Pentesting Prevention Proposal Report

Executive Summary

This comprehensive security assessment consolidates findings from three penetration testing exercises conducted on target infrastructure comprising a vulnerable Linux machine and a web application. The assessment identified 9 significant security vulnerabilities across both systems, including critical issues such as SQL injection, privileged SUID binary exploitation, and insecure file upload functionality.

The most severe findings allow for complete system compromise through privilege escalation on the machine and remote code execution on the web server. If exploited by malicious actors, these vulnerabilities could lead to unauthorized access to sensitive data, service disruption, and potential lateral movement within the network.

This report details specific technical recommendations to address these vulnerabilities through both immediate tactical mitigations and long-term strategic prevention measures. Key recommendations include:

1. Immediate implementation of SSH hardening measures and SUID binary remediation to prevent privilege escalation
2. Critical web application fixes for SQL injection and file upload vulnerabilities
3. Strategic development of secure coding practices and input validation frameworks
4. Implementation of network segmentation and proper access controls

By implementing these recommendations, the organization will significantly enhance its security posture and develop resilience against common attack vectors. The proposed measures follow security best practices from OWASP, CIS, and NIST frameworks, providing a balanced approach to security improvement with consideration for implementation complexity and business impact.

Table of Contents

Introduction

This report consolidates findings and recommendations from three penetration testing exercises conducted between January and March 2025. The objective was to identify security vulnerabilities, demonstrate exploitation techniques, and propose comprehensive prevention and mitigation strategies for both immediate remediation and long-term security improvement.

Project Background

The penetration testing exercises were conducted on two primary targets:

1. **Linux Machine**: A server running multiple network services including SSH, HTTP, and databases
2. **Web Application**: A custom-developed content management system with user authentication and file management capabilities

The assessment was performed using an ethical hacking methodology following the PTES (Penetration Testing Execution Standard) framework, with explicit permission from system owners and within the defined scope of work.

Assessment Objectives

- Identify exploitable vulnerabilities in both the server infrastructure and web application
- Document exploitation paths to demonstrate real-world attack scenarios
- Analyze root causes of discovered vulnerabilities
- Provide prioritized, actionable recommendations for remediation
- Propose strategic security improvements to prevent similar vulnerabilities in the future

Regulatory Context

This assessment supports the organization's compliance efforts with relevant security frameworks and regulations including:

- PCI DSS Requirement 11.3 (Penetration Testing)
- NIST Cybersecurity Framework (Identify, Protect, Detect)
- SOC 2 Type II audit requirements

Scope of Assessment

The assessment included the following systems and components:

- Linux server (192.168.1.100) - All network services and local configurations
- Web application (https://webapp.example.com) - All public-facing functionality
- Authentication mechanisms for both systems
- File upload and management functionality
- Database interactions and data handling

The scope explicitly excluded:

- Denial of service testing
- Physical security assessment
- Social engineering attacks
- Testing of third-party hosted services

Approach and Strategy

The penetration testing followed a structured methodology with distinct approaches for the machine and website targets:

Machine Testing Strategy

- Reconnaissance using passive and active information gathering techniques
- Network scanning to identify open ports and running services
- Vulnerability scanning to detect potential weaknesses
- Exploitation of identified vulnerabilities to gain initial access
- Privilege escalation attempts to obtain higher-level permissions
- Post-exploitation activities to determine the extent of potential compromise

Website Testing Strategy

- Web application reconnaissance and mapping
- Authentication and session management testing
- Input validation and injection testing
- Cross-site scripting (XSS) vulnerability assessment
- SQL injection testing
- Access control and authorization testing
- Business logic flaw identification

Phases of Pentesting

Phase 1: Reconnaissance

- **Machine**: Used Nmap for port scanning and service enumeration. Discovered open ports 22 (SSH), 80 (HTTP), 445 (SMB), and 3306 (MySQL).

- **Website**: Utilized dirb and gobuster for directory enumeration. Employed Burp Suite for request interception and analysis. Discovered admin portal, login pages, and file upload functionality.

Phase 2: Vulnerability Identification

- **Machine**: Identified outdated service versions, weak SSH configuration, and misconfigured file permissions.
- **Website**: Discovered XSS vulnerabilities, SQL injection points, and insecure file upload functionality.

Phase 3: Exploitation

- **Machine**: Exploited weak credentials to gain SSH access, used misconfigured SUID binaries for privilege escalation.
- **Website**: Performed SQL injection to bypass authentication, executed stored XSS attacks, and uploaded malicious files through the vulnerable file upload feature.

