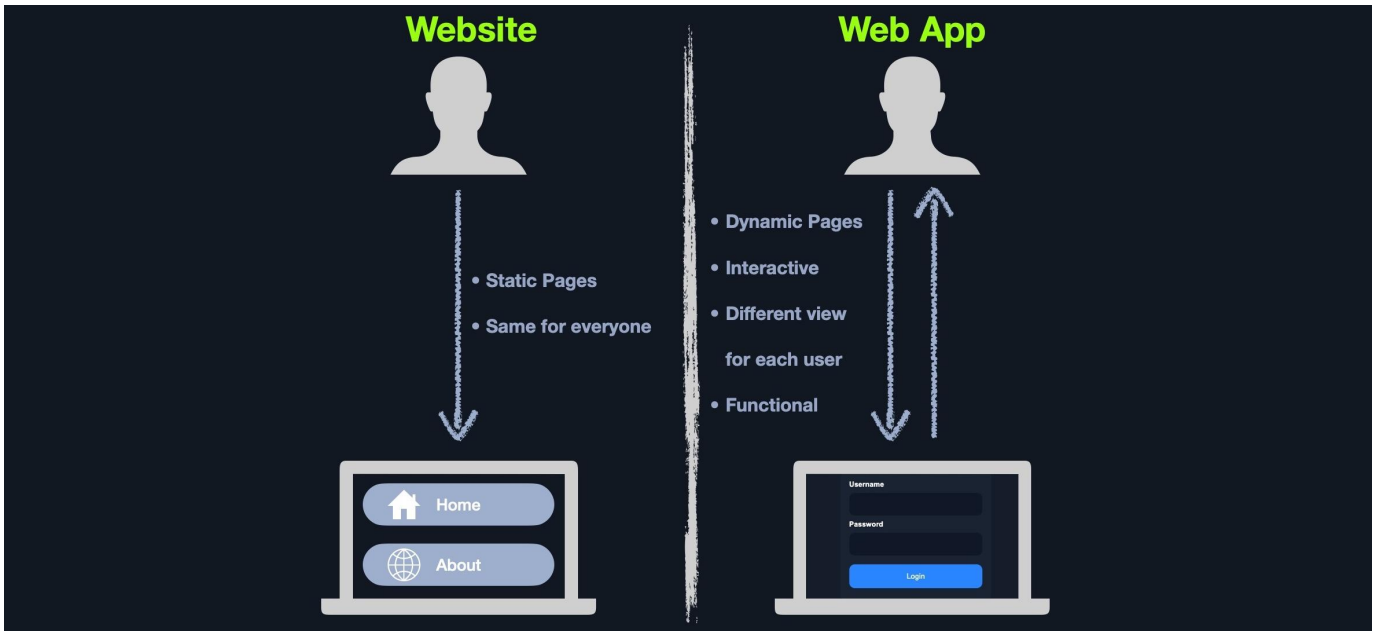
Web applications are interactive applications that run on web browsers. Web applications usually adopt a client-server architecture to run and handle interactions. They typically have front end components (i.e., the website interface, or “what the user sees”) that run on the client-side (browser) and other back end components (web application source code) that run on the server-side (back end server/databases).

This allows organizations to host powerful applications with near-complete real-time control over their design and functionality while being accessible worldwide. Some examples of typical web applications include online email services like Gmail, online retailers Amazon, and online word processors like Google Docs.

Web applications are not exclusive to giant providers like Google or Microsoft but can be developed by any web developer and hosted online in any of the common hosting services, to be used by anyone on the internet. This is why today we have millions of web applications all over the internet, with billions of users interacting with them every day

Web Applications vs. Websites

In the past, we interacted with websites that are static and cannot be changed in real-time. This means that traditional websites were statically created to represent specific information, and this information would not change with our interaction. To change the website’s content, the corresponding page has to be edited by the developers manually. These types of static pages do not contain functions and, therefore, do not produce real-time changes. That type of website is also known as Web 1.0.



In the past, we interacted with websites that are static and cannot be changed in real-time. This means that traditional websites were statically created to represent specific information, and this information would not change with our interaction. To change the website's content, the corresponding page has to be edited by the developers manually. These types of static pages do not contain functions and, therefore, do not produce real-time changes. That type of website is also known as Web 1.0.

- Being modular
- Running on any display size
- Running on any platform without being optimized

Web Application Distribution

There are many open-source web applications used by organizations worldwide that can be customized to meet each organization's needs. Some common open source web applications includ

- WordPress
- OpenCart
- Joomla

There are also proprietary 'closed source' web applications, which are usually developed by a certain organization and then sold to another organization or used by organizations through a subscription plan model. Some common closed source web applications include:

- Wix
- Shopify
- DotNetNuke

Security Risks of Web Applications

Web application attacks are prevalent and present a challenge for most organizations with a web presence, regardless of their size. After all, they are usually accessible from any country by everyone with an internet connection and a web browser and usually offer a vast attack surface. As web applications become more complicated and advanced, so does the possibility of critical vulnerabilities being incorporated into their design.

Since web applications are run on servers that may host other sensitive information and are often also linked to databases containing sensitive user or corporate data, all of this data could be compromised if a web site is successfully attacked. This is why it is critical for any business that utilizes web applications to properly test these applications for vulnerabilities and patch them promptly while testing that the patch fixes the flaw and does not inadvertently introduce any new flaws.

Web application penetration testing is an increasingly critical skill to learn. Any organization looking to secure their internet-facing (and internal) web applications should undergo frequent web application tests and implement secure coding practices at every development life cycle stage.

To properly pentest web applications, we need to understand how they work, how they are developed, and what kind of risk lies at each layer and component of the application depending on the technologies in use.

ne of the most current and widely used methods for testing web applications is the OWASP Web Security Testing Guide.

One of the most common procedures is to start by reviewing a web application's front end components, such as , , and (also known as the front end trinity), and attempt to find vulnerabilities such as Sensitive Data Exposure and Cross-Site Scripting (XSS). Once all front end components are thoroughly tested, we would typically review the web application's core functionality and the interaction between the browser and the webserver to enumerate the technologies the webserver uses and look for exploitable flaws. We typically assess web applications from both an unauthenticated and authenticated perspective (if the application has login functionality) to maximize coverage and review every possible attack scenario.

Web Application Layout

No two web applications are identical. Businesses create web applications for a multitude of uses and audiences. Web applications are designed and programmed differently, and back end infrastructure can be set up in many different ways. It is important to understand the various ways web applications can run behind the scenes, the structure of a web application, its components, and how they can be set up within a company's infrastructure.

Web application layouts consist of many different layers that can be summarized with the following three main categories:

Category	Description
Web Application Infrastructure	Describes the structure of required components, such as the database, needed for the web application to function as intended. Since the web application can be set up to run on a separate server, it is essential to know which database server it needs to access.
Web Application Components	The components that make up a web application represent all the components that the web application interacts with. These are divided into the following three areas: UI/UX, Client, and Server components.
Web Application Architecture	Architecture comprises all the relationships between the various web application component

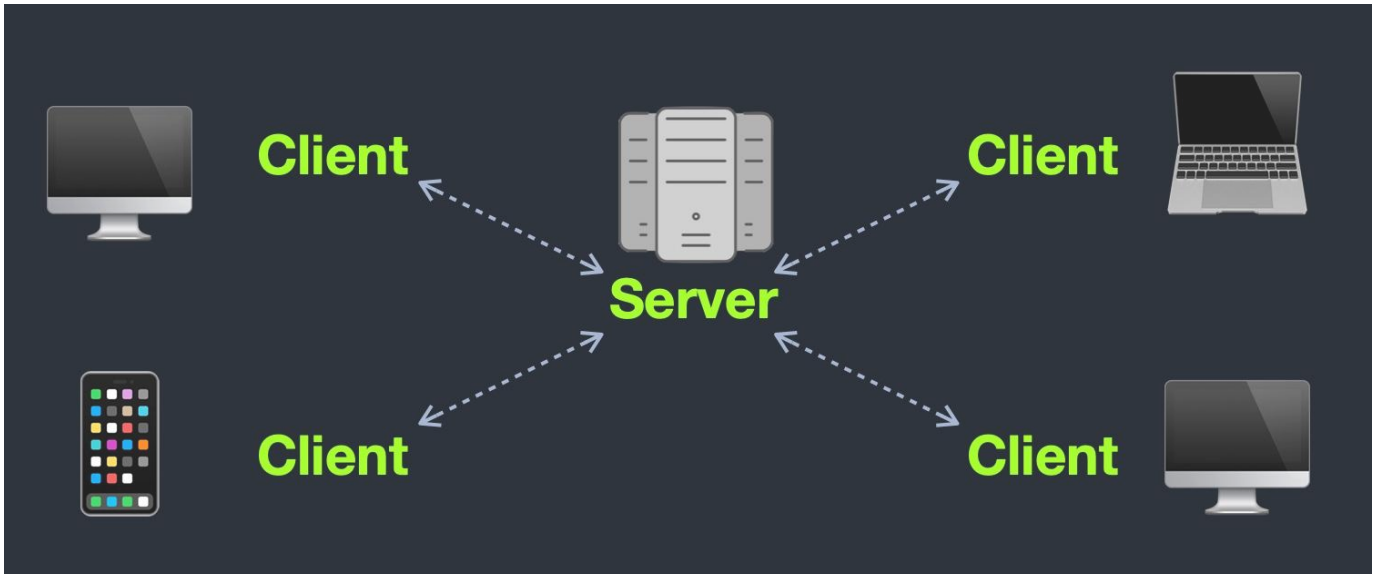
Web Application Infrastructure

Web applications can use many different infrastructure setups. These are also called models. The most common ones can be grouped into the following four types:

- Client-Server
- One Server
- Many Servers - One Database
- Many Servers - Many Databases

CLIENT-SERVER

Web applications often adopt the client-server model. A server hosts the web application in a client-server model and distributes it to any clients trying to access it.

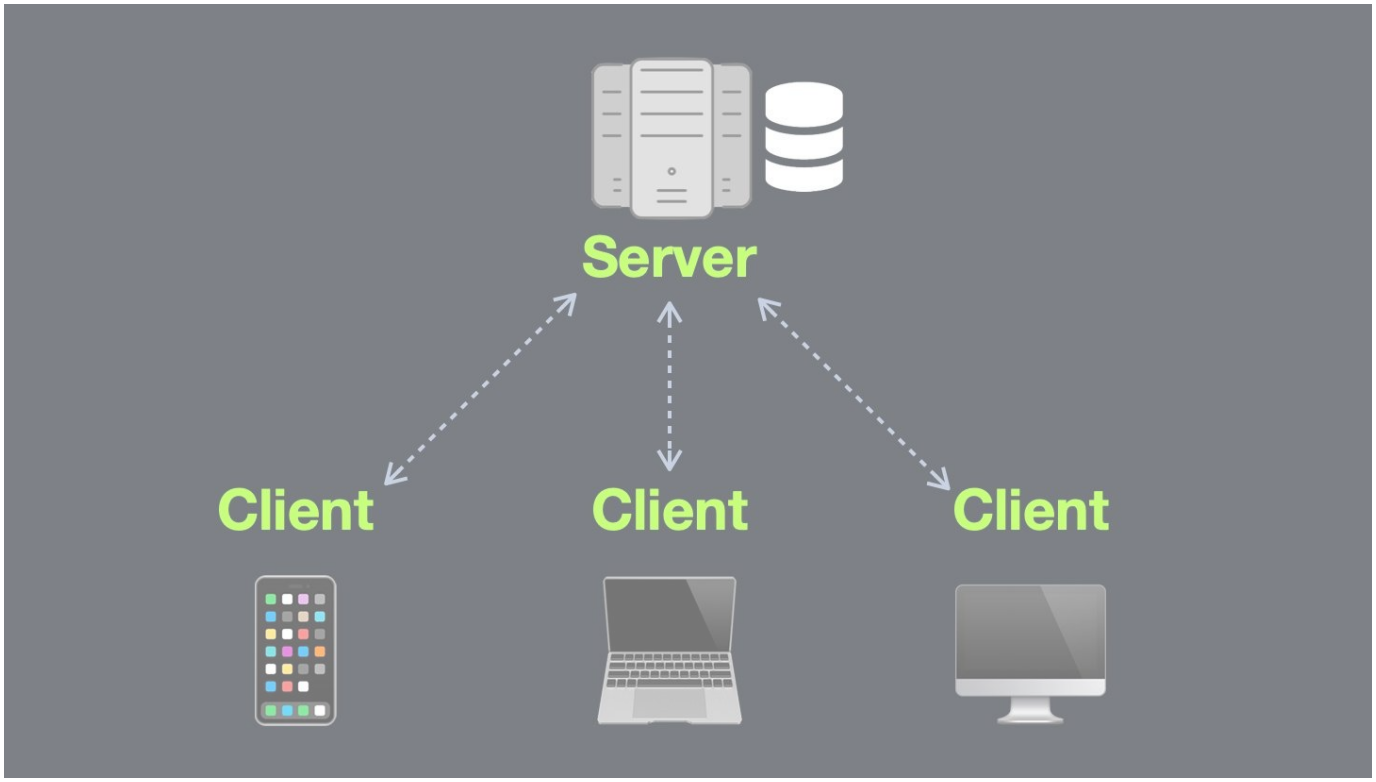


In this model, web applications have two types of components, those in the front end, which are usually interpreted and executed on the client-side (browser), and components in the back end, usually compiled, interpreted, and executed by the hosting server.

When a client visits the web application's URL (web address, i.e., https://www.acme.local), the server uses the main web application interface (UI). Once the user clicks on a button or requests a specific function, the browser sends an HTTP web request to the server, which interprets this request and performs the necessary task(s) to complete the request (i.e., logging the user in, adding an item to the shopping cart, browsing to another page, etc.). Once the server has the required data, it sends the result back to the client's browser, displaying the result in a human-readable way.

However, even though most web applications utilize a client-server front-back end architecture, there are many design implementations.

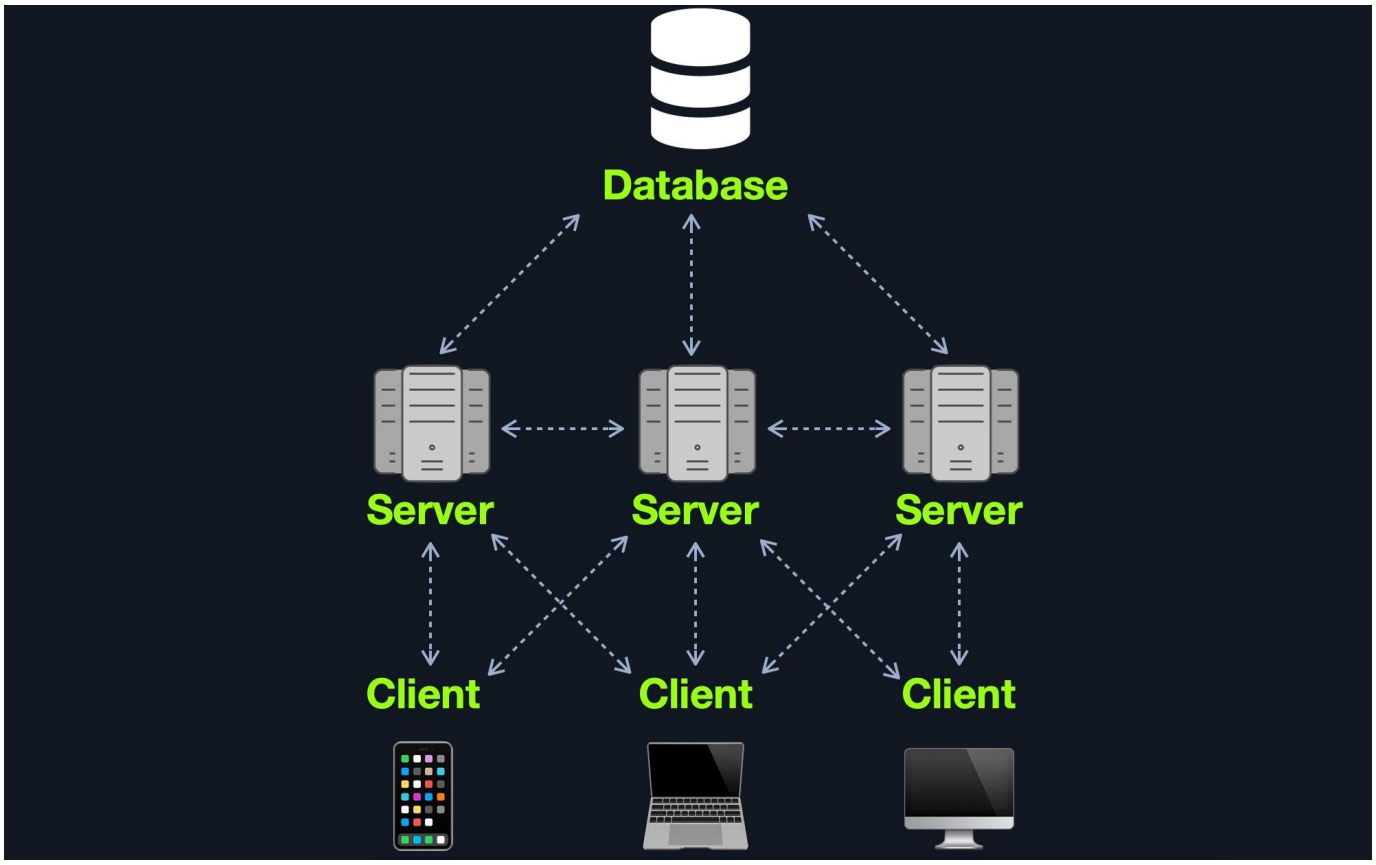
ONE SERVER



If any web application hosted on this server is compromised in this architecture, then all web applications' data will be compromised. This design represents an "all eggs in one basket" approach since if any of the hosted web applications are vulnerable, the entire webserver becomes vulnerable.

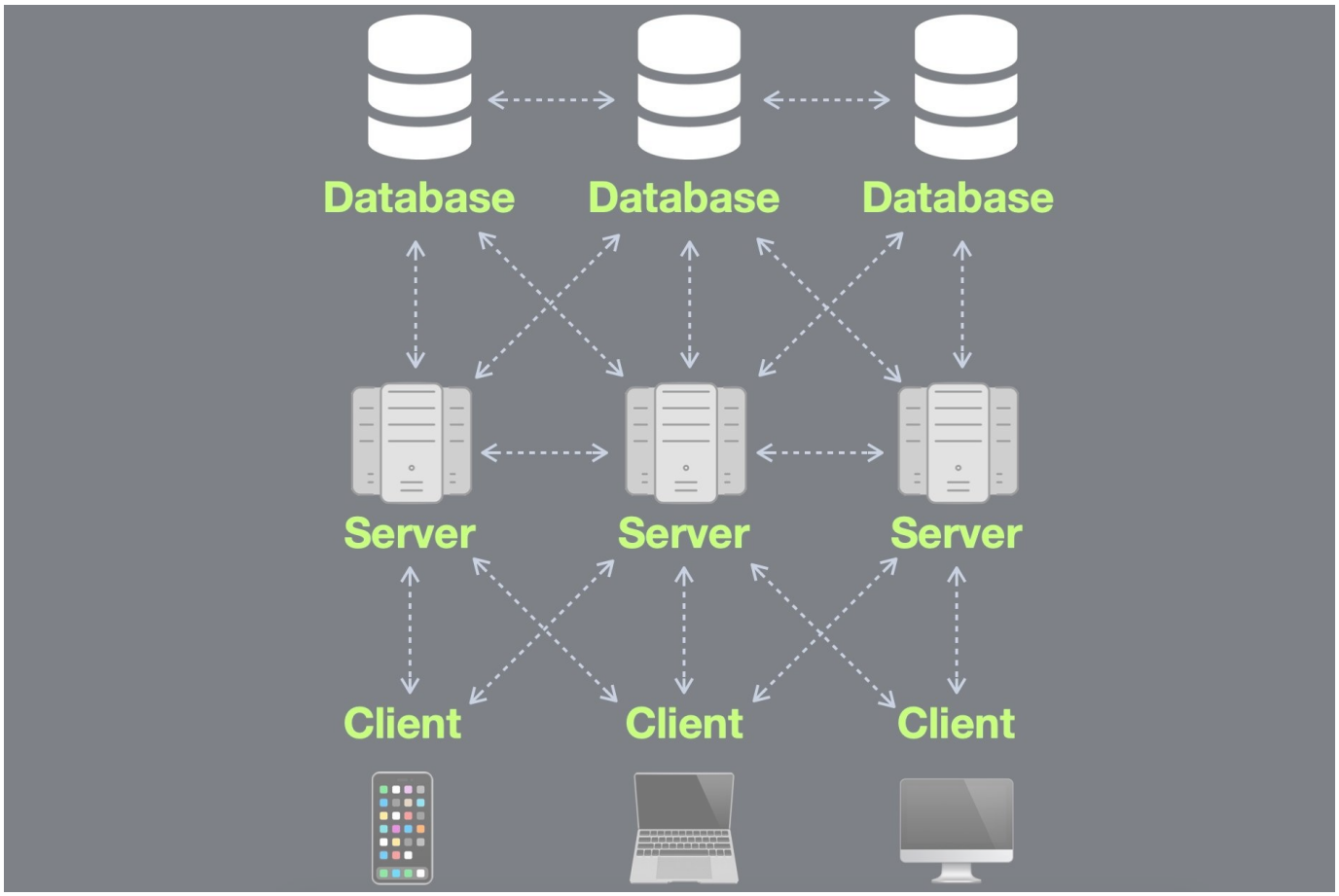
MANY SERVERS - ONE DATABASE

This model separates the database onto its own database server and allows the web applications' hosting server to access the database server to store and retrieve data. It can be seen as many-servers to one-database and one-server to one-database, as long as the database is separated on its own database server.



This model’s main advantage (from a security point of view) is segmentation, where each of the main components of a web application is located and hosted separately. In case one webserver is compromised, other webservers are not directly affected. Similarly, if the database is compromised (i.e., through a SQL injection vulnerability), the web application itself is not directly affected. There are still access control measures that need to be implemented after asset segmentation, such as limiting web application access to only data needed to function as intendedThis model’s main advantage (from a security point of view) is segmentation, where each of the main components of a web application is located and hosted separately. In case one webserver is compromised, other webservers are not directly affected. Similarly, if the database is compromised (i.e., through a SQL injection vulnerability), the web application itself is not directly affected. There are still access control measures that need to be implemented after asset segmentation, such as limiting web application access to only data needed to function as intended

MANY SERVERS - MANY DATABASES



This design is also widely used for redundancy purposes, so if any web server or database goes offline, a backup will run in its place to reduce downtime as much as possible

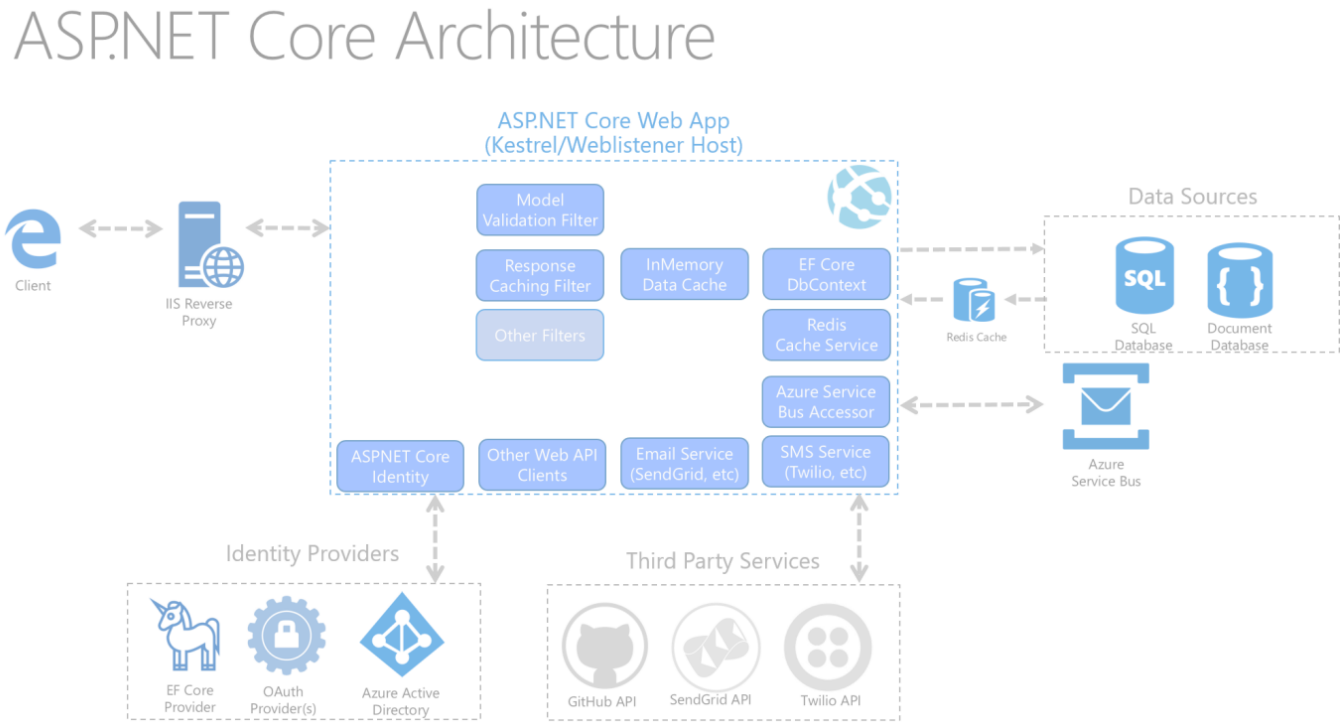
WEB APPLICATION COMPONENT

Each web application can have a different number of components. Nevertheless, all of the components of the models mentioned previously can be broken down to:

1. Client
2. Server
   - Webserver
   - Web Application Logic
   - Database
3. Services (Microservices)
   - 3rd Party Integrations
   - Web Application Integrations
4. Functions (Serverless)

WEB APPLICATION ARCHITECTURE

Layer	Description
Presentation Layer	Consists of UI process components that enable communication with the application and the system. These can be accessed by the client via the web browser and are returned in the form of HTML, JavaScript, and CSS.
Application Layer	This layer ensures that all client requests (web requests) are correctly processed. Various criteria are checked, such as authorization, privileges, and data passed on to the client.
Data Layer	The data layer works closely with the application layer to determine exactly where the required data is stored and can be accessed



MICROSERVICES

We can think of microservices as independent components of the web application, which in most cases are programmed for one task only. For example, for an online store, we can decompose core tasks into the following components:

- Registration
- Search
- Payments
- Ratings
- Reviews

These components communicate with the client and with each other. The communication between these microservices is , which means that the request and response are independent. This is because the stored data is from the respective microservices.

This AWS whitepaper provides an excellent overview of microservice implementation.

SERVERLESS

Cloud providers such as AWS, GCP, Azure, among others, offer serverless architectures. These platforms provide application frameworks to build such web applications without having to worry about the servers themselves. These web applications then run in stateless computing containers (Docker, for example). This type of architecture gives a company the flexibility to build and deploy applications and services without having to manage infrastructure; all server management is done by the cloud provider, which gets rid of the need to provision, scale, and maintain servers needed to run applications and databases.

You can read more about serverless computing and its various use cases here.

Front End vs. Back End

We may have heard the terms front end and back end web development, or the term Full Stack web development, which refers to both front and back end web development. These terms are becoming synonymous with web application development, as they comprise the majority of the web development cycle. However, these terms are very different from each other, as each refers to one side of the web application, and each function and communicate in different are

FRONT END

The front end of a web application contains the user's components directly through their web browser (client-side). These components make up the source code of the web page we view when visiting a web application and usually include , , and , which is then interpreted in real-time by our browser

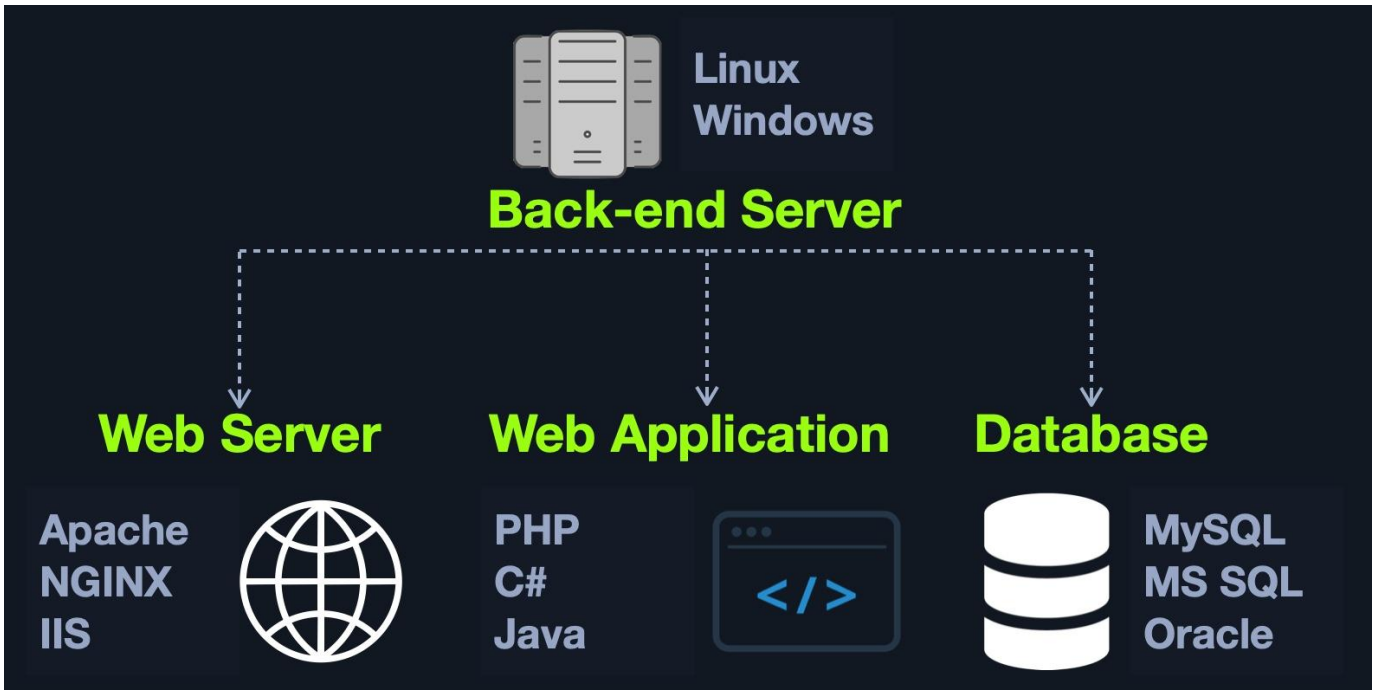
This includes everything that the user sees and interacts with, like the page's main elements such as the title and text , the design and animation of all elements , and what function each part of a page performs .

There are many sites available to us to practice front end coding. One example is this one.

BACK END

The back end of a web application drives all of the core web application functionalities, all of which is executed at the back end server, which processes everything required for the web application to run correctly. It is the part we may never see or directly interact with, but a website is just a collection of static web pages without a back end.

There are four main back end components for web applications: |**Components**|**Description**| |--------------|---------------| |Back end Servers |The hardware and operating system that hosts all other components and are usually run on operating systems like Linux, Windows, or using Containers. |Web Servers |Web servers handle HTTP requests and connections. Some examples are Apache, NGINX, and IIS. |Databases |Databases (DBs) store and retrieve the web application data. Some examples of relational databases are MySQL, MSSQL, Oracle, PostgreSQL, while examples of non-relational databases include NoSQL and MongoDB. | Development Frameworks | Development Frameworks are used to develop the core Web Application. Some well-known frameworks include PHP, C#, Java, Python, and NodeJS JavaScript.



SECURING FRONT/BACK END

Even though in most cases, we will not have access to the back end code to analyze the individual functions and the structure of the code, it does not make the application invulnerable. It could still be exploited by various injection attacks, for example.

Suppose we have a search function in a web application that mistakenly does not process our search queries correctly. In that case, we could use specific techniques to manipulate the queries in such a way that we gain unauthorized access to specific database data SQL injections or even execute operating system commands via the web application, also known as Command Injections.

The top 20 most common mistakes web developers make that are essential for us as penetration testers are:

1. Permitting Invalid Data to Enter the Database
2. Focusing on the System as a Whole
3. Establishing Personally Developed Security Methods
4. Treating Security to be Your Last Step
5. Developing Plain Text Password Storage
6. Creating Weak Passwords
7. Storing Unencrypted Data in the Database
8. Depending Excessively on the Client Side
9. Being Too Optimistic
10. Permitting Variables via the URL Path Name
11. Trusting third-party code
12. Hard-coding backdoor accounts
13. Unverified SQL injections
14. Remote file inclusions
15. Insecure data handling
16. Failing to encrypt data properly
17. Not using a secure cryptographic system
18. Ignoring layer 8
19. Review user actions
20. Web Application Firewall misconfigurations

These mistakes lead to the OWASP Top 10 vulnerabilities for web applications, which we will discuss in other modules:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitorin

Sensitive Data Exposure

All of the front end components we covered are interacted with on the client-side. Therefore, if they are attacked, they do not pose a direct threat to the core back end of the web application and usually will not lead to permanent damage. However, as these components are executed on the client-side, they put the end-user in danger of being attacked and exploited if they do have any vulnerabilities. If a front end vulnerability is leveraged to attack admin users, it could result in unauthorized access, access to sensitive data, service disruption, and more.

Although the majority of web application penetration testing is focused on back end components and their functionality, it is important also to test front end components for potential vulnerabilities, as these types of vulnerabilities are can sometimes be utilized to gain access to sensitive functionality (i.e., an admin panel), which may lead to compromising the entire server.

refers to the availability of sensitive data in clear-text to the end-user. This is usually found in the source code of the web page or page source on the front end of web applications.

HTML Injection

Another major aspect of front end security is validating and sanitizing accepted user input. In many cases, user input validation and sanitization is carried out on the back end. However, some user input would never make it to the back end in some cases and is completely processed and rendered on the front end. Therefore, it is critical to validate and sanitize user input on both the front end and the back end.

When a user has complete control of how their input will be displayed, they can submit HTML code, and the browser may display it as part of the page. This may include a malicious HTML code, like an external login form, which can be used to trick users into logging in while actually sending their login credentials to a malicious server to be collected for other attacks.

Another example of is web page defacing. This consists of injecting new code to change the web page's appearance, inserting malicious ads, or even completely changing the page. This type of attack can result in severe reputational damage to the company hosting the web application.

Cross-Site Scripting (XSS)

vulnerabilities can often be utilized to also perform attacks by injecting code to be executed on the client-side. Once we can execute code on the victim's machine, we can potentially gain access to the victim's account or even their machine. is very similar to in practice. However, involves the injection of code to perform more advanced attacks on the client-side, instead of merely injecting code. There are three main types of : |**Type**|**Description**| |--------|---------------| |Reflected XSS |Occurs when user input is displayed on the page after processing (e.g., search result or error message). |Stored XSS |Occurs when user input is stored in the back end database and then displayed upon retrieval (e.g., posts or comments). |DOM XSS |Occurs when user input is directly shown in the browser and is written to an HTML DOM object (e.g., vulnerable username or page title).

In the example we saw for HTML Injection, there was no input sanitization whatsoever. Therefore, it may be possible for the same page to be vulnerable to XSS attacks. We can try to inject the following DOM XSS JavaScript code as a payload, which should show us the cookie value for the current user:

```

```

This payload is accessing the HTML document tree and retrieving the cookie object's value. When the browser processes our input, it will be considered a new DOM, and our JavaScript will be executed, displaying the cookie value back to us in a pop

Cross-Site Request Forgery (CSRF)

The third type of front end vulnerability that is caused by unfiltered user input is Cross-Site Request Forgery (CSRF). CSRF attacks may utilize XSS vulnerabilities to perform certain queries, and API calls on a web application that the victim is currently authenticated to.

A common CSRF attack to gain higher privileged access to a web application is to craft a JavaScript payload that automatically changes the victim's password to the value set by the attacker. Once the victim views the payload on the vulnerable page (e.g., a malicious comment containing the JavaScript CSRF payload), the JavaScript code would execute automatically. It would use the victim's logged-in session to change their password. Once that is done, the attacker can log in to the victim's account and control it.

CSRF can also be leveraged to attack admins and gain access to their accounts. Admins usually have access to sensitive functions, which can sometimes be used to attack and gain control over the back-end server (depending on the functionality provided to admins within a given web application). Following this example, instead of using JavaScript code that would return the session cookie, we would load a remote .js (JavaScript) file, as follows:

```

```

The file would contain the malicious code that changes the user's password. Developing the in this case requires knowledge of this web application's password changing procedure and . The attacker would need to create JavaScript code that would replicate the desired functionality and automatically carry it out (i.e., JavaScript code that changes our password for this specific web application).

PREVENTION

Though there should be measures on the back end to detect and filter user input, it is also always important to filter and sanitize user input on the front end before it reaches the back end, and especially if this code may be displayed directly on the client-side without communicating with the back end. Two main controls must be applied when accepting user input:

Type	Description
Sanitization	Removing special characters and non-standard characters from user input before displaying it or storing it.
Validation	Ensuring that submitted user input matches the expected format (i.e., submitted email matched email format)

Furthermore, it is also important to sanitize displayed output and clear any special/non-standard characters. In case an attacker manages to bypass front end and back end sanitization and validation filters, it will still not cause any harm on the front end.

Once we sanitize and/or validate user input and displayed output, we should be able to prevent attacks like , , or . Another solution would be to implement a , which should help to prevent injection attempts automatically. However, it should be noted that WAF solutions can potentially be bypassed, so developers should follow coding best practices and not merely rely on an appliance to detect/block attacks.

This Cross-Site Request Forgery Prevention Cheat Sheet from OWASP discusses the attack and prevention measures in greater detail.

Back End Servers

A back end server is the hardware and operating system on the back end that hosts all of the applications necessary to run the web application. It is the real system running all of the processes and carrying all of the tasks that make up the entire web application. The back end server would fit in the Data access layer.

SOFTWARE

The back end server contains the other 3 back end components:

- Web Server
- Database
- Development Framework

There are many popular combinations of "stacks" for back-end servers, which contain a specific set of back end components. Some common examples include:

Combinations	Components
LAMP	Linux, Apache, MySQL, and PHP.
WAMP	Windows, Apache, MySQL, and PHP.
WINS	Windows, IIS, .NET, and SQL Server
MAMP	macOS, Apache, MySQL, and PHP.
XAMPP	Cross-Platform, Apache, MySQL, and PHP/PERL.

WEB SERVERS

A web server is an application that runs on the back end server, which handles all of the HTTP traffic from the client-side browser, routes it to the requested pages, and finally responds to the client-side browser. Web servers usually run on TCP ports or , and are responsible for connecting end-users to various parts of the web application, in addition to handling their various responses.

WORKFLOW

A typical web server accepts HTTP requests from the client-side, and responds with different HTTP responses and codes, like a code 200 OK response for a successful request, a code 404 NOT FOUND when requesting pages that do not exist, code 403 FORBIDDEN for requesting access to restricted pages, and so on.