Detected Vulnerabilities

This section catalogs all vulnerabilities discovered during the reconnaissance and exploitation phases of both the machine and website penetration tests. Each vulnerability is categorized by target system, with details on severity, exploitation status, and potential impact.

Machine Vulnerabilities (Report v1 and v2)

1. **Weak SSH Configuration (CVE-2021-28041)**
    - **Description**: SSH server configured to allow password authentication with weak complexity requirements and no account lockout policy
    - **Severity**: High
    - **Exploitation Method**: Used Hydra to brute force SSH credentials of user "admin" with a common password list
    - **Impact**: Full remote access to the target machine with user-level privileges
    - **Evidence**:
        - Successful login via SSH using discovered credentials
        - Shell access with user privileges confirmed
        - Access to user files and ability to execute commands
2. **Privileged SUID Binary (Custom Application)**
    - **Description**: Custom maintenance script installed with SUID bit set had improper input validation and unsafe system calls
    - **Severity**: Critical

- **Exploitation Method**: Created malicious input that triggered command injection in the maintenance utility, allowing privilege escalation
- **Impact**: Full root access to the system, complete compromise of the machine
- **Evidence**:
  - Successful privilege escalation to root user
  - Full system access, ability to modify any system file
  - Ability to create persistent backdoors

3. **Outdated Apache Server (2.4.41 with CVE-2021-44790)**

   - **Description**: Web server running outdated version with known buffer overflow vulnerability
   - **Severity**: Medium
   - **Exploitation Method**: Used publicly available exploit to trigger buffer overflow condition
   - **Impact**: Remote code execution with web server privileges
   - **Evidence**:
     - Successful deployment of reverse shell
     - Access to web server configuration files
     - Ability to modify web content
4. **Unprotected Internal Service (Redis on port 6379)**

   - **Description**: Redis database accessible without authentication from external networks
   - **Severity**: High
   - **Exploitation Method**: Direct connection to Redis service and extraction of stored data
   - **Impact**: Unauthorized access to potentially sensitive database information
   - **Evidence**:
     - Successful connection to Redis without credentials
     - Extraction of database contents including user information
     - Ability to modify database entries

Website Vulnerabilities (Report v1 and v2)

1. **SQL Injection in Login Form**

   - **Description**: Website login page vulnerable to SQL injection due to improper input sanitization
   - **Severity**: Critical
   - **Exploitation Method**: Used payload `' OR 1=1 --` to bypass authentication
   - **Impact**: Unauthorized access to admin portal, potential access to all user accounts
   - **Evidence**:
     - Successful authentication bypass

        - Access to administrative functionality
        - Ability to view all user data
2. **Persistent Cross-Site Scripting (XSS)**

    - **Description**: Comment section on blog posts did not sanitize user input, allowing JavaScript injection
    - **Severity**: Medium
    - **Exploitation Method**: Injected `<script>` tag in comment field that executed when other users viewed the page
    - **Impact**: Potential for session hijacking, credential theft, and defacement
    - **Evidence**:
        - Successfully stored and executed JavaScript code
        - Pop-up alerts displayed to other users viewing the page
        - Ability to access document.cookie
3. **Insecure File Upload Functionality**

    - **Description**: File upload feature in the admin portal lacked proper file type validation
    - **Severity**: High
    - **Exploitation Method**: Uploaded PHP webshell disguised as an image file by modifying Content-Type header
    - **Impact**: Remote code execution on the web server
    - **Evidence**:
        - Successfully uploaded malicious file
        - Confirmed execution of commands through webshell
        - Full access to web server file system
4. **Broken Access Control**

    - **Description**: User profile pages accessible by manipulating user ID parameter in URL
    - **Severity**: High
    - **Exploitation Method**: Changed user ID in URL from authorized profile to access unauthorized profiles
    - **Impact**: Unauthorized access to other users' personal information
    - **Evidence**:
        - Successfully accessed profiles of other users
        - Retrieved personal information including email addresses
        - Modified profile information of other users
5. **Cross-Site Request Forgery (CSRF)**

    - **Description**: No CSRF tokens implemented in form submissions
    - **Severity**: Medium
    - **Exploitation Method**: Created malicious page that automatically submitted form to change account details

- **Impact**: Forced actions on behalf of authenticated users
- **Evidence**:
  - Successfully executed unauthorized actions
  - Changed user passwords without knowledge
  - Modified account details without proper authorization

Prevention Proposal

This section outlines comprehensive strategies to prevent the introduction of similar vulnerabilities in future development and deployment. These preventive measures are designed to address the root causes of the vulnerabilities identified in our penetration testing exercises.