HTTP response code's

Databases

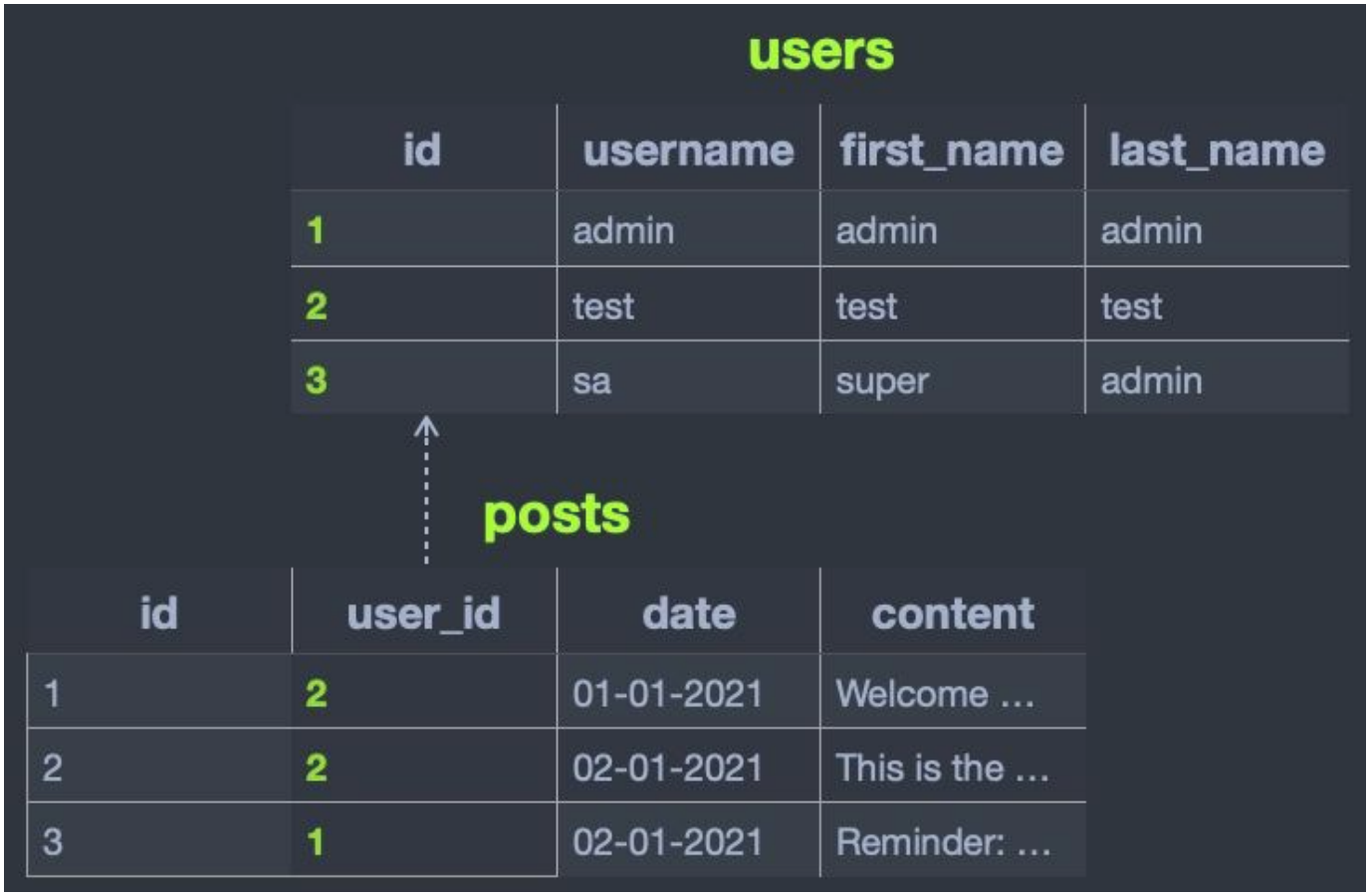
Web applications utilize back end databases to store various content and information related to the web application. This can be core web application assets like images and files, web application content like posts and updates, or user data like usernames and passwords. This allows web applications to easily and quickly store and retrieve data and enable dynamic content that is different for each user.

There are many different types of databases, each of which fits a certain type of use. Most developers look for certain characteristics in a database, such as speed in storing and retrieving data, size when storing large amounts of data, scalability as the web application grows, and cost.

RELATIONAL (SQL)

Relational (SQL) databases store their data in tables, rows, and columns. Each table can have unique keys, which can link tables together and create relationships between tables.

For example, we can have a users table in a relational database containing columns like id, username, first_name, last_name, and so on. The id can be used as the table key. Another table, posts, may contain posts made by all users, with columns like id, user_id, date, content, and so on.



We can link the id from the users table to the user_id in the posts table to easily retrieve the user details for each post, without having to store all user details with each post.

The relationship between tables within a database is called a Schema.

This way, by using relational databases, it becomes very quick and easy to retrieve all data about a certain element from all databases. For example, we can retrieve all details linked to a certain user from all tables with a single query.

This makes relational databases very fast and reliable for big datasets that have a clear structure and design. Databases also make data management very efficient.

Some of the most common relational databases include:

Type	Description
MySQL	The most commonly used database around the internet. It is an open-source database and can be used completely free of charge
MSSQL	Microsoft's implementation of a relational database. Widely used with Windows Servers and IIS web servers
Oracle	A very reliable database for big businesses, and is frequently updated with innovative database solutions to make it faster and more reliable. It can be costly, even for big businesses
PostgreSQL	Another free and open-source relational database. It is designed to be easily extensible, enabling adding advanced new features without needing a major change to the initial database design

Other common SQL databases include: , , , , and SQL.

NON-RELATIONAL (NOSQL)

A non-relational database does not use tables, rows, columns, primary keys, relationships, or schemas. Instead, a NoSQL database stores data using various storage models, depending on the type of data stored.

Due to the lack of a defined structure for the database, NoSQL databases are very scalable and flexible. When dealing with datasets that are not very well defined and structured, a NoSQL database would be the best choice for storing our data.

There are 4 common storage models for NoSQL databases:

- Key-Value
- Document-Based
- Wide-Column
- Graph

Each of the above models has a different way of storing data. For example, the Key-Value model usually stores data in JSON or XML, and has a key for each pair, storing all of its data as its value:

Some of the most common NoSQL databases include:

Type	Description
MongoDB	The most common NoSQL database. It is free and open-source, uses the
ElasticSearch	Another free and open-source NoSQL database. It is optimized for storing and analyzing huge datasets. As its name suggests, searching for data within this database is very fast and efficient
Apache Cassandra	Also free and open-source. It is very scalable and is optimized for gracefully handling faulty values

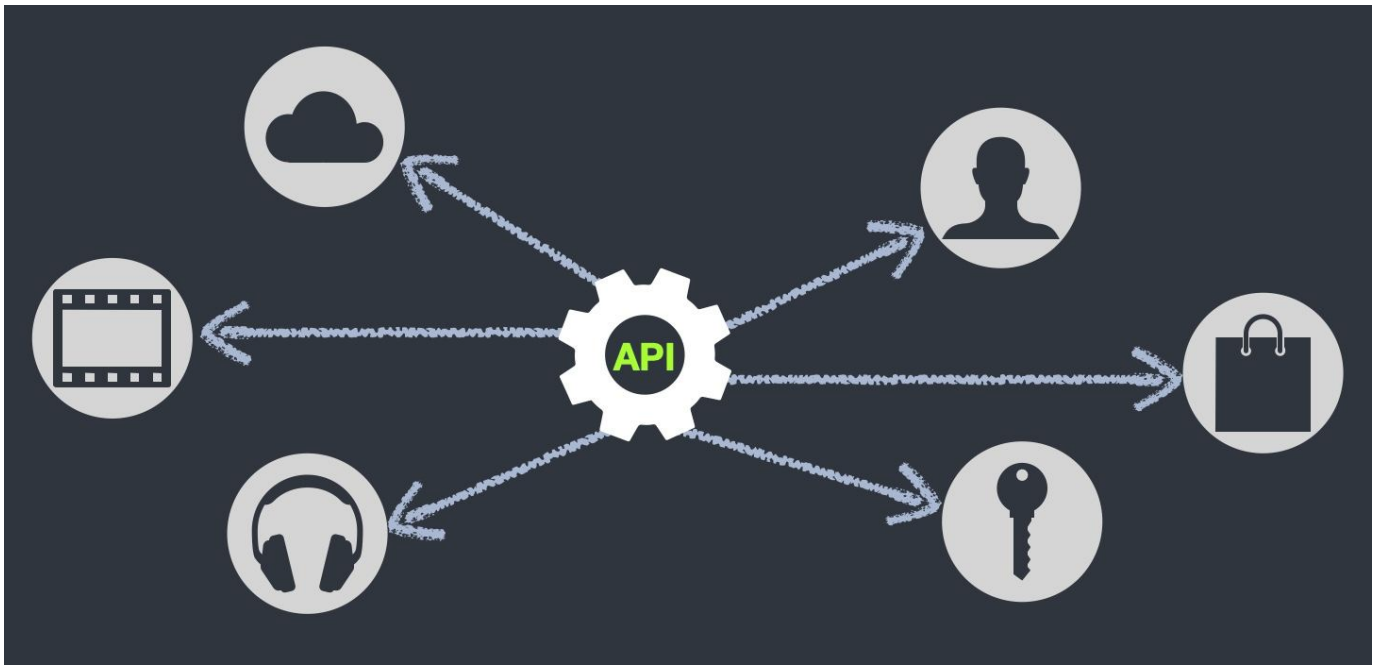
Other common NoSQL databases include: Redis, Neo4j, CouchDB, and Amazon DynamoDB.

Development Frameworks & APIs

In addition to web servers that can host web applications in various languages, there are many common web development frameworks that help in developing core web application files and functionality. With the increased complexity of web applications, it may be challenging to create a modern and sophisticated web application from scratch. Hence, most of the popular web applications are developed using web frameworks.

Web APIs

An API (Application Programming Interface) is an interface within an application that specifies how the application can interact with other applications. For Web Applications, it is what allows remote access to functionality on back end components. APIs are not exclusive to web applications and are used for software applications in general. Web APIs are usually accessed over the HTTP protocol and are usually handled and translated through web servers.



A weather web application, for example, may have a certain API to retrieve the current weather for a certain city. We can request the API URL and pass the city name or city id, and it would return the current weather in a JSON object. Another example is Twitter's API, which allows us to retrieve the latest Tweets from a certain account in XML or JSON formats, and even allows us to send a Tweet 'if authenticated', and so on.

To enable the use of APIs within a web application, the developers have to develop this functionality on the back end of the web application by using the API standards like or .

SOAP

The SOAP (Simple Objects Access) standard shares data through XML, where the request is made in XML through an HTTP request, and the response is also returned in XML. Front end components are designed to parse this XML output properly.

REST

The REST (Representational State Transfer) standard shares data through the URL path 'i.e. search/users/1', and usually returns the output in JSON format 'i.e. userid 1'.

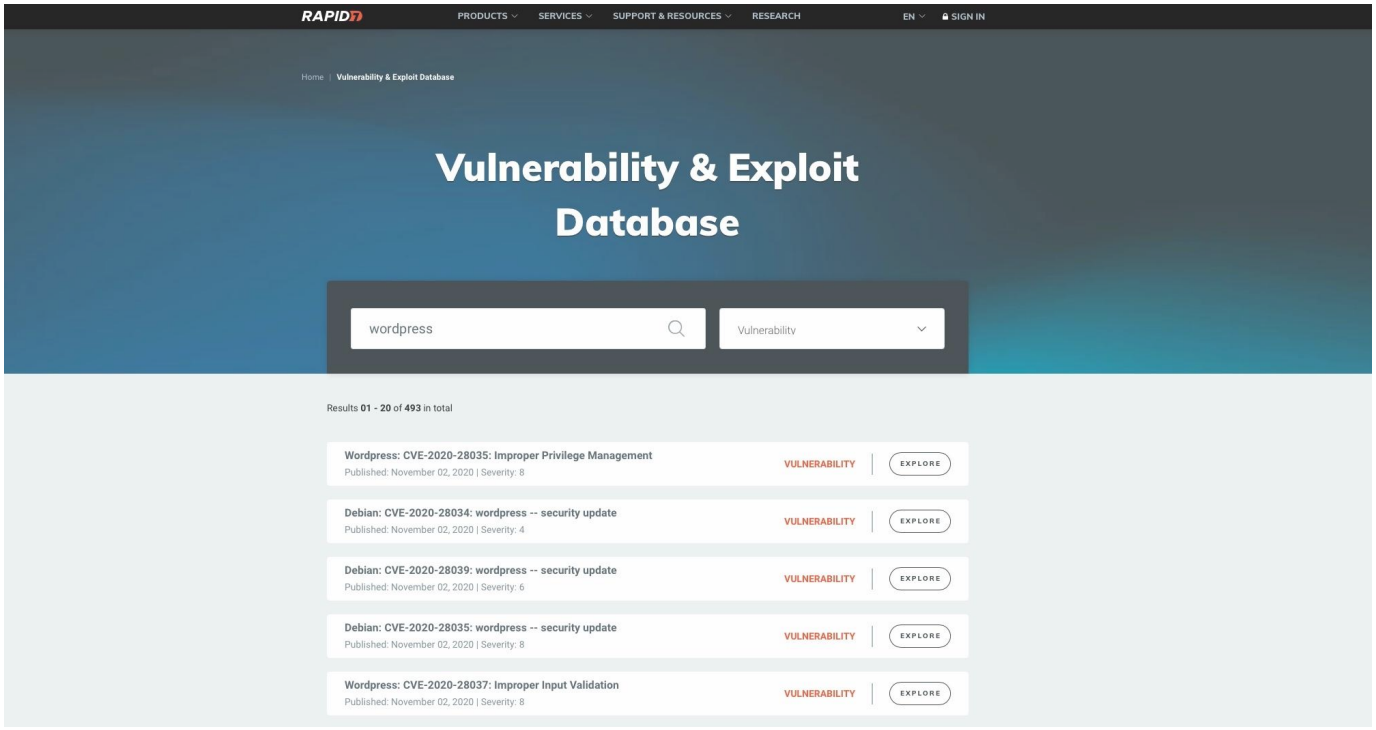
Public CVE

As many organizations deploy web applications that are publicly used, like open-source and proprietary web applications, these web applications tend to be tested by many organizations and experts around the world. This leads to frequently uncovering a large number of vulnerabilities, most of which get patched and then shared publicly and assigned a CVE (Common Vulnerabilities and Exposures) record and score.

Many penetration testers also make proof of concept exploits to test whether a certain public vulnerability can be exploited and usually make these exploits available for public use, for testing and educational purposes. This makes searching for public exploits the very first step we must go through for web applications

Tip: The first step is to identify the version of the web application. This can be found in many locations, like the source code of the web application. For open source web applications, we can check the repository of the web application and identify where the version number is shown (e.g,. in (version.php) page), and then check the same page on our target web application to confirm.

Once we identify the web application version, we can search Google for public exploits for this version of the web application. We can also utilize online exploit databases, like Exploit DB, Rapid7 DB, or Vulnerability Lab. The following example shows a search for WordPress public exploits in Rapid7 DB:



We would usually be interested in exploits with a CVE score of 8-10 or exploits that lead to . Other types of public exploits should also be considered if none of the above is available.

Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System (CVSS) is an open-source industry standard for assessing the severity of security vulnerabilities. This scoring system is often used as a standard measurement for organizations and governments that need to produce accurate and consistent severity scores for their systems' vulnerabilities. This helps with the prioritization of resources and the response to a given threat.

Nist search for CVSS

8.3 Web Requests

cURL cheat sheet

Command	Description
	cURL help menu
	Basic GET request
	Download file
	Skip HTTPS (SSL) certificate validation
	Print full HTTP request/response details
	Send HEAD request (only prints response headers)
	Print response headers and response body
	Set User-Agent header
	Set HTTP basic authorization credentials
	Pass HTT basic authorization credentials in the URL
	Set request header
	Pass GET parameters
	Send POST request with POST data
	Set request cookies
	Send POST request with JSON data

APIs

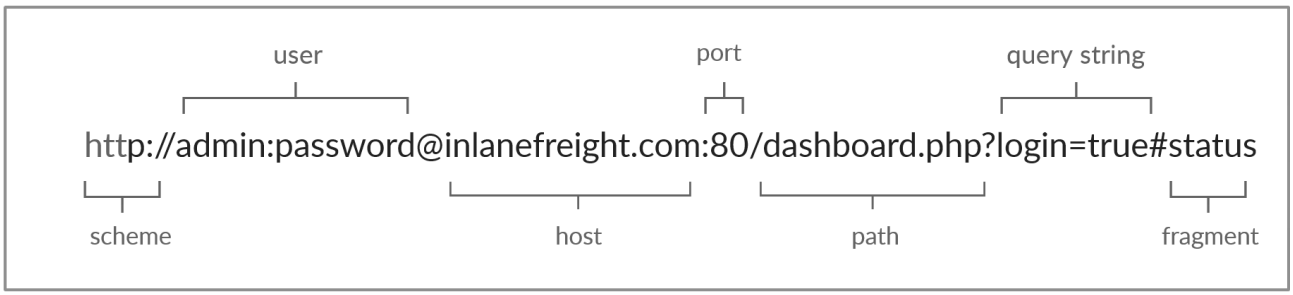
Command	Description
	Read entry
	Read all entries
	Create (add) entry
	Update (modify) entry
	Delete entry

Browser DevTools

Shortcut	Description
[ ] or [ ]	Show devtools
[ ]	Show Network tab
[ ]	Show Console tab

URL

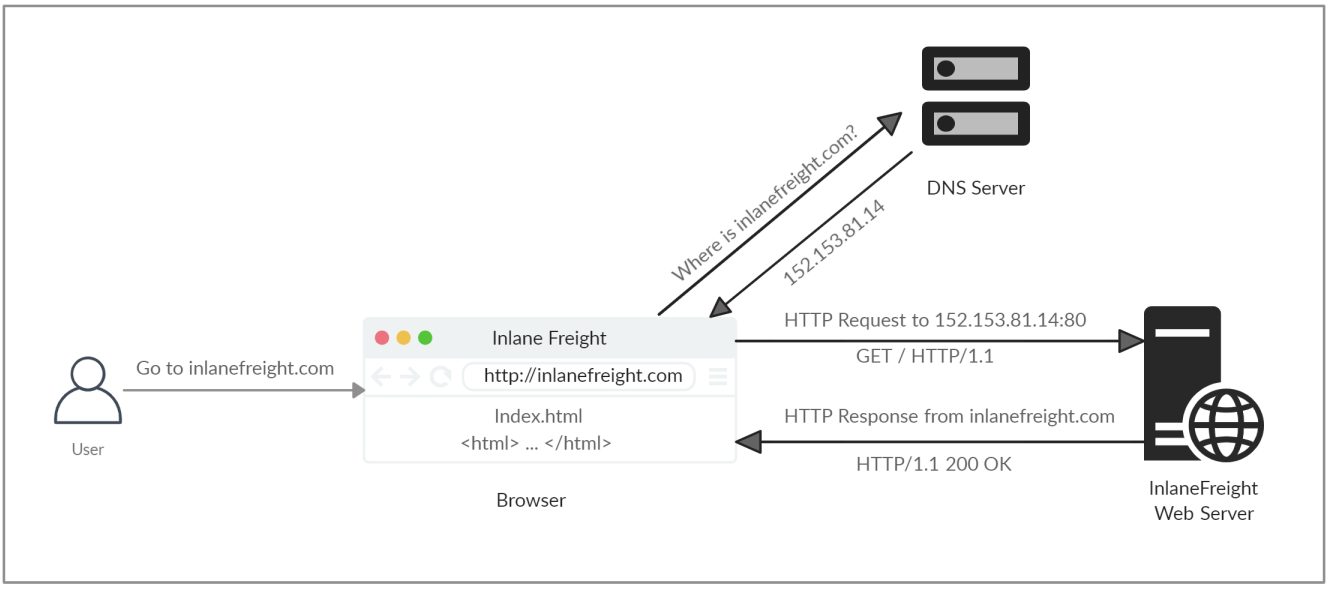
Resources over HTTP are accessed via a URL, which offers many more specifications than simply specifying a website we want to visit. Let's look at the structure of a URL:



Component	Example	Description
Scheme	hhrp:// or https://	This is used to identify the protocol being accessed by the client, and ends with a colon and a double slash (://
User info	admin:password@	This is an optional component that contains the credentials (separated by a colon :) used to authenticate to the host, and is separated from the host with an at sign (@)
host	inlanefreight.com	The host signifies the resource location. This can be a hostname or an IP address
Port	:80	The Port is separated from the Host by a colon (:). If no port is specified, http schemes default to port 80 and https default to port 443
Path	/dashboard.php	This points to the resource being accessed, which can be a file or a folder. If there is no path specified, the server returns the default index (e.g. index.html).
Query String	?login=true	The query string starts with a question mark (?), and consists of a parameter (e.g. login) and a value (e.g. true). Multiple parameters can be separated by an ampersand (&).
Fragments	#status	Fragments are processed by the browsers on the client-side to locate sections within the primary resource (e.g. a header or section on the page).

Not all components are required to access a resource. The main mandatory fields are the scheme and the host, without which the request would have no resource to request

HTTP Flow



The diagram above presents the anatomy of an HTTP request at a very high level. The first time a user enters the URL (inlanefreight.com) into the browser, it sends a request to a DNS (Domain Name Resolution) server to resolve the domain and get its IP. The DNS server looks up the IP address for inlanefreight.com and returns it. All domain names need to be resolved this way, as a server can't communicate without an IP address.

Note

Our browsers usually first look up records in the local '/etc/hosts' file, and if the requested domain does not exist within it, then they would contact other DNS servers. We can use the '/etc/hosts' to manually add records to for DNS resolution, by adding the IP followed by the domain name.

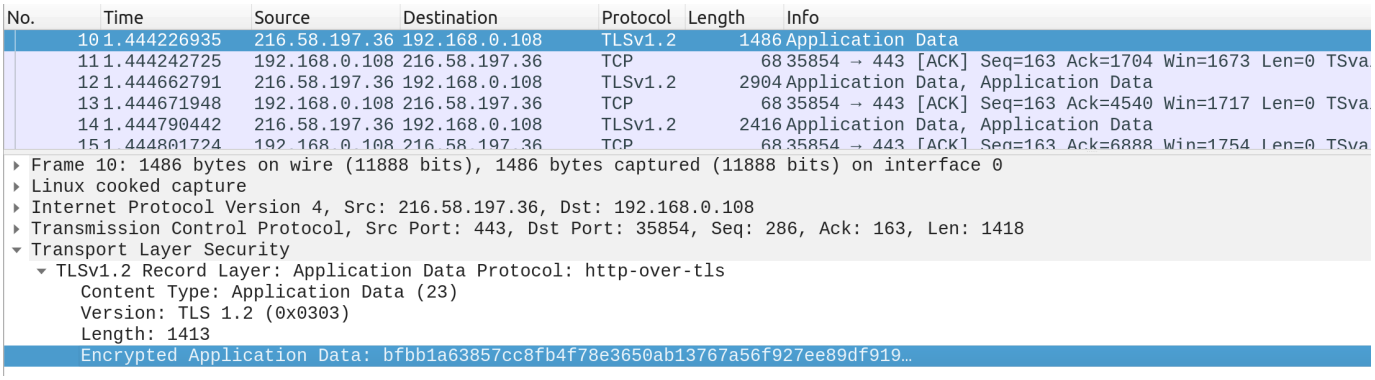
Hypertext Transfer Protocol Secure (HTTPS)

If we examine an HTTP request, we can see the effect of not enforcing secure communications between a web browser and a web application. For example, the following is the content of an HTTP login request:

```
      7 4.573774918   192.168.0.108 192.168.0.108   TCP          76 40386 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
      8 4.573794134   192.168.0.108 192.168.0.108   TCP          76 80 → 40386 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 S
      9 4.573806187   192.168.0.108 192.168.0.108   TCP          68 40386 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=280780439
    10 4.573966701   192.168.0.108 192.168.0.108   HTTP        640 POST /login.php HTTP/1.1  (application/x-www-form-urlencoded)
    11 4.573985767   192.168.0.108 192.168.0.108   TCP          68 80 → 40386 [ACK] Seq=1 Ack=573 Win=65024 Len=0 TSval=280780439
Frame 10: 640 bytes on wire (5120 bits), 640 bytes captured (5120 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.0.108, Dst: 192.168.0.108
Transmission Control Protocol, Src Port: 40386, Dst Port: 80, Seq: 1, Ack: 1, Len: 572
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "username" = "admin"
      Key: username
      Value: admin
  Form item: "password" = "password"
      Key: password
      Value: password
```

We can see that the login credentials can be viewed in clear-text. This would make it easy for someone on the same network (such as a public wireless network) to capture the request and reuse the credentials for malicious purposes.

In contrast, when someone intercepts and analyzes traffic from an HTTPS request, they would see something like the followin



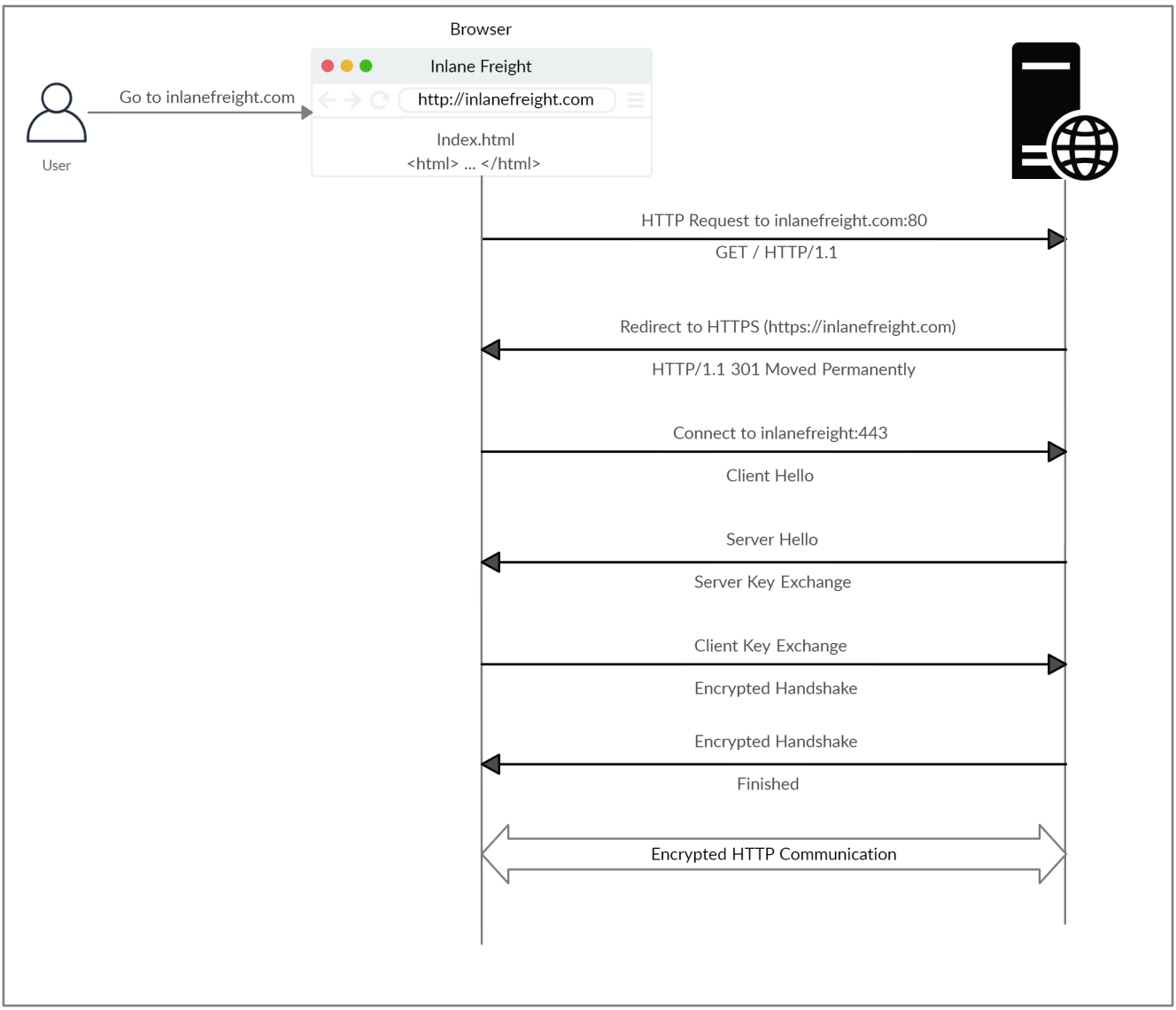
As we can see, the data is transferred as a single encrypted stream, which makes it very difficult for anyone to capture information such as credentials or any other sensitive data.

Note

Although the data transferred through the HTTPS protocol may be encrypted, the request may still reveal the visited URL if it contacted a clear-text DNS server. For this reason, it is recommended to utilize encrypted DNS servers (e.g. 8.8.8.8 or 1.2.3.4), or utilize a VPN service to ensure all traffic is properly encrypted.

HTTPS Flow

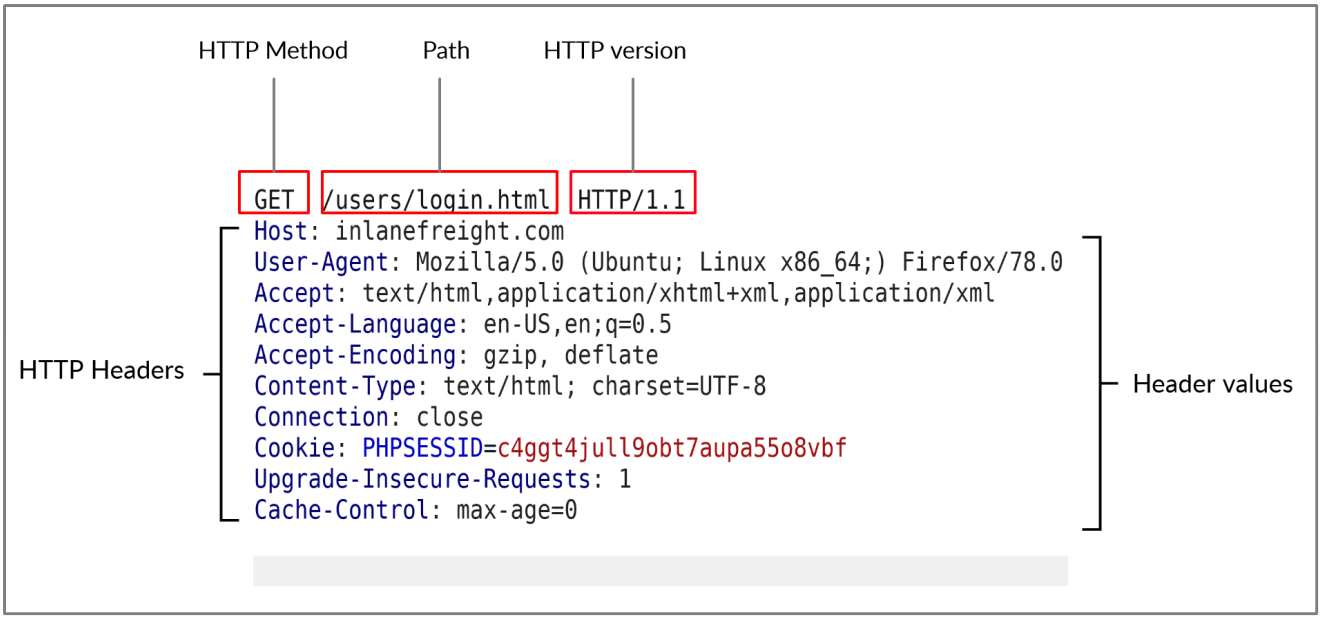
Let’s look at how HTTPS operates at a high level:



Next, the client (web browser) sends a “client hello” packet, giving information about itself. After this, the server replies with “server hello”, followed by a key exchange to exchange SSL certificates. The client verifies the key/certificate and sends one of its own. After this, an encrypted handshake is initiated to confirm whether the encryption and transfer are working correctly

HTTP Request

Let’s start by examining the following example HTTP request:

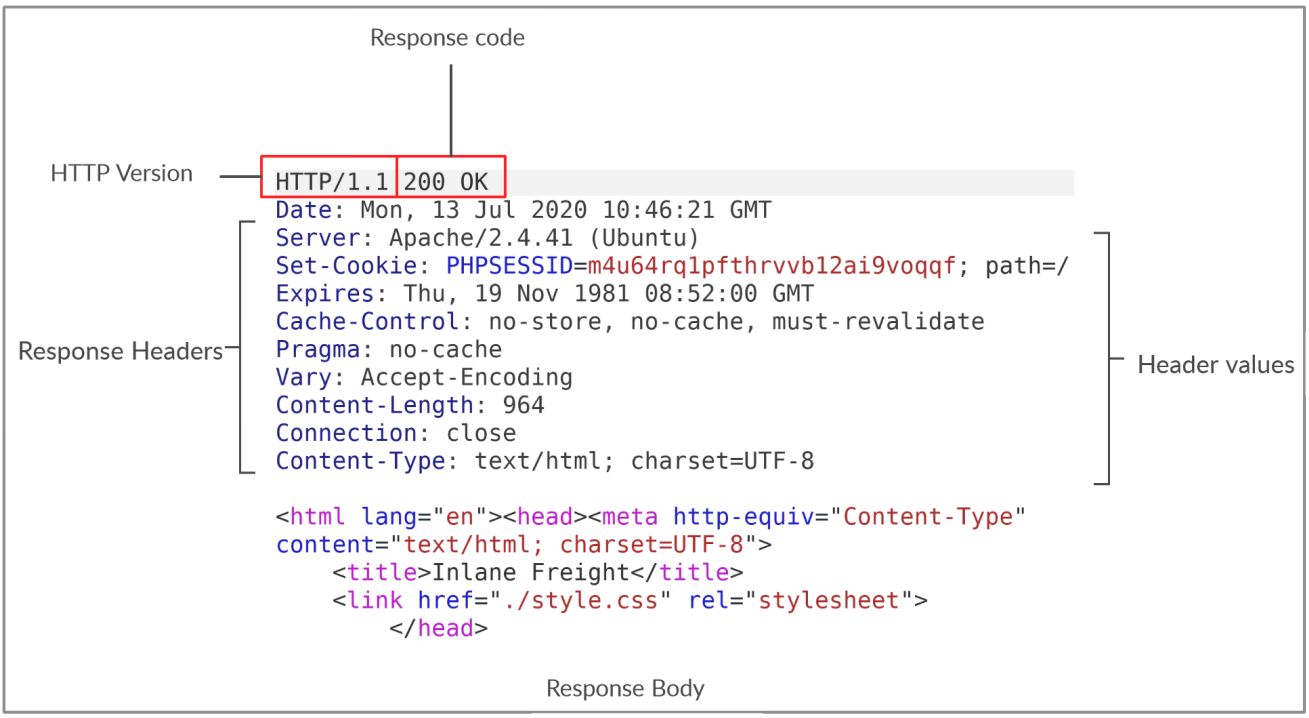


Note

Note: HTTP version 1.X sends requests as clear-text, and uses a new-line character to separate different fields and different requests. HTTP version 2.X, on the other hand, sends requests as binary data in a dictionary form.

HTTP Response

Once the server processes our request, it sends its response. The following is an example HTTP response:



The first line of an HTTP response contains two fields separated by spaces. The first being the HTTP version (e.g. HTTP/1.1), and the second denotes the (e.g.).

cURL

In our earlier examples with cURL, we only specified the URL and got the response body in return. However, cURL also allows us to preview the full HTTP request and the full HTTP response, which can become very handy when performing web penetration tests or writing exploits. To view the full HTTP request and response, we can simply add the -v verbose flag to our earlier commands, and it should print both the request and response:

HTTP Headers

Headers can have one or multiple values, appended after the header name and separated by a colon. We can divide headers into the following categories:

1. General Headers
2. Entity Headers
3. Request Headers
4. Response Headers
5. Security Headers

GENERAL HEADERS

General Headers are used in both HTTP requests and responses. They are contextual and are used to

Header	Example	Description
Date	Date: Wed, 16 Feb 2022 10:38:44 GMT	Holds the date and time at which the message originated. It's preferred to convert the time to the standard UTC time zone.
Connection	Connection: close	Dictates if the current network connection should stay alive after the request finishes. Two commonly used values for this header are close and keep-alive. The close value from either the client or server means that they would like to terminate the connection, while the keep-alive header indicates that the connection should remain open to receive more data and input.

ENTITY HEADERS

Similar to general headers, Entity Headers can be . These headers are used to transferred by a message. They are usually found in responses and POST or PUT requests.

Header	Example	Description
Content-Type	Content-Type: text/html	Used to describe the type of resource being transferred. The value is automatically added by the browsers on the client-side and returned in the server response. The charset field denotes the encoding standard, such as UTF-8.
Media-Type	Media-Type: application/pdf	The media-type is similar to Content-Type, and describes the data being transferred. This header can play a crucial role in making the server interpret our input. The charset field may also be used with this header.
Boundary	boundary="b4e4fbd93540"	Acts as a maker to separate content when there is more than one in the same message. For example, within a form data, this boundary gets used as –b4e4fbd93540 to separate different parts of the form.
Content-Length	Content-Length: 385	Holds the size of the entity being passed. This header is necessary as the server uses it to read data from the message body, and is automatically generated by the browser and tools like cURL
Content-Encoding	Content-Encoding: gzip	Data can undergo multiple transformations before being passed. For example, large amounts of data can be compressed to reduce the message size. The type of encoding being used should be specified using the Content-Encoding header.

REQUEST HEADERS

The client sends Request Headers in an HTTP transaction. These headers are of the message. The following headers are commonly seen in HTTP requests.

Header	Example	Description
Host	Host: www.inlanefreight.com	Used to specify the host being queried for the resource. This can be a domain name or an IP address. HTTP servers can be configured to host different websites, which are revealed based on the hostname. This makes the host header an important enumeration target, as it can indicate the existence of other hosts on the target server.
User-Agent	User-Agent: curl/7.77.0	The User-Agent header is used to describe the client requesting resources. This header can reveal a lot about the client, such as the browser, its version, and the operating system.
Referer	Referer: http://www.inlanefreight.com/	Denotes where the current request is coming from. For example, clicking a link from Google search results would make https://google.com the referer. Trusting this header can be dangerous as it can be easily manipulated, leading to unintended consequences
Accept	Accept: /	The Accept header describes which media types the client can understand. It can contains multiple media types separated by commas. The / value signifies that all media types are accepted.
Cookie	Cookie: PHPSESSID=b4e4fbd93540	Contains cookie-value pairs in the format name=value. A cookie is a piece of data stored on the client-side and on the server, which acts as an identifier. These are passed to the server per request, thus maintaining the client's access. Cookies can also serve other purposes, such as saving user preferences or session tracking. There can be multiple cookies in a single header separated by a semi-colon.
Authorization	Authorization: BASIC cGFzc3dvcmQK	Another method for the server to identify clients. After successful authentication, the server returns a token unique to the client. Unlike cookies, tokens are stored only on the client-side and retrieved by the server per request. There are multiple types of authentication types based on the webserver and application type used.

A complete list of request headers and their usage can be found here

RESPONSE HEADERS

Response Headers can be . Certain response headers such as Age, Location, and Server are used to provide more context about the response. The following headers are commonly seen in HTTP responses.

Header	Example	Description
Server	Server: Apache/2.2.14 (Win32)	Contains information about the HTTP server, which processed the request. It can be used to gain information about the server, such as its version, and enumerate it furthe
Set-Cookie	Set-Cookie: PHPSESSID=b4e4fbd93540	Contains the cookies needed for client identification. Browsers parse the cookies and store them for future requests. This header follows the same format as the Cookie request header.
WWW-Authenticate	WWW-Authenticate: BASIC realm="localhost"	Notifies the client about the type of authentication required to access the requested resource.

SECURITY HEADERS

Finally, we have Security Headers. With the increase in the variety of browsers and web-based attacks, defining certain headers that enhanced security was necessary. HTTP Security headers are a class of response headers used to specify certain rules and policies to be followed by the browser while accessing the websit.

Header	Example	Description
Content-Security-Policy	Content-Security-Policy: script-src 'self'	Dictates the website's policy towards externally injected resources. This could be JavaScript code as well as script resources. This header instructs the browser to accept resources only from certain trusted domains, hence preventing attacks such as Cross-site scripting (XSS).
Strict-Transport-Security	Strict-Transport-Security: max-age=31536000	Prevents the browser from accessing the website over the plaintext HTTP protocol, and forces all communication to be carried over the secure HTTPS protocol. This prevents attackers from sniffing web traffic and accessing protected information such as passwords or other sensitive dat
Referrer-Policy	Referrer-Policy: origin	Dictates whether the browser should include the value specified via the Referer header or not. It can help in avoiding disclosing sensitive URLs and information while browsing the website.

This section only mentions a small subset of commonly seen HTTP headers. There are many other contextual headers that can be used in HTTP communications

HTTP Methods and Codes

HTTP supports multiple methods for accessing a resource. In the HTTP protocol, several request methods allow the browser to send information, forms, or files to the server. These methods are used, among other things, to tell the server how to process the request we send and how to reply.

REQUEST METHODS

The following are some of the commonly used methods:

Method	Description
GET	Requests a specific resource. Additional data can be passed to the server via query strings in the URL (e.g. ?param=value).
POST	Sends data to the server. It can handle multiple types of input, such as text, PDFs, and other forms of binary data. This data is appended in the request body present after the headers. The POST method is commonly used when sending information (e.g. forms/logins) or uploading data to a website, such as images or documents.
HEAD	Requests the headers that would be returned if a GET request was made to the server. It doesn't return the request body and is usually made to check the response length before downloading resources.
PUT	Creates new resources on the server. Allowing this method without proper controls can lead to uploading malicious resources.
DELETE	Deletes an existing resource on the webserver. If not properly secured, can lead to Denial of Service (DoS) by deleting critical files on the web server
OPTIONS	Returns information about the server, such as the methods accepted by it.
PATCH	Applies partial modifications to the resource at the specified location.
CONNECT	Applies partial modifications to the resource at the specified location.
TRACE	The TRACE method performs a message loop-back test along the path to the target resource.