Secure Development Practices

1. **Input Validation and Sanitization Framework**

   - **Specific Target**: SQL Injection, XSS vulnerabilities
   - **Implementation**: Develop a centralized input validation library that all applications must use
   - **Technical Details**:
     - Implement context-aware output encoding for different parts of HTML documents
     - Create validated input types for common data (email, phone, etc.)
     - Use parameterized queries for all database operations with prepared statements

Example code snippet for parameterized queries:
 $stmt = $db->prepare("SELECT * FROM users WHERE username = ? AND password = ?");$stmt->bind_param("ss", $username, $password_hash);$stmt->execute();

     -
2. **Secure File Upload Controls**

   - **Specific Target**: Insecure file upload vulnerability
   - **Implementation**: Create a secure file handling service
   - **Technical Details**:
     - Validate file content rather than relying on extensions or MIME types
     - Use file type verification libraries to confirm actual file types
     - Store uploaded files outside the web root
     - Rename files with random names to prevent path traversal
     - Implement content disposition headers to prevent execution of downloaded files
3. **Secure Authentication Framework**

- **Specific Target**: Weak SSH configuration, password vulnerabilities
- **Implementation**: Centralized authentication service with strong controls
- **Technical Details**:
  - Implement password strength meters during registration
  - Use bcrypt or Argon2 for password hashing with appropriate work factors
  - Enforce multi-factor authentication for administrative access
  - Implement secure password recovery mechanisms

Code Review Process

1. **Security-Focused Code Review Checklist**

   - **Specific Target**: All discovered vulnerabilities
   - **Implementation**: Formalized security review process
   - **Technical Details**:
     - Create language-specific security checklists for reviewers
     - Require at least one security-focused reviewer for each pull request
     - Implement automated code linting with security rules
     - Document specific patterns to avoid (e.g., unsafe system calls, dynamic SQL)
2. **Automated Security Testing**

   - **Specific Target**: All vulnerability types
   - **Implementation**: Multi-layered security testing in CI/CD pipeline
   - **Technical Details**:
     - Integrate SAST tools like SonarQube, Checkmarx, or Fortify
     - Implement DAST scanning with tools like OWASP ZAP or Burp Suite
     - Deploy Interactive Application Security Testing (IAST) for runtime analysis
     - Create security unit tests for previous vulnerabilities to prevent regression
3. **Security Knowledge Base**

   - **Specific Target**: All vulnerabilities
   - **Implementation**: Internal repository of vulnerabilities and fixes
   - **Technical Details**:
     - Document each discovered vulnerability with root cause analysis
     - Create secure code examples showing both vulnerable and fixed code
     - Develop security training modules based on real findings
     - Implement regular security training sessions for developers

Security Policies

1. **Service Exposure Policy**

   - **Specific Target**: Unprotected Redis service vulnerability

    - **Implementation**: Formal process for exposing internal services
    - **Technical Details**:
        - Require security review and approval before exposing any internal service
        - Implement network access control lists for all services
        - Configure proper authentication for all exposed services
        - Regular scanning for unauthorized open ports or services
2. **Access Control Framework**

    - **Specific Target**: Broken access control vulnerability
    - **Implementation**: Centralized authorization service
    - **Technical Details**:
        - Implement role-based access control (RBAC) model
        - Develop API gateway with consistent authorization checks
        - Enforce access checks at both API and UI levels
        - Implement proper session management with secure cookies
        - Use security headers like Content-Security-Policy
3. **CSRF Protection Policy**

    - **Specific Target**: CSRF vulnerability
    - **Implementation**: Framework-level CSRF protection
    - **Technical Details**:
        - Generate and validate anti-CSRF tokens for all state-changing operations
        - Implement SameSite cookie attribute
        - Use custom request headers for AJAX requests
        - Verify origin and referer headers for sensitive operations

Mitigation Proposal

This section details the specific tactical measures recommended to mitigate the identified vulnerabilities in the current system. These mitigation strategies are designed to be implemented immediately to address existing security risks.

SSH and Authentication Security

1. **SSH Hardening**

    - **Target Vulnerability**: Weak SSH Configuration
    - **Implementation Priority**: High
    - **Technical Details**:
        - Disable password authentication and implement key-based authentication

Update `/etc/ssh/sshd_config` with the following changes:
PasswordAuthentication noPermitRootLogin noAllowUsers [specific_users]Protocol 2MaxAuthTries 3PubkeyAuthentication yes

        -
        - Implement SSH key rotation policy

Configure fail2ban for SSH with thresholds:
bantime = 3600findtime = 600maxretry = 3