The list only highlights a few of the most commonly used HTTP methods. The availability of a particular method depends on the server as well as the application configuration. For a full list of HTTP methods, you can visit this link

RESPONSE CODES

HTTP status codes are used to tell the client the status of their request. An HTTP server can return five types of response codes:

Type	Description
1xx	Provides information and does not affect the processing of the request.
2xx	Returned when a request succeeds.
3xx	Returned when the server redirects the client.
4xx	Signifies improper requests from the client. For example, requesting a resource that doesn't exist or requesting a bad format.
5xx	Returned when there is some problem with the HTTP server itself.

The following are some of the commonly seen examples from each of the above HTTP method type

Code	Description
200 OK	Returned on a successful request, and the response body usually contains the requested resource.
302 Found	Redirects the client to another URL. For example, redirecting the user to their dashboard after a successful login.
400 Bad Request	Returned on encountering malformed requests such as requests with missing line terminators.
403 Forbidden	Signifies that the client doesn't have appropriate access to the resource. It can also be returned when the server detects malicious input from the user.
404 Not Found	Returned when the client requests a resource that doesn't exist on the server.
500 Internal Server Error	Returned when the server cannot process the request.

GET

Whenever we visit any URL, our browsers default to a GET request to obtain the remote resources hosted at that URL. Once the browser receives the initial page it is requesting; it may send other requests using various HTTP methods. This can be observed through the Network tab in the browser devtools, as seen in the previous section.

POST

Unlike HTTP GET, which places user parameters within the URL, HTTP POST places user parameters within the HTTP Request body. This has three main benefits:

- : As POST requests may transfer large files (e.g. file upload), it would not be efficient for the server to log all uploaded files as part of the requested URL, as would be the case with a file uploaded through a GET request.
- : URLs are designed to be shared, which means they need to conform to characters that can be converted to letters. The POST request places data in the body which can accept binary data. The only characters that need to be encoded are those that are used to separate parameters.
- : The maximum URL Length varies between browsers (Chrome/Firefox/IE), web servers (IIS, Apache, nginx), Content Delivery Networks (Fastly, Cloudfront, Cloudflare), and even URL Shorteners (bit.ly, amzn.to). Generally speaking, a URL's lengths should be kept to below 2,000 characters, and so they cannot handle a lot of data.

CRUD API

APIs

There are several types of APIs. Many APIs are used to interact with a database, such that we would be able to specify the requested table and the requested row within our API query, and then use an HTTP method to perform the operation needed. For example, for the api.php endpoint in our example, if we wanted to update the city table in the database, and the row we will be updating has a city name of london, then the URL would look something like this:

CRUD

As we can see, we can easily specify the table and the row we want to perform an operation on through such APIs. Then we may utilize different HTTP methods to perform different operations on that row. In general, APIs perform 4 main operations on the requested database entity:

Operation	HTTP Method	Description
Create	POST	Adds the specified data to the database table
Read	GET	Reads the specified entity from the database table
Updated	PUT	Updates the data of the specified database table
Delete	DELETE	Removes the specified row from the database table

Read

The first thing we will do when interacting with an API is reading data. As mentioned earlier, we can simply specify the table name after the API (e.g. /city) and then specify our search term (e.g. /london), as follows:

We see that the result is sent as a JSON string. To have it properly formatted in JSON format, we can pipe the output to the utility, which will format it properly. We will also silent any unneeded cURL output with , as follows:

Create

As this API is using JSON data, we will also set the Content-Type header to JSON, as follows:

slehee@htb[/htb]$ curl -X POST http://:/api.php/city/ -d ‘{“city_name”:”HTB_City”, “country_name”:”HTB”}’ -H ‘Content-Type: application/json’

Update

Now that we know how to read and write entries through APIs, let’s start discussing two other HTTP methods we have not used so far: and . As mentioned at the beginning of the section, is used to updated API entries and modify their details, while is used to remove a specific entity.

Note

The HTTP method may also be used to update API entries instead of . To be precise, is used to partially update an entry (only modify some of its data “e.g. only city_name”), while is used to update the entire entry. We may also use the HTTP OPTIONS method to see which of the two is accepted by the server, and then use the appropriate method accordingly. In this section, we will be focusing on the method, though their usage is quite similar.

Using is quite similar to in this case, with the only difference being that we have to specify the name of the entity we want to edit in the URL, otherwise the API will not know which entity to edit. So, all we have to do is specify the city name in the URL, change the request method to , and provide the JSON data like we did with , as follows:

DELETE

8.4 Web Proxies

Web Proxies

Today, most modern web and mobile applications work by continuously connecting to back-end servers to send and receive data and then processing this data on the user's device, like their web browsers or mobile phones. With most applications heavily relying on back-end servers to process data, testing and securing the back-end servers is quickly becoming more important.

Testing web requests to back-end servers make up the bulk of Web Application Penetration Testing, which includes concepts that apply to both web and mobile applications. To capture the requests and traffic passing between applications and back-end servers and manipulate these types of requests for testing purposes, we need to use Web Proxies.

What Are Web Proxies?

Web proxies are specialized tools that can be set up between a browser/mobile application and a back-end server to capture and view all the web requests being sent between both ends, essentially acting as man-in-the-middle (MITM) tools.

Uses of Web Proxies

- Web application vulnerability scanning
- Web fuzzing
- Web crawling
- Web application mapping
- Web request analysis
- Web configuration testing
- Code reviews

Burp Suite

Burp Suite (Burp) - is the most common web proxy for web penetration testing. It has an excellent user interface for its various features and even provides a built-in Chromium browser to test web applications. Certain Burp features are only available in the commercial version Burp Pro/Enterprise, but even the free version is an extremely powerful testing tool to keep in our arsenal.

Some of the features are:

- Active web app scanner
- Fast Burp Intruder
- The ability to load certain Burp Extensions

The community free version of Burp Suite should be enough for most penetration testers. Once we start more advanced web application penetration testing, the pro features may become handy. Most of the features we will cover in this module are available in the community free version of Burp Suite, but we will also touch upon some of the pro features, like the Active Web App Scanner.

Tip: If you have an educational or business email address, then you can apply for a free trial of Burp Pro at this link to be able to follow along with some of the Burp Pro only features showcased later in this module.

If you have an educational or business email address, then you can apply for a free trial of Burp Pro at this link to be able to follow along with some of the Burp Pro only features showcased later in this module.

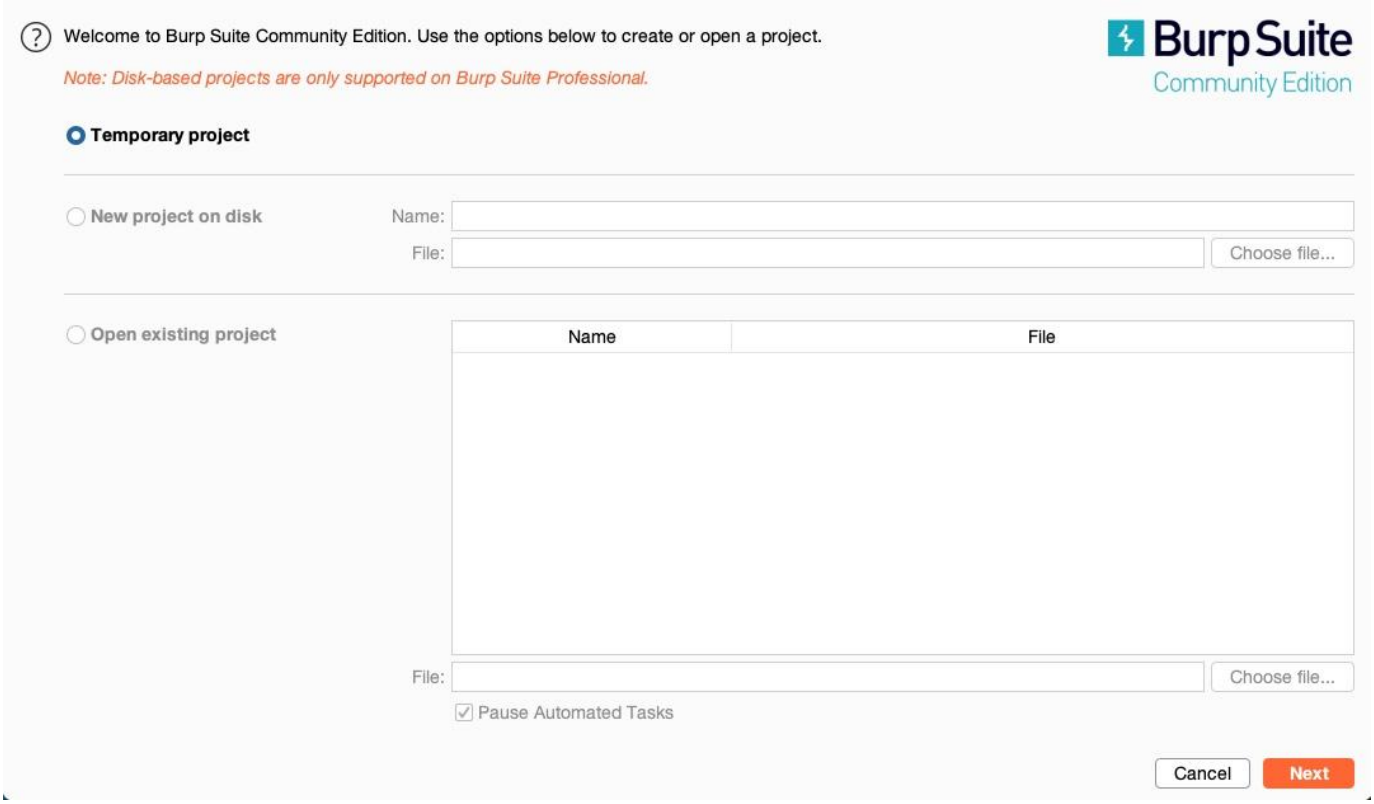
OWASP Zed Attack Proxy (ZAP)

OWASP Zed Attack Proxy (ZAP) is another common web proxy tool for web penetration testing. ZAP is a free and open-source project initiated by the Open Web Application Security Project (OWASP) and maintained by the community, so it has no paid-only features as Burp does. It has grown significantly over the past few years and is quickly gaining market recognition as the leading open-source web proxy tool.

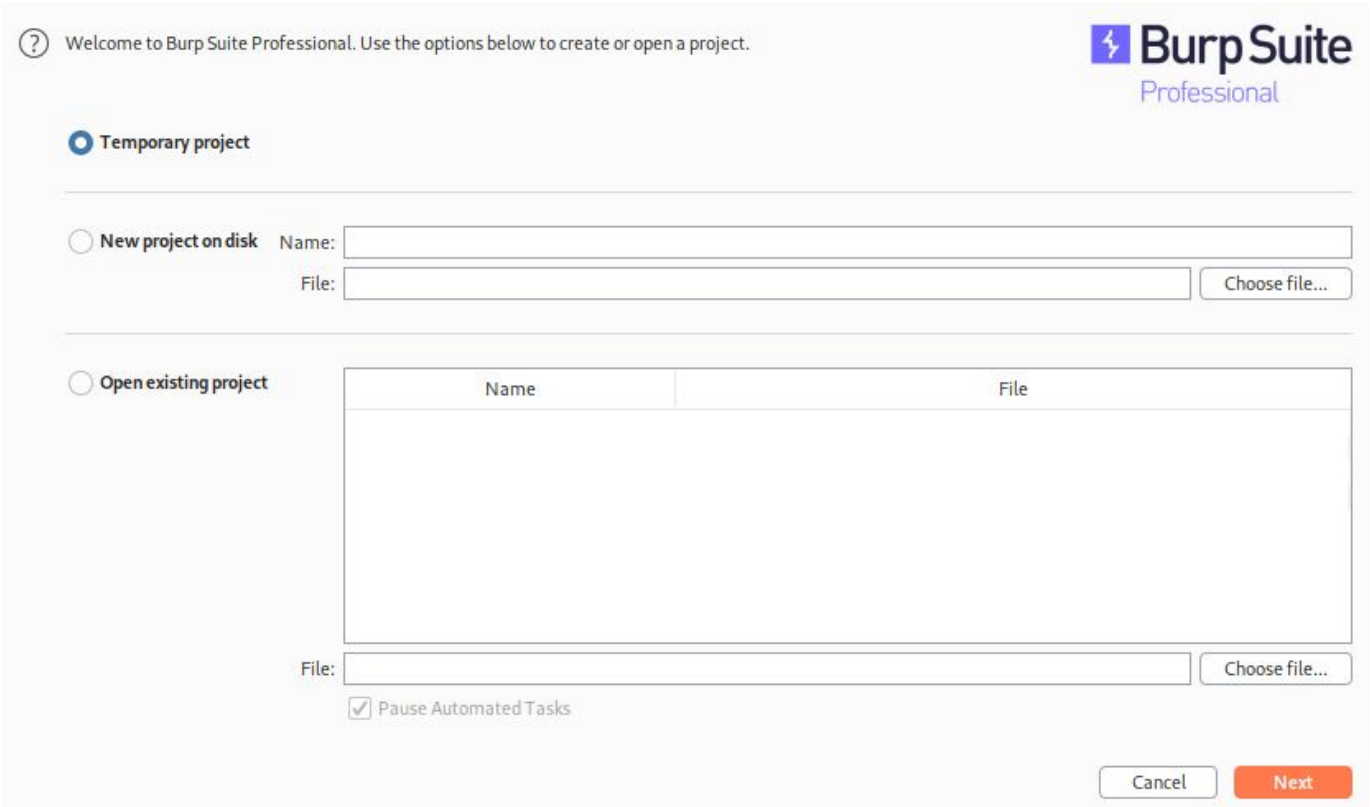
In the end, learning both tools can be quite similar and will provide us with options for every situation through a web pentest, and we can choose to use whichever one we find more suitable for our needs. In some instances, we may not see enough value to justify a paid Burp subscription, and we may switch ZAP to have a completely open and free experience. In other situations where we want a more mature solution for advanced pentests or corporate pentesting, we may find the value provided by Burp Pro to be justified and may switch to Burp for these features.

Setting Up

Once we start up Burp, we are prompted to create a new project. If we are running the community version, we would only be able to use temporary projects without the ability to save our progress and carry on later:

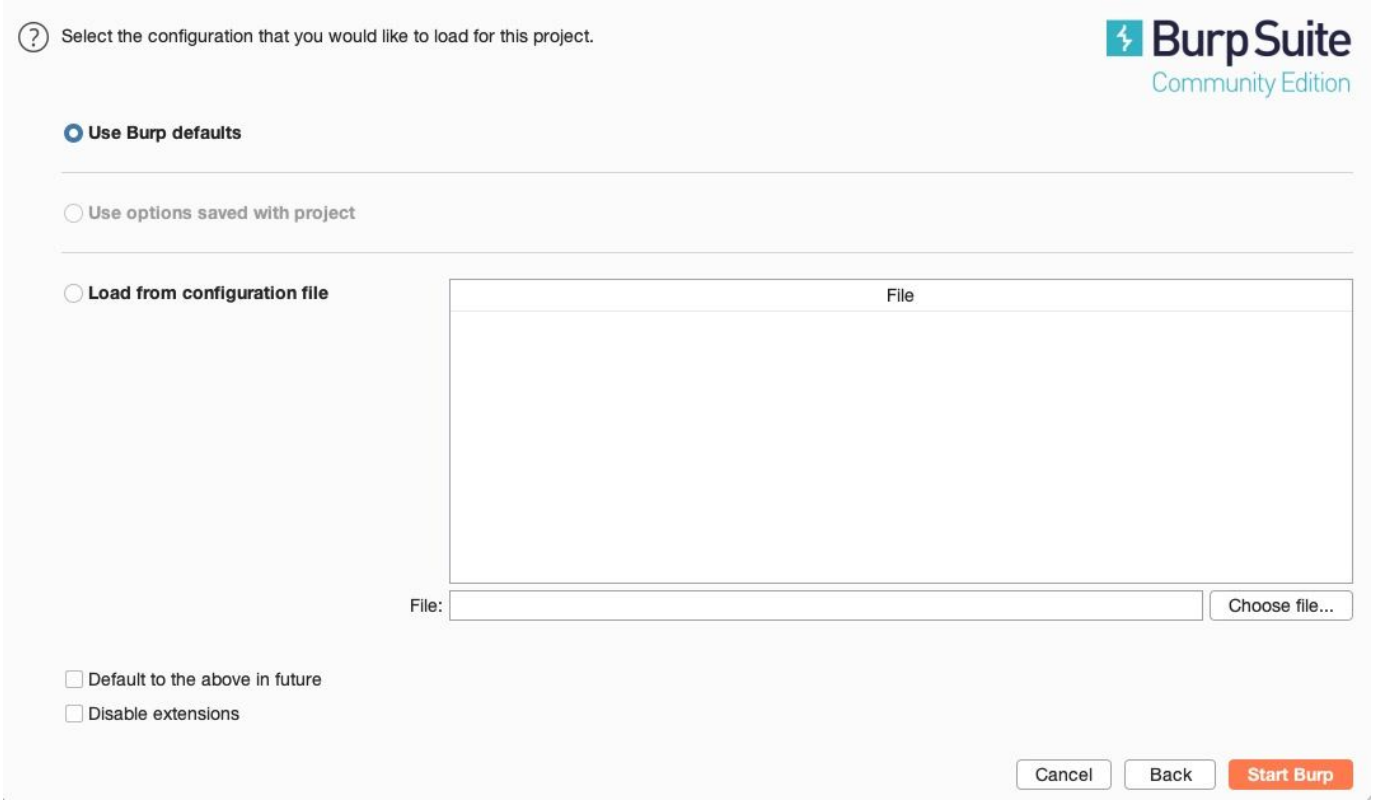


If we are using the pro/enterprise version, we will have the option to either start a new project or open an existing project.



We may need to save our progress if we were pentesting huge web applications or running an . However, we may not need to save our progress and, in many cases, can start a y project every time.

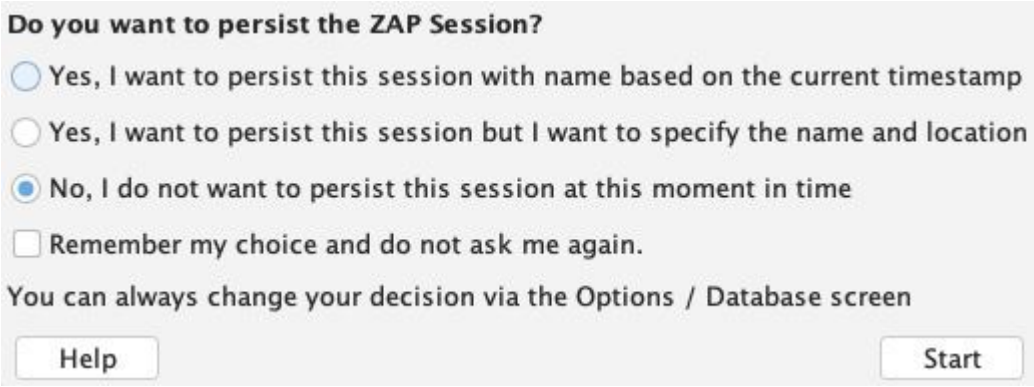
So, let's select project, and click continue. Once we do, we will be prompted to either use , or to , and we'll choose the first option:



ZAP

We can download ZAP from its download page, choose the installer that fits our operating system, and follow the basic installation instructions to get it installed.

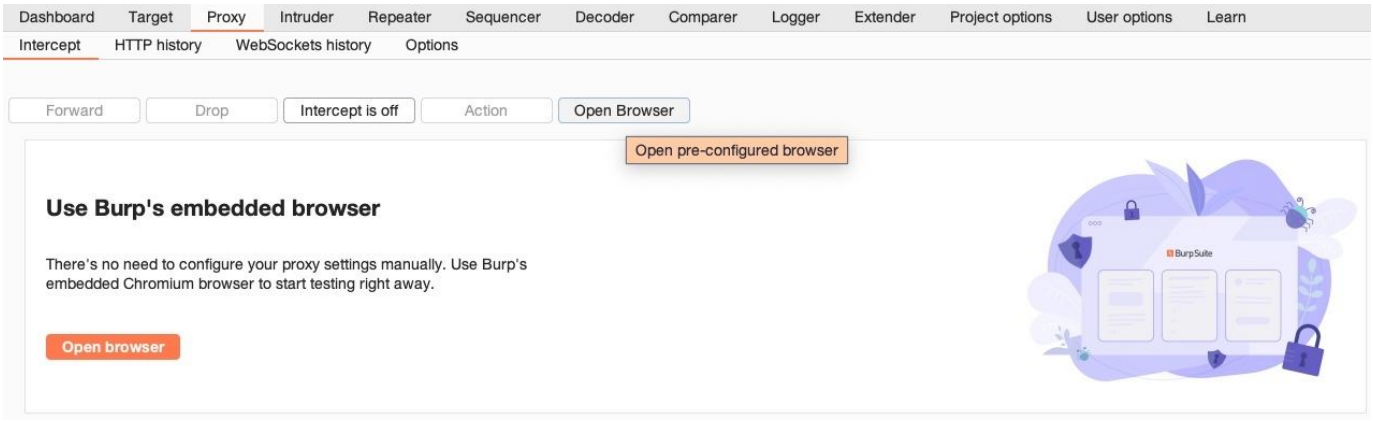
To get started with ZAP, we can launch it from the terminal with the command or access it from the application menu like Burp.



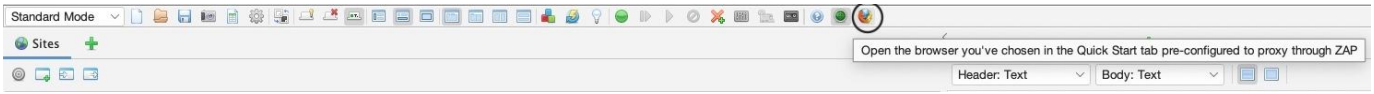
Pre-Configured Browser

To use the tools as web proxies, we must configure our browser proxy settings to use them as the proxy or use the pre-configured browser. Both tools have a pre-configured browser that comes with pre-configured proxy settings and the CA certificates pre-installed, making starting a web penetration test very quick and easy.

In Burp's , we can click on , which will open Burp's pre-configured browser, and automatically route all web traffic through Burp:



In ZAP, we can click on the Firefox browser icon at the end of the top bar, and it will open the pre-configured browser:



Proxy Setup

In many cases, we may want to use a real browser for pentesting, like Firefox. To use Firefox with our web proxy tools, we must first configure it to use them as the proxy. We can manually go to Firefox preferences and set up the proxy to use the web proxy listening port. Both Burp and ZAP use port by default, but we can use any available port. If we choose a port that is in use, the proxy will fail to start, and we will receive an error message.

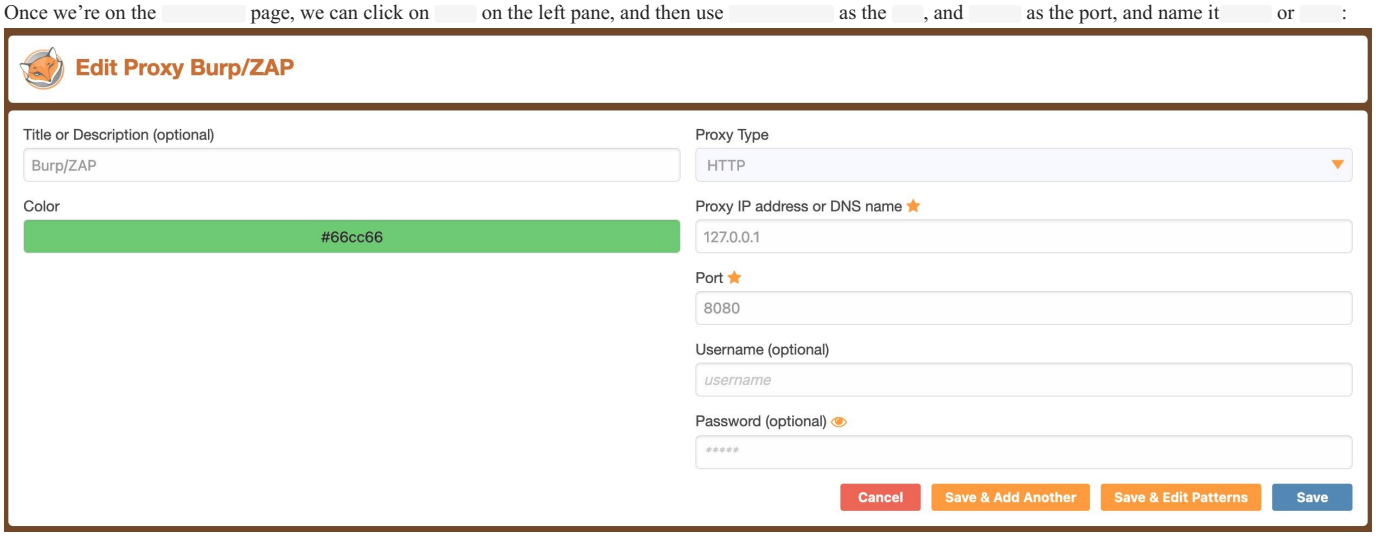
Note

In case we wanted to serve the web proxy on a different port, we can do that in Burp under (Proxy>Options), or in ZAP under (Tools>Options>Local Proxies). In both cases, we must ensure that the proxy configured in Firefox uses the same port.

Instead of manually switching the proxy, we can utilize the Firefox extension to easily and quickly change the Firefox proxy.

Once we have the extension added, we can configure the web proxy on it by clicking on its icon on Firefox top bar and then choosing options:

Once we're on the page, we can click on on the left pane, and then use as the , and as the port, and name it or :

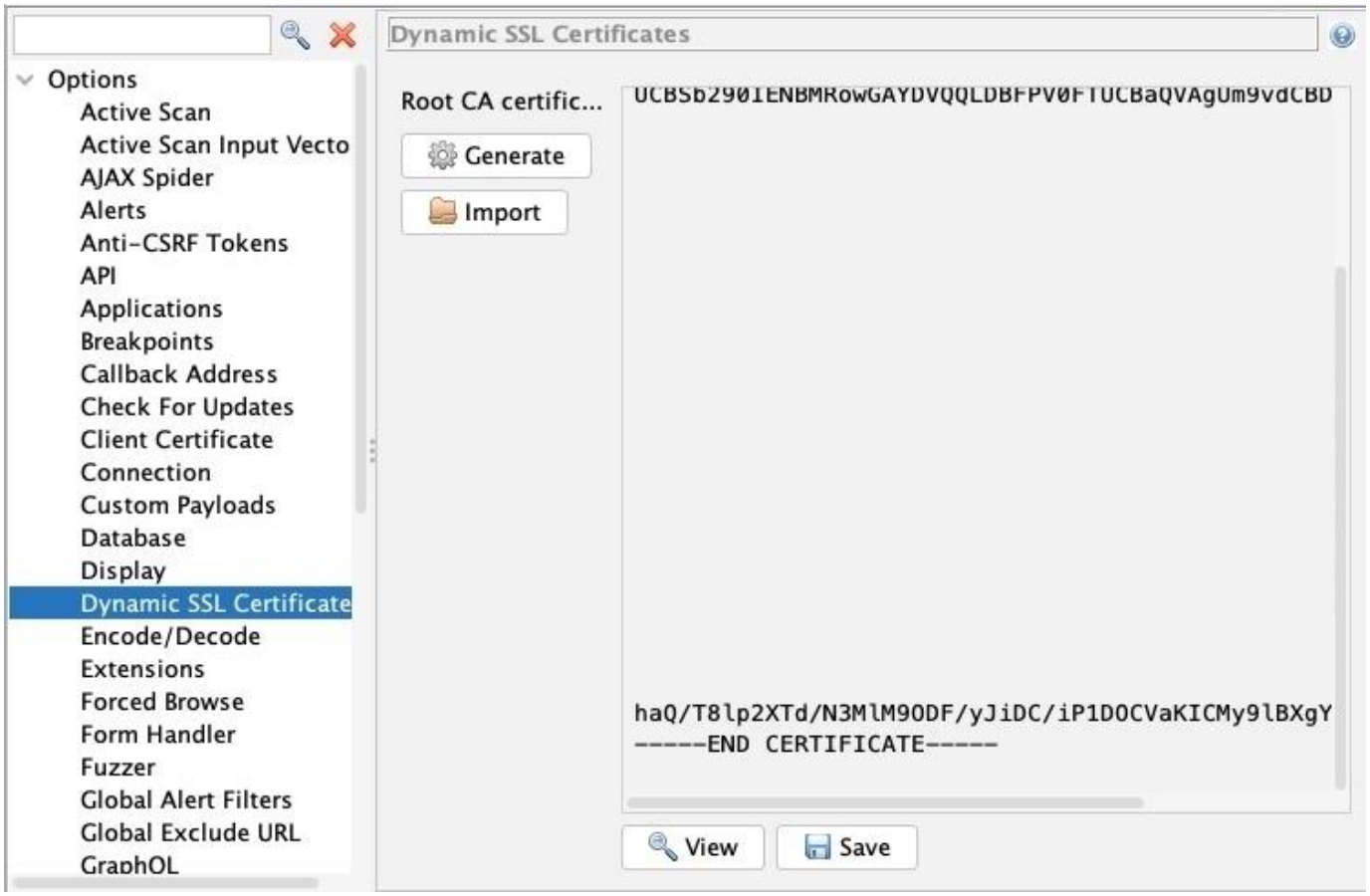


Installing CA Certificate

Another important step when using Burp Proxy/ZAP with our browser is to install the web proxy's CA Certificates. If we don't do this step, some HTTPS traffic may not get properly routed, or we may need to click every time Firefox needs to send an HTTPS request.

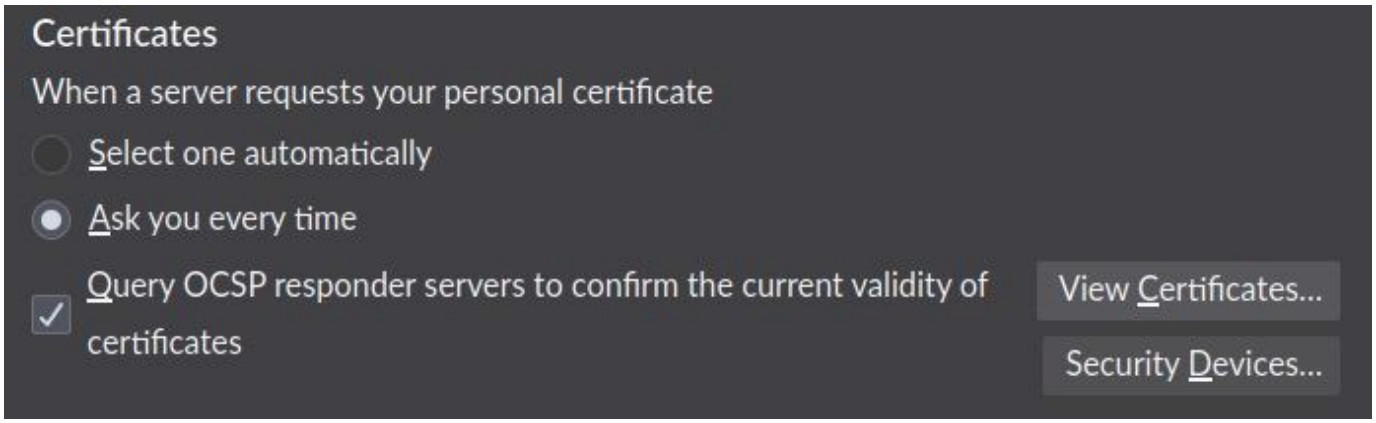
We can install Burp's certificate once we select Burp as our proxy in , by browsing to , and download the certificate from there by clicking on :

To get ZAP's certificate, we can go to , then click on :

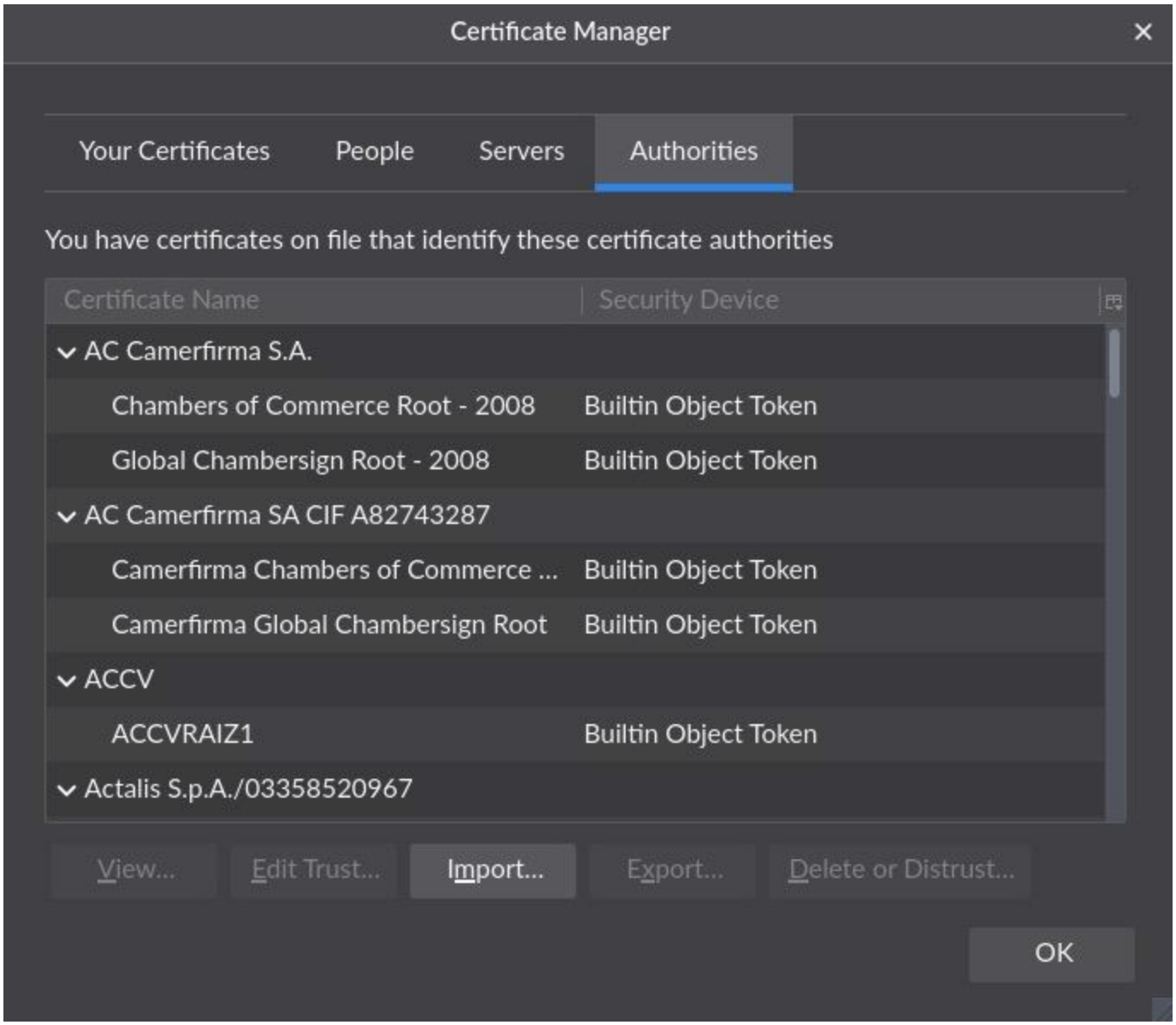


We can also change our certificate by generating a new one with the button.

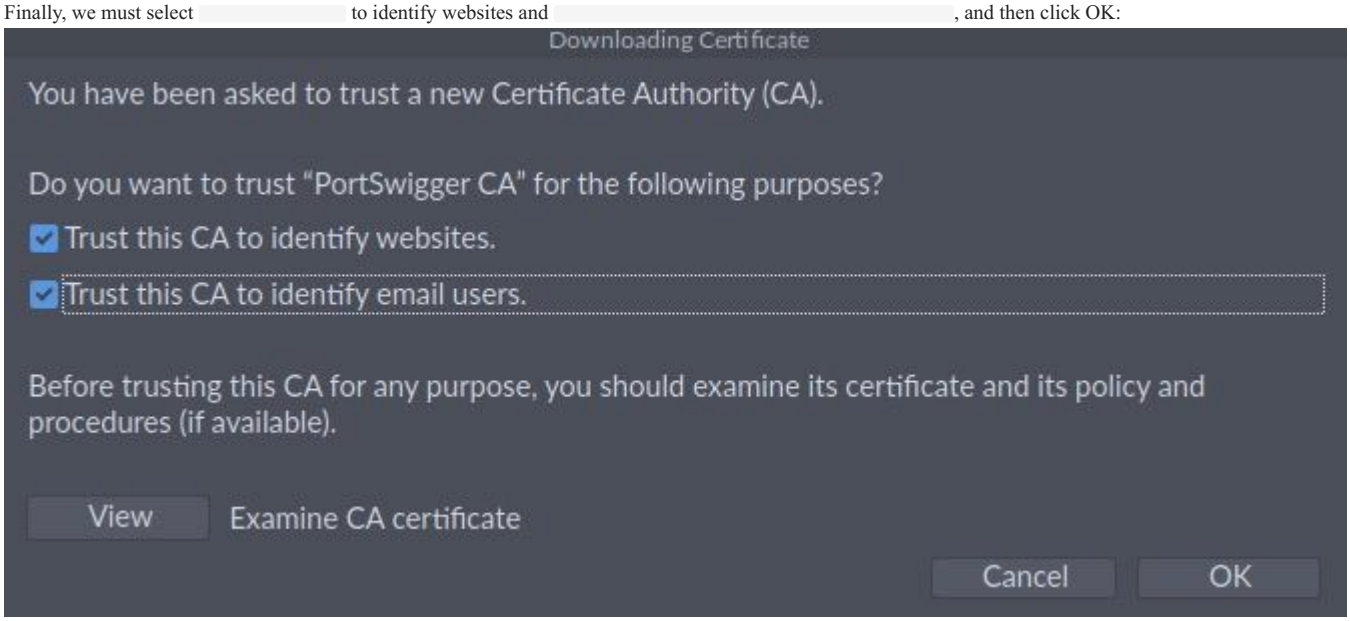
Once we have our certificates, we can install them within Firefox by browsing to , scrolling to the bottom, and clicking View :



After that, we can select the tab, and then click on , and select the downloaded CA certificate:



Finally, we must select to identify websites and , and then click OK:



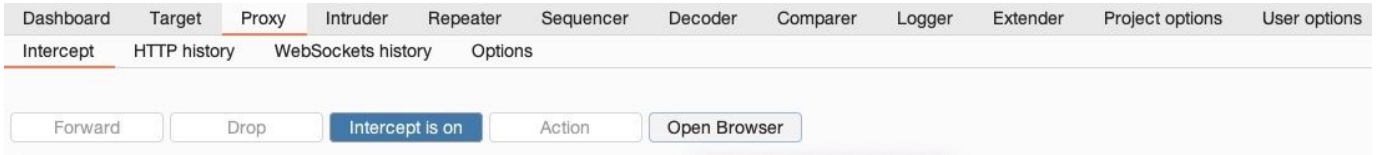
Once we install the certificate and configure the Firefox proxy, all Firefox web traffic will start routing through our web proxy.

Intercepting Web Requests

Now that we have set up our proxy, we can use it to intercept and manipulate various HTTP requests sent by the web application we are testing. We'll start by learning how to intercept web requests, change them, and then send them through to their intended destination.