        -

2. **Multi-Factor Authentication Implementation**

    - **Target Vulnerability**: Password-only authentication
    - **Implementation Priority**: Medium
    - **Technical Details**:
        - Deploy Google Authenticator or YubiKey for SSH and admin access
        - Install required packages: `libpam-google-authenticator`

Configure PAM to require MFA by updating `/etc/pam.d/sshd`:
auth required pam_google_authenticator.so

        -
        - Generate MFA seeds for all administrative users

Service and System Hardening

1. **SUID Binary Review and Remediation**

    - **Target Vulnerability**: Privileged SUID Binary
    - **Implementation Priority**: Critical
    - **Technical Details**:
        - Immediate audit of all SUID/SGID binaries: `find / -type f -perm -4000 -o -perm -2000`
        - For the vulnerable maintenance script:
            - Remove SUID bit: `chmod u-s /path/to/vulnerable_script`
            - Implement proper input validation in the code
            - Consider replacing with safer sudo rules for specific commands
        - Create regular automated SUID/SGID audit task
2. **Service Version Updates**

    - **Target Vulnerability**: Outdated Apache Server
    - **Implementation Priority**: High
    - **Technical Details**:

Immediately update Apache to latest version:
apt updateapt upgrade apache2

-
- Implement specific security patches:
  - CVE-2021-44790: Apply patch or update to Apache 2.4.52+

Configure automatic security updates:
apt install unattended-upgradesdpkg-reconfigure unattended-upgrades

-

3. **Redis Security Implementation**

   - **Target Vulnerability**: Unprotected Redis Service
   - **Implementation Priority**: High
   - **Technical Details**:

Bind Redis to localhost only by updating `/etc/redis/redis.conf`:
bind 127.0.0.1

-

Enable Redis authentication:
requirepass StrongRandomPassword

-

Implement proper ACLs in Redis 6.0+:
acl setuser app on >appPassword ~app:* +get +set

-

Disable dangerous commands:
rename-command FLUSHALL ""rename-command CONFIG ""

-

Web Application Security

1. **SQL Injection Remediation**

   - **Target Vulnerability**: SQL Injection in Login Form
   - **Implementation Priority**: Critical
   - **Technical Details**:

Replace vulnerable code with parameterized queries:
// Replace this:$query = "SELECT * FROM users WHERE username='$username' AND password='$password'";// With this:$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ? AND password = ?");$stmt->execute([$username, $password_hash]);

        -
        - Implement input validation for all user-supplied data
        - Consider implementing an ORM like Doctrine or Eloquent
2. **XSS Protection Implementation**

    - **Target Vulnerability**: Persistent XSS in Comment Section
    - **Implementation Priority**: Medium
    - **Technical Details**:

Implement contextual output encoding:
 // Replace direct output:echo $comment;// With encoded output:echo htmlspecialchars($comment, ENT_QUOTES, 'UTF-8');

        -

Add Content-Security-Policy header:
 Content-Security-Policy: default-src 'self'; script-src 'self'

        -
        - Sanitize input using HTML Purifier or similar library
        - Install XSS filters at the application level
3. **Secure File Upload Implementation**

    - **Target Vulnerability**: Insecure File Upload
    - **Implementation Priority**: High
    - **Technical Details**:

Validate file content using functions like `finfo_file()`:
 $finfo = new finfo(FILEINFO_MIME_TYPE);$fileContentsType = $finfo->file($_FILES['upload']['tmp_name']);if (!in_array($fileContentsType, $allowedMimeTypes)) {   die("Invalid file type");}

        -
        - Store uploaded files outside web root
        - Generate random filenames for uploaded files
        - Set proper file permissions: `chmod 0644 $uploaded_file`
        - Configure web server to not execute files in upload directories

Network Security

1. **Network Segmentation Implementation**

    - **Target Vulnerability**: Flat network topology
    - **Implementation Priority**: Medium
    - **Technical Details**:

        - Create separate VLANs for different server types:
            - Web servers: VLAN 10
            - Database servers: VLAN 20
            - Administrative access: VLAN 30

Configure firewall rules to restrict traffic between segments:
 # Allow web servers to access DB servers on MySQL port onlyiptables -A FORWARD -s 10.0.10.0/24 -d 10.0.20.0/24 -p tcp --dport 3306 -j ACCEPT# Default deny between segmentsiptables -A FORWARD -j DROP

        -
        - Implement 802.1X authentication for network access
2. **Web Application Firewall Deployment**

    - **Target Vulnerability**: Multiple web vulnerabilities
    - **Implementation Priority**: Medium
    - **Technical Details**:

Install and configure ModSecurity with OWASP Core Rule Set:
 apt install libapache2-mod-security2cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf

        -
        - Enable specific rules for detected vulnerabilities:
            - SQL Injection: Enable CRS rules 942*
            - XSS: Enable CRS rules 941*
            - File upload: Enable CRS rules 930*
        - Set ModSecurity to blocking mode after testing period

Mitigation Analysis

Effectiveness Assessment

Mitigation Measure	Vulnerabilities Addressed	Implementation Complexity	Effectiveness	Time to Implement
Patch Management	Outdated Service Versions	Medium	High	2-4 weeks
SSH Hardening	Weak SSH Configuration	Low	High	1 week
SUID Binary Review	Privileged SUID Binary	Medium	High	2 weeks

Input Validation	SQL Injection, XSS	Medium	High	4-6 weeks
File Upload Restrictions	Insecure File Upload	Low	High	1-2 weeks
Network Segmentation	Multiple	High	Medium	2-3 months

Cost-Benefit Analysis

The proposed mitigation measures have been prioritized based on:

1. Severity of vulnerabilities addressed
2. Implementation complexity and cost
3. Effectiveness in preventing exploitation

High-impact, low-cost measures such as SSH hardening and file upload restrictions should be implemented immediately, while more complex measures like network segmentation can be planned for phased implementation.

Potential Impact

Security Posture Improvement

The proposed prevention and mitigation measures will significantly enhance the overall security posture by:

- Reducing the attack surface through proper configuration and network segmentation
- Eliminating known vulnerabilities through effective patch management
- Preventing common web vulnerabilities through secure development practices
- Limiting the impact of potential breaches through proper access controls

Business Continuity

Implementing these measures will improve business continuity by:

- Reducing downtime due to security incidents
- Preventing data breaches and associated costs
- Maintaining customer trust and business reputation
- Ensuring compliance with security regulations and standards

Long-term Benefits

Beyond immediate vulnerability remediation, long-term benefits include:

- Establishment of a security-aware culture
- Development of repeatable security processes
- Creation of security metrics for continuous improvement
- Reduced total cost of ownership through proactive security

Conclusion

The penetration testing exercises revealed a series of critical security vulnerabilities that expose the organization to significant risk of unauthorized access, data breach, and system compromise. If exploited by malicious actors, these vulnerabilities could lead to:

1. Complete compromise of server infrastructure through privilege escalation
2. Unauthorized access to sensitive user and business data
3. Website defacement and manipulation of web content
4. Execution of malicious code on both server and client systems
5. Potential lateral movement to other systems on the network

The security issues identified stem from common root causes that can be addressed through systematic improvement:

1. **Lack of secure development practices** - especially input validation and output encoding
2. **Insufficient system hardening** - particularly regarding service configurations and privilege management
3. **Inadequate authentication controls** - across both machine and web application environments
4. **Missing network segmentation** - allowing excessive lateral movement opportunities

The proposed mitigation and prevention measures offer a balanced approach to addressing these issues with both immediate tactical fixes and strategic long-term improvements. We recommend a phased implementation approach:

Phase 1: Critical Remediation (0-30 days)

- Fix SQL injection vulnerabilities
- Remove/fix vulnerable SUID binary
- Implement SSH hardening
- Secure Redis service configuration
- Fix file upload validation

Phase 2: Enhanced Protection (31-90 days)

- Implement web application firewall

- Deploy network segmentation
- Implement MFA for administrative access
- Deploy XSS protection mechanisms
- Create patch management process

Phase 3: Strategic Improvement (91-180 days)

- Develop secure coding standards
- Implement security testing in development pipeline
- Create security training program
- Deploy automated vulnerability scanning
- Implement security metrics and reporting

Security is not a point-in-time effort but a continuous process requiring ongoing attention and adaptation to evolving threats. By implementing these recommendations and establishing a proactive security program, the organization will significantly reduce its exposure to common attack vectors and develop resilience against emerging threats.

We strongly recommend scheduling a follow-up penetration test within 6 months to validate the effectiveness of implemented security controls and identify any remaining vulnerabilities.

By prioritizing security as an integral part of both infrastructure management and software development processes, the organization will build a strong security foundation that supports business objectives while protecting critical assets and maintaining stakeholder trust.

References

1. OWASP Top 10 Web Application Security Risks: https://owasp.org/www-project-top-ten/
2. CIS Benchmarks: https://www.cisecurity.org/cis-benchmarks/
3. NIST Special Publication 800-53: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
4. NIST Cybersecurity Framework: https://www.nist.gov/cyberframework
5. SANS Critical Security Controls: https://www.sans.org/critical-security-controls/

Report prepared by: Theresa Baker

Date: April 15, 2025 at 4:41pm, Central Time