BURP

In Burp, we can navigate to the tab, and request interception should be on by default. If we want to turn request interception on or off, we may go to the sub-tab and click on button to do so:



ZAP

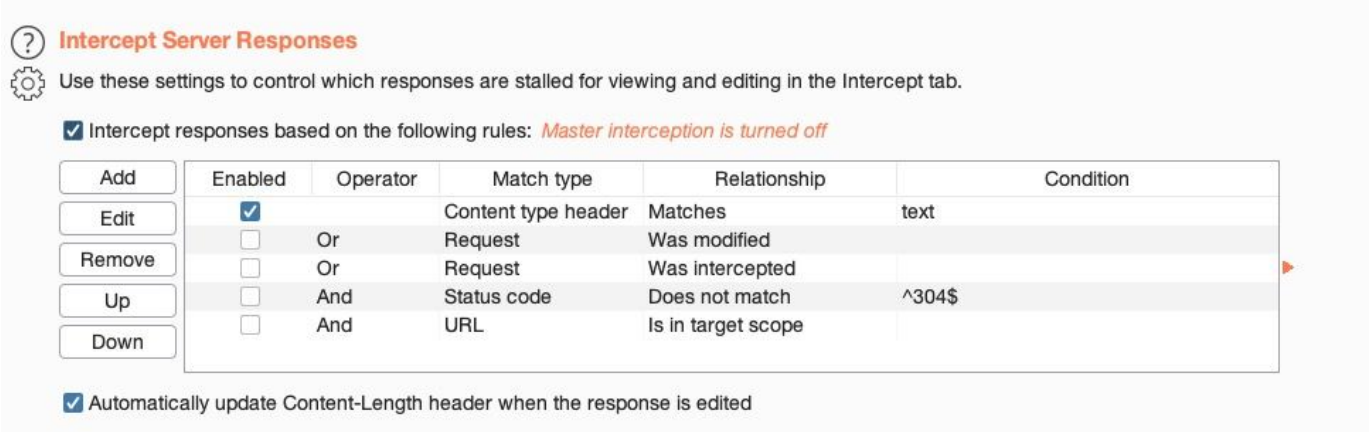
In ZAP, interception is off by default, as shown by the green button on the top bar (green indicates that requests can pass and not be intercepted). We can click on this button to turn the Request Interception on or off, or we can use the shortcut [CTRL+B] to toggle it on or off:

Intercepting Responses

In some instances, we may need to intercept the HTTP responses from the server before they reach the browser. This can be useful when we want to change how a specific web page looks, like enabling certain disabled fields or showing certain hidden fields, which may help us in our penetration testing activities.

BURP

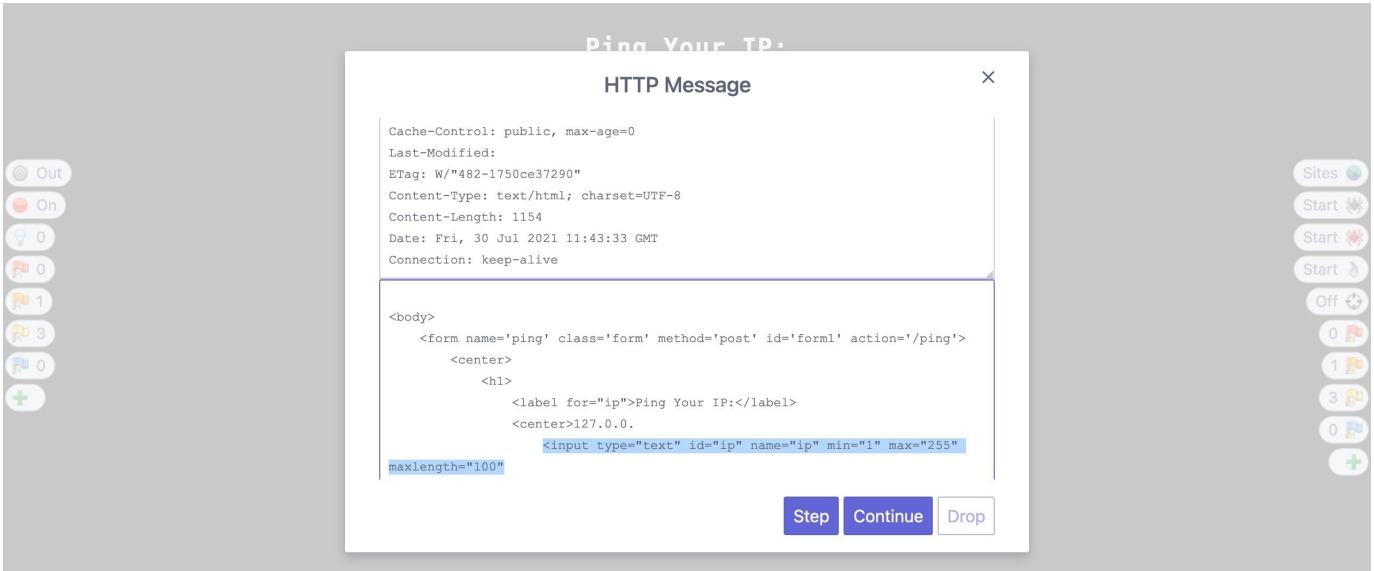
In Burp, we can enable response interception by going to and enabling under :



After that, we can enable request interception once more and refresh the page with [CTRL+SHIFT+R] in our browser (to force a full refresh). When we go back to Burp, we should see the intercepted request, and we can click on forward. Once we forward the request, we'll see our intercepted response:

ZAP

Let's try to see how we can do the same with ZAP, when our requests are intercepted by ZAP, we can click on , and it will send the request and automatically intercept the response:



However, ZAP HUD also has another powerful feature that can help us in cases like this. While in many instances we may need to intercept the response to make custom changes, if all we wanted was to enable disabled input fields or show hidden input fields, then we can click on the third button on the left (the light bulb icon), and it will enable/show these fields without us having to intercept the response or refresh the page.

Automatic Modification

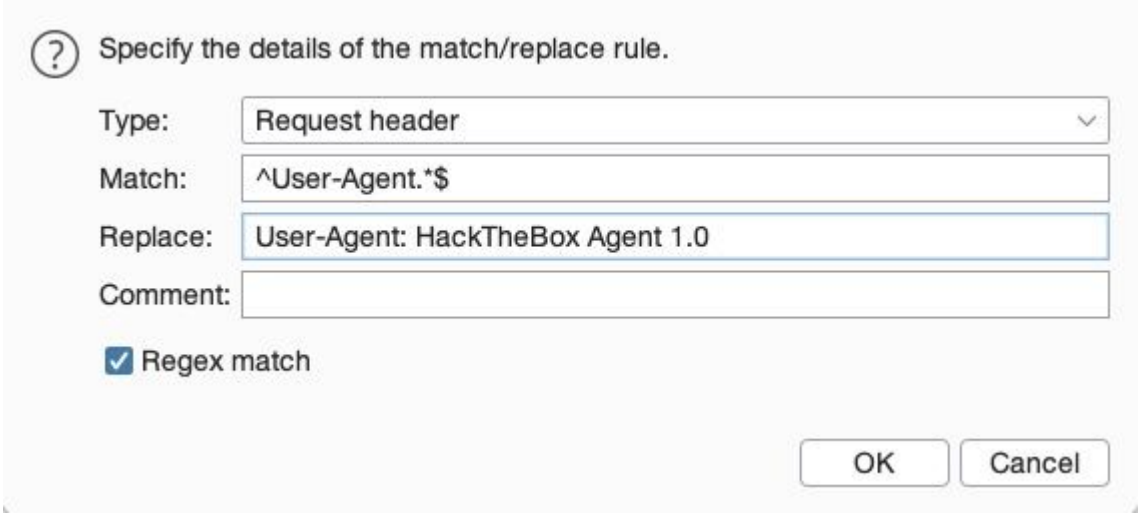
We may want to apply certain modifications to all outgoing HTTP requests or all incoming HTTP responses in certain situations. In these cases, we can utilize automatic modifications based on rules we set, so the web proxy tools will automatically apply them.

Automatic Request Modification

We can choose to match any text within our requests, either in the request header or request body, and then replace them with different text.

Burp Match and Replace

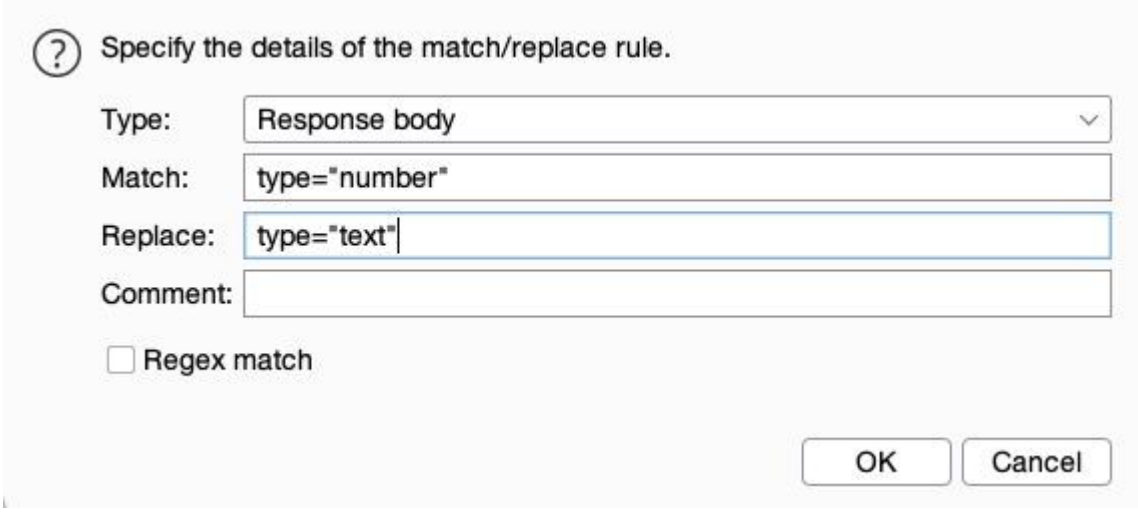
We can go to and click on in Burp. As the below screenshot shows, we will set the following options:



Once we enter the above options and click Ok, our new Match and Replace option will be added and enabled and will start automatically replacing the User-Agent header in our requests with our new User-Agent. We can verify that by visiting any website using the pre-configured Burp browser and reviewing the intercepted request. We will see that our User-Agent has indeed been automatically replaced:

Automatic Response Modification

Go to in Burp to add another rule. This time we will use the type of Response body since the change we want to make exists in the response's body and not in its headers. In this case, we do not have to use regex as we know the exact string we want to replace, though it is possible to use regex to do the same thing if we prefer.



Repeating Requests

We successfully bypassed the input validation to use a non-numeric input to reach on the remote server.

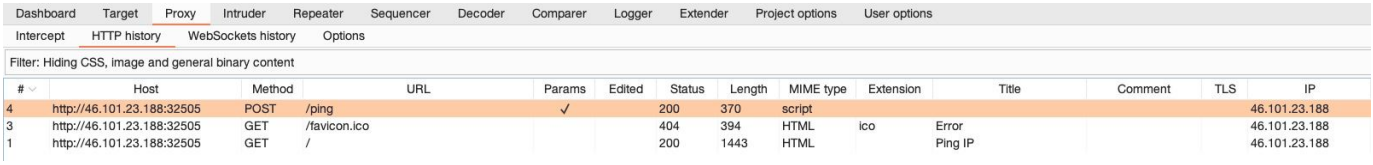
As you can imagine, if we would do this for each command, it would take us forever to enumerate a system, as each command would require 5-6 steps to get executed. However, for such repetitive tasks, we can utilize request repeating to make this process significantly easier.

Request repeating allows us to resend any web request that has previously gone through the web proxy.

This allows us to make quick changes to any request before we send it, then get the response within our tools without intercepting and modifying each request.

Proxy History

To start, we can view the HTTP requests history in at :



In ZAP HUD, we can find it in the bottom History pane or ZAP's main UI at the bottom History tab as well:

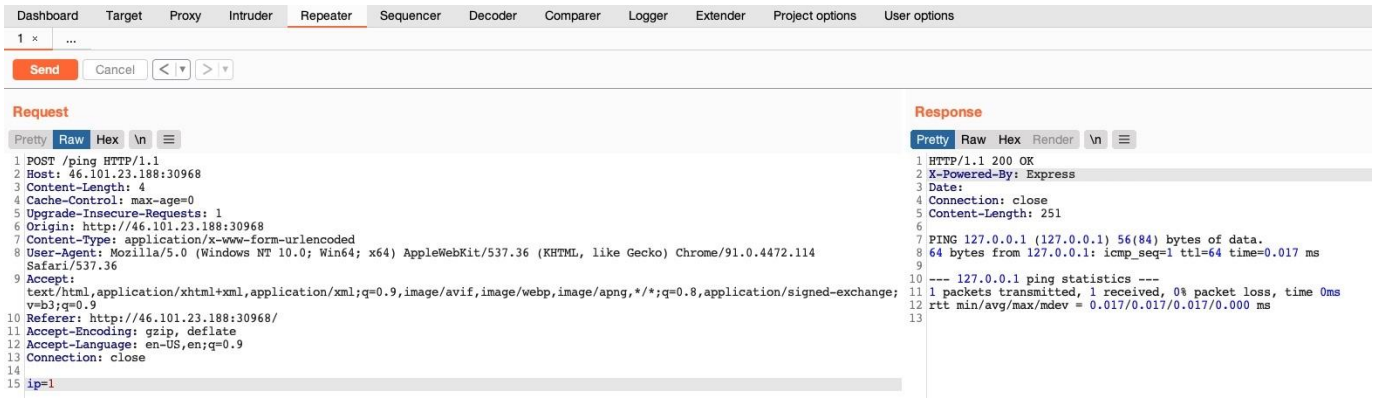
Note

Both tools also maintain WebSockets history, which shows all connections initiated by the web application even after being loaded, like asynchronous updates and data fetching. WebSockets can be useful when performing advanced web penetration testing, and are out of the scope of this module.

If we click on any request in the history in either tool, its details will be shown:

Repeating Requests

Once we locate the request we want to repeat, we can click [CTRL+R] in Burp to send it to the tab, and then we can either navigate to the Repeater tab or click [CTRL+SHIFT+R] to go to it directly. Once in , we can click on to send the request:

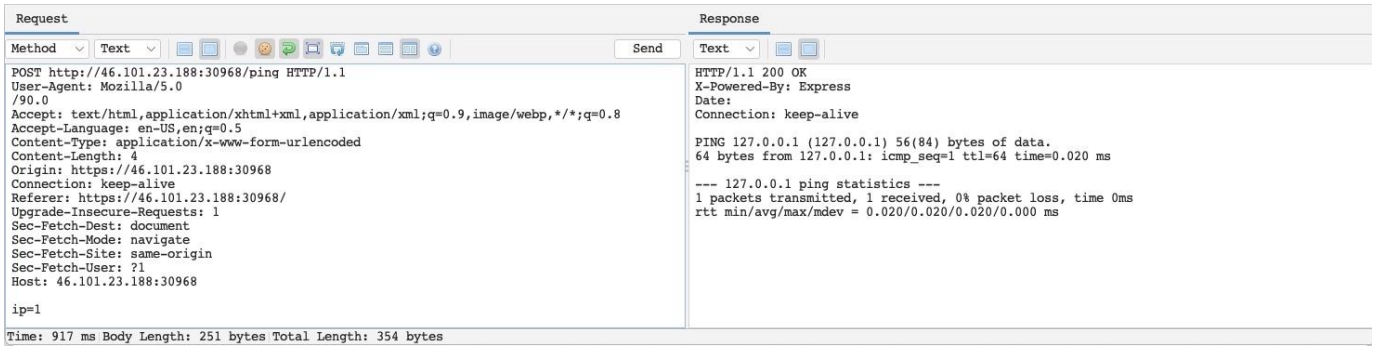


Tip

We can also right-click on the request and select Change Request Method to change the HTTP method between POST/GET without having to rewrite the entire request.

ZAP

In ZAP, once we locate our request, we can right-click on it and select Open/Resend with Request Editor, which would open the request editor window, and allow us to resend the request with the Send button to send our request:



We can also see the drop-down menu, allowing us to quickly switch the request method to any other HTTP method.

Encoding/Decoding

As we modify and send custom HTTP requests, we may have to perform various types of encoding and decoding to interact with the webserver properly. Both tools have built-in encoders that can help us in quickly encoding and decoding various types of text.

URL Encoding

It is essential to ensure that our request data is URL-encoded and our request headers are correctly set. Otherwise, we may get a server error in the response. This is why encoding and decoding data becomes essential as we modify and repeat web requests. Some of the key characters we need to encode are:

- : May indicate the end of request data if not encoded
- : Otherwise interpreted as a parameter delimiter
- : Otherwise interpreted as a fragment identifier

To URL-encode text in Burp Repeater, we can select that text and right-click on it, then select ( ), or by selecting the text and clicking . Burp also supports URL-encoding as we type if we right-click and enable that option, which will encode all of our text as we type it. On the other hand, ZAP should automatically URL-encode all of our request data in the background before sending the request, though we may not see that explicitly.

There are other types of URL-encoding, like Full URL-Encoding or Unicode URL encoding, which may also be helpful for requests with many special characters.

Decoding

While URL-encoding is key to HTTP requests, it is not the only type of encoding we will encounter. It is very common for web applications to encode their data, so we should be able to quickly decode that data to examine the original text. On the other hand, back-end servers may expect data to be encoded in a particular format or with a specific encoder, so we need to be able to quickly encode our data before we send it.

The following are some of the other types of encoders supported by both tools:

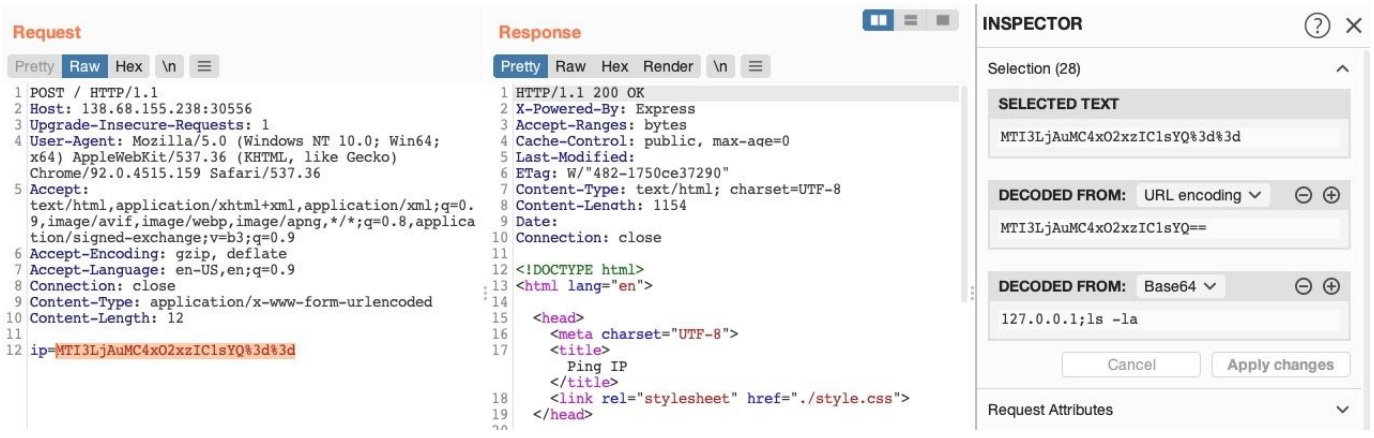
While URL-encoding is key to HTTP requests, it is not the only type of encoding we will encounter. It is very common for web applications to encode their data, so we should be able to quickly decode that data to examine the original text. On the other hand, back-end servers may expect data to be encoded in a particular format or with a specific encoder, so we need to be able to quickly encode our data before we send it.

The following are some of the other types of encoders supported by both tools:

- HTML
- Unicode
- Base64
- ASCII hex

For example, perhaps we came across the following cookie that is base64 encoded, and we need to decode it:

In recent versions of Burp, we can also use the tool to perform encoding and decoding (among other things), which can be found in various places like Burp Proxy or Burp Repeater:



Encoding

As we can see, the text holds the value . So, if we were performing a penetration test on a web application and find that the cookie holds this value, we may want to test modifying it to see whether it changes our user privileges. So, we can copy the above value, change to and to , and try to encode it again using its original encoding method ( ):

Decode the flag

Tip

base64 and URL

Intro to Proxychains

Proxying Tools

An important aspect of using web proxies is enabling the interception of web requests made by command-line tools and thick client applications. This gives us transparency into the web requests made by these applications and allows us to utilize all of the different proxy features we have used with web applications.

To route all web requests made by a specific tool through our web proxy tools, we have to set them up as the tool's proxy (i.e.), similarly to what we did with our browsers. Each tool may have a different method for setting its proxy, so we may have to investigate how to do so for each one.

Note

Proxying tools usually slows them down, therefore, only proxy tools when you need to investigate their requests, and not for normal usage

- Tor and proxychains

Note

Use tor proxy with a vpn If your choise in NordVpn, the form the terminal use with nordvpn login.

PROXYCHAINS FEATURES

- Support SOCKS5, SOCKS4, and HTTP CONNECT proxy servers.
- Proxychains can be mixed up with a different proxy types in a list
- Proxychains also supports any kinds of chaining option methods, like: random, which takes a random proxy in the list stored in a configuration file, or chaining proxies in the exact order list, different proxies are separated by a new line in a file. There is also a dynamic option, that lets Proxychains go through the live only proxies, it will exclude the dead or unreachable proxies, the dynamic option often called smart option.
- Proxychains can be used with servers, like squid, sendmail, etc.
- Proxychains is capable to do DNS resolving through proxy.
- Proxychains can handle any TCP client application, ie., nmap, telnet.

PROXYCHAINS SYNTAX

Add command "proxychains" for every job, that means we enable Proxychains service. For example, we want to scan available hosts and its ports in our network using Nmap using Proxychains the command should look like this:

NMAP

As we can see, we can use the flag. We should also add the flag to skip host discovery (as recommended on the man page). Finally, we'll also use the flag to examine what an nmap script scan does:

Note

Nmap's built-in proxy is still in its experimental phase, as mentioned by its manual (man nmap), so not all functions or traffic may be routed through the proxy. In these cases, we can simply resort to proxychains, as we did earlier.

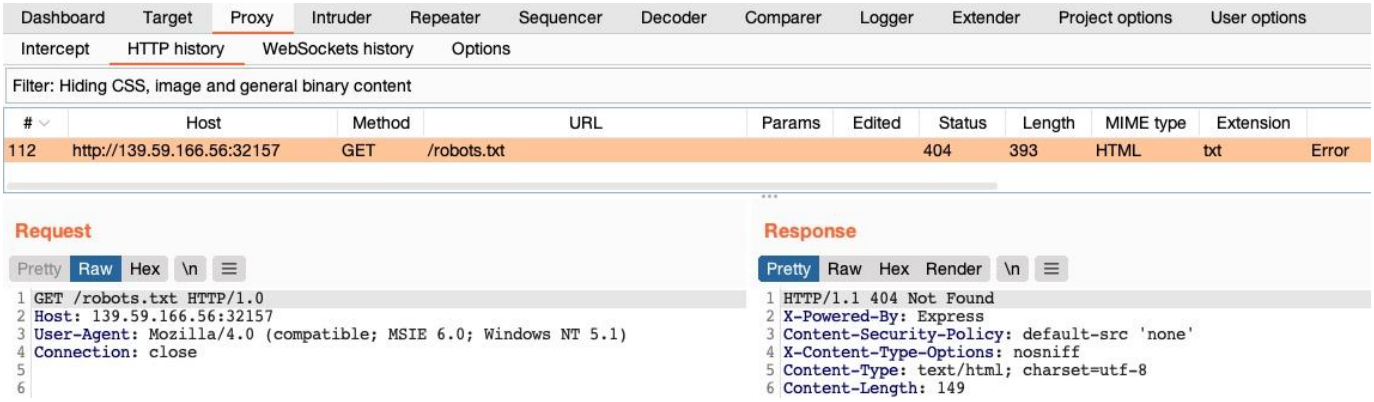
Proxychains configuration file located on

By default proxychains directly sends the traffic first through our host at 127.0.0.1 on port 9050 (the default Tor configuration). If you are using Tor, leave this as it is. If you are not using Tor, you will need to comment out this line.

We should also enable to reduce noise by un-commenting . Once that's done, we can prepend to any command, and the traffic of that command should be routed through (i.e., our web proxy). For example, let's try using cURL on one of our previous exercises:

METASPLOIT

Finally, let's try to proxy web traffic made by Metasploit modules to better investigate and debug them. We should begin by starting Metasploit with . Then, to set a for any exploit within Metasploit, we can use the set flag. Let's try the robots_txt scanner as an example and run it against one of our previous exercises:



We can similarly use our web proxies with other tools and applications, including scripts and thick clients. All we have to do is set the proxy of each tool to use our web proxy. This allows us to examine exactly what these tools are sending and receiving and potentially repeat and modify their requests while performing web application penetration testing.

Burp Intruder

Both Burp and ZAP provide additional features other than the default web proxy, which are essential for web application penetration testing. Two of the most important extra features are and . The built-in web fuzzers are powerful tools that act as web fuzzing, enumeration, and brute-forcing tools. This may also act as an alternative for many of the CLI-based fuzzers we use, like , , , , among others.

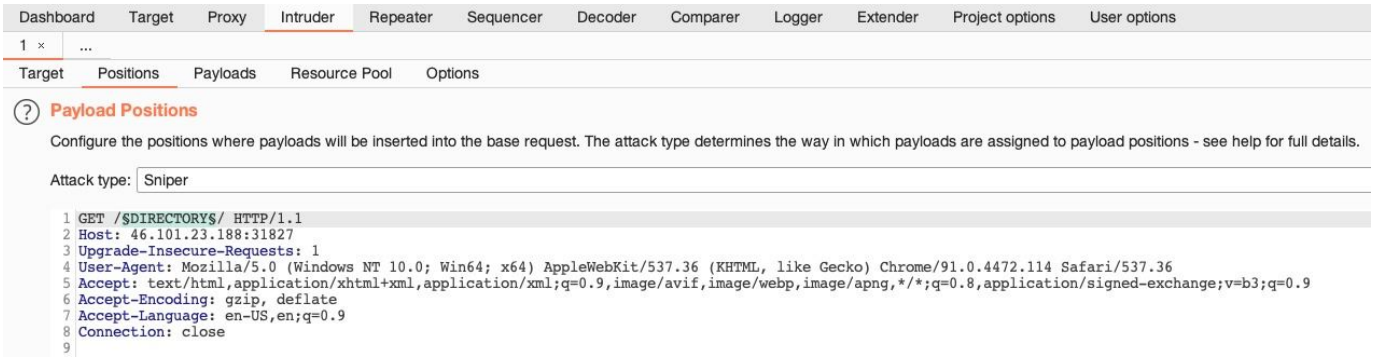
Burp's web fuzzer is called , and can be used to fuzz pages, directories, sub-domains, parameters, parameters values, and many other things. Though it is much more advanced than most CLI-based web fuzzing tools, the free version is throttled at a speed of 1 request per second, making it extremely slow compared to CLI-based web fuzzing tools, which can usually read up to 10k requests per second. This is why we would only use the free version of Burp Intruder for short queries. The version has unlimited speed, which can rival common web fuzzing tools, in addition to the very useful features of Burp Intruder. This makes it one of the best web fuzzing and brute-forcing tools.

We can then go to Intruder by clicking on its tab or with the shortcut [CTRL+SHIFT+I], which takes us right to Burp Intruder:

POSITIONS

The tab, 'Positions', is where we place the payload position pointer, which is the point where words from our wordlist will be placed and iterated over. We will be demonstrating how to fuzz web directories, which is similar to what's done by tools like ffuf or gobuster.

To check whether a web directory exists, our fuzzing should be in 'GET /DIRECTORY/', such that existing pages would return 200 OK, otherwise we'd get 404 NOT FOUND. So, we will need to select DIRECTORY as the payload position, by either wrapping it with § or by selecting the word DIRECTORY and clicking on the the Add § button:



Tip

the DIRECTORY in this case is the pointer's name, which can be anything, and can be used to refer to each pointer, in case we are using more than position with different wordlists for each.

The final thing to select in the target tab is the Attack Type. The attack type defines how many payload pointers are used and determines which payload is assigned to which position. For simplicity, we'll stick to the first type, Sniper, which uses only one position. Try clicking on the ? at the top of the window to read more about attack types, or check out this link.

> **Note**
>
> Be sure to leave the extra two lines at the end of the request, otherwise we may get an error response from the server.

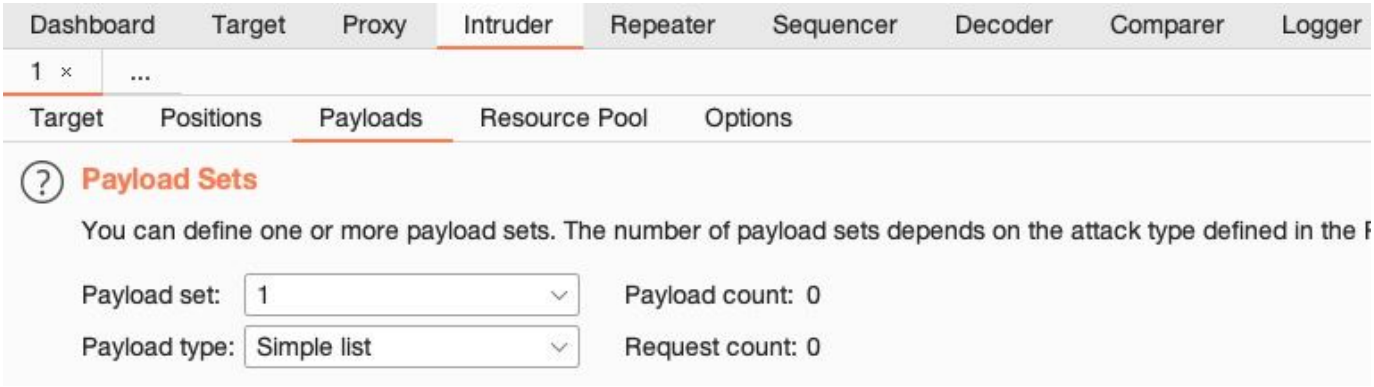
PAYLOADS

The tab, we get to choose and customize our payloads/wordlists. This payload/wordlist is what would be iterated over, and each element/line of it would be placed and tested one by one in the Payload Position we chose earlier. There are four main things we need to configure:

- Payload Sets
- Payload Options
- Payload Processing
- Payload Encoding

Payload Sets

The first thing we must configure is the . The payload set identifies the Payload number, depending on the attack type and number of Payloads we used in the Payload Position Pointers:



In this case, we only have one Payload Set, as we chose the Cluster Bomb` attack type, for example, and added several payload positions, we would get more payload sets to choose from and choose different options for each. In our case, we'll select 1 for the payload set.

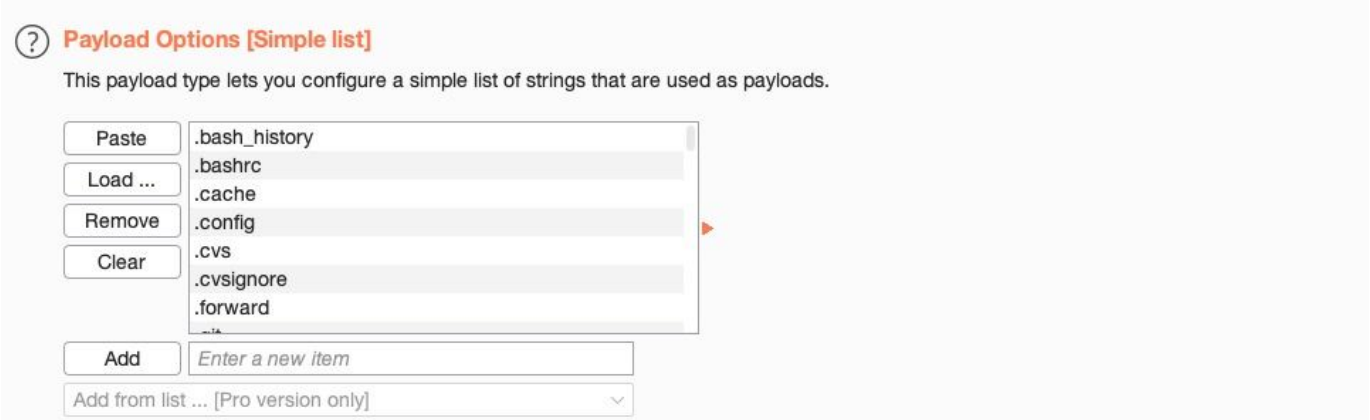
Next, we need to select the , which is the type of payloads/wordlists we will be using. Burp provides a variety of Payload Types, each of which acts in a certain way. For example:

- Simple List: The basic and most fundamental type. We provide a wordlist, and Intruder iterates over each line in it.
- Runtime file: Similar to Simple List, but loads line-by-line as the scan runs to avoid excessive memory usage by Burp.
- Character Substitution: Lets us specify a list of characters and their replacements, and Burp Intruder tries all potential permutations.

Payload Options

Next, we must specify the Payload Options, which is different for each we select in . For a , we have to create or load a wordlist. To do so, we can input each item manually by clicking , which would build our wordlist on the fly. The other more common option is to click on , and then select a file to load into Burp Intruder.

We will select as our wordlist. We can see that Burp Intruder loads all lines of our wordlist into the Payload Options table:



We can add another wordlist or manually add a few items, and they would be appended to the same list of items. We can use this to combine multiple wordlists or create customized wordlists. In Burp Pro, we also can select from a list of existing wordlists contained within Burp by choosing from the Add from list menu option.

Tip

In case you wanted to use a very large wordlist, it's best to use Runtime file as the Payload Type instead of Simple List, so that Burp Intruder won't have to load the entire wordlist in advance, which may throttle memory usage.

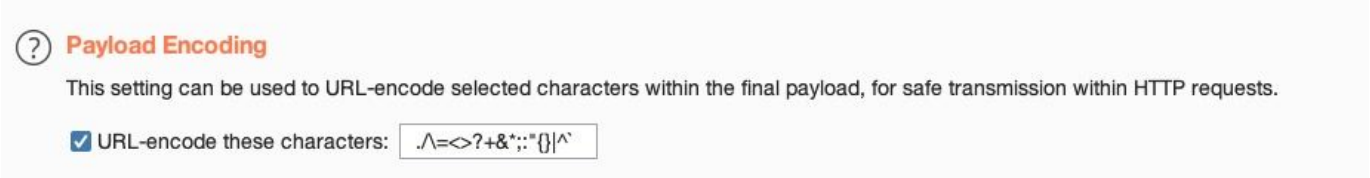
Payload Processing

Another option we can apply is , which allows us to determine fuzzing rules over the loaded wordlist. For example, if we wanted to add an extension after our payload item, or if we wanted to filter the wordlist based on specific criteria, we can do so with payload processing.

Let's try adding a rule that skips any lines that start with a . (as shown in the wordlist screenshot earlier). We can do that by clicking on the button and then selecting , which allows us to provide a regex pattern for items we want to skip. Then, we can provide a regex pattern that matches lines starting with ., which is: :

Payload Encoding

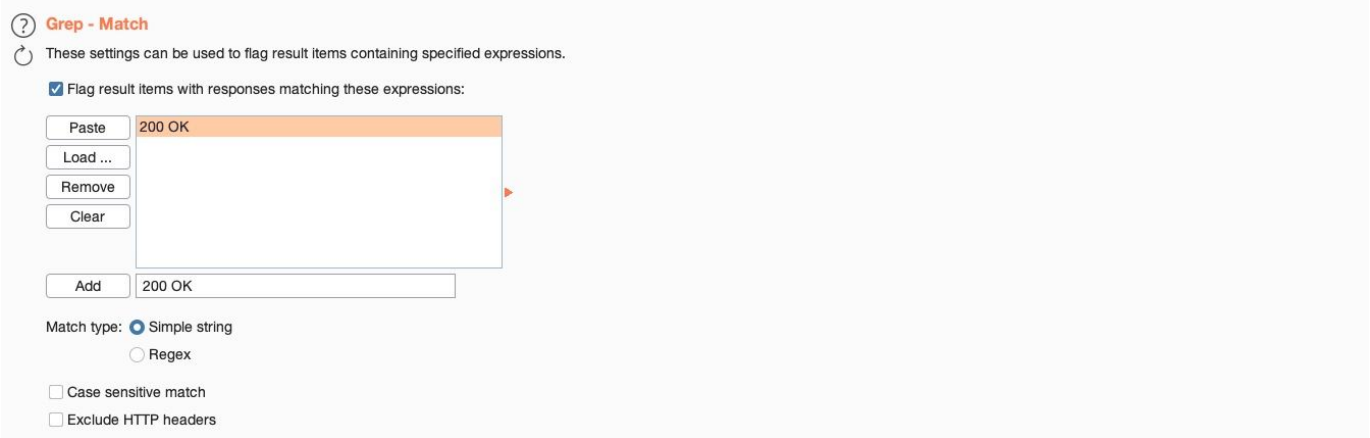
The fourth and final option we can apply is Payload Encoding, enabling us to enable or disable Payload URL-encoding.



OPTIONS

Finally, we can customize our attack options from the tab. There are many options we can customize (or leave at default) for our attack. For example, we can set the number of and to 0.

Another useful option is the - Match, which enables us to flag specific requests depending on their responses. As we are fuzzing web directories, we are only interested in responses with HTTP code . So, we'll first enable it and then click Clear to clear the current list. After that, we can type 200 OK to match any requests with this string and click Add to add the new rule. Finally, we'll also disable , as what we are looking for is in the HTTP header:



ATTACK

Now that everything is properly set up, we can click on the button and wait for our attack to finish. Once again, in the free Community Version, these attacks would be very slow and take a considerable amount of time for longer wordlists.

ZAP Fuzzer

ZAP's Fuzzer is called (ZAP Fuzzer). It can be very powerful for fuzzing various web end-points, though it is missing some of the features provided by Burp Intruder. ZAP Fuzzer, however, does not throttle the fuzzing speed, which makes it much more useful than Burp's free Intruder.

Fuzz

To start our fuzzing, we will visit the URL from the exercise at the end of this section to capture a sample request. As we will be fuzzing for directories, let's visit to place our fuzzing location on later on. Once we locate our request in the proxy history, we will right-click on it and select ( ), which will open the window:

The main options we need to configure for our Fuzzer attack are:

- Fuzz Location
- Payloads
- Processors
- Options

Burp Scanner

An essential feature of web proxy tools is their web scanners. Burp Suite comes with , a powerful scanner for various types of web vulnerabilities, using a for building the website structure, and for passive and active scanning.

Burp Scanner is a Pro-Only feature, and it is not available in the free Community version of Burp Suite. However, given the wide scope that Burp Scanner covers and the advanced features it includes, it makes it an enterprise-level tool, and as such, it is expected to be a paid feature.

ZAP SCANNER

ZAP also comes bundled with a Web Scanner similar to Burp Scanner. ZAP Scanner is capable of building site maps using ZAP Spider and performing both passive and active scans to look for various types of vulnerabilities.

Spider

Let’s start with ZAP Spider, which is similar to the Crawler feature in Burp. To start a Spider scan on any website, we can locate a request from our History tab and select (Attack>Spider) from the right-click menu.

Another option is to use the HUD in the pre-configured browser. Once we visit the page or website we want to start our Spider scan on, we can click on the second button on the right pane (Spider Start), which would prompt us to start the scan.

> **Note**
>
> When we click on the Spider button, ZAP may tell us that the current website is not in our scope, and will ask us to automatically add it to the scope before starting the scan, to which we can say ‘Yes’. The Scope is the set of URLs ZAP will test if we start a generic scan, and it can be customized by us to scan multiple websites and URLs. Try to add multiple targets to the scope to see how the scan would run differently.

8.5 HTTP Status Codes

HTTP server responds, the first line always contains a status code informing the client of the outcome of their request and also potentially how to handle it. These status codes can be broken down into 5 different ranges:

Code	Descritpion
100-199 - Information Response	These are sent to tell the client the first part of their request has been accepted and they should continue sending the rest of their request. These codes are no longer very common.
200-299 - Success	This range of status codes is used to tell the client their request was successful.
300-399 - Redirection	These are used to redirect the client's request to another resource. This can be either to a different webpage or a different website altogether.
400-499 - Client Errors	Used to inform the client that there was an error with their request.
500-599 - Server Errors	This is reserved for errors happening on the server-side and usually indicate quite a major problem with the server handling the request.

Common HTTP Status Codes:

Code	Descritpion
200 - OK	The request was completed successfully.
201 - Created	A resource has been created (for example a new user or new blog post).
301 - Permanent Redirect	This redirects the client's browser to a new webpage or tells search engines that the page has moved somewhere else and to look there instead.
302 - Temporary Redirect	Similar to the above permanent redirect, but as the name suggests, this is only a temporary change and it may change again in the near future.
400 - Bad Request	This tells the browser that something was either wrong or missing in their request. This could sometimes be used if the web server resource that is being requested expected a certain parameter that the client didn't send.
401 - Not Authorised	You are not currently allowed to view this resource until you have authorised with the web application, most commonly with a username and password.
403 - Forbidden	You do not have permission to view this resource whether you are logged in or not.
405 - Method Not Allowed	The resource does not allow this method request, for example, you send a GET request to the resource /create-account when it was expecting a POST request instead.
404 - Page Not Found	The page/resource you requested does not exist.
500 - Internal Service Error	The server has encountered some kind of error with your request that it doesn't know how to handle properly.
503 - Service Unavailable	This server cannot handle your request as it's either overloaded or down for maintenance.