Penetration Testing Report on Security Shepherd

Product Name: Security Shepherd

Product Version: v3.0

Test Completion: 16/04/2025

Lead Penetration Tester: Sanad Masannat

Prepared for: Mark Scanlon

Consultant Information

Name: Sanad Masannat

Email: sanad.masannat@ucdconnect.ie

Location: University College Dublin, Belfield, Dublin 4

Manager: Mark Scanlon

Manager email: Mark.Scanlon@ucd.ie

NOTICE

This document contains confidential and proprietary information that is provided for the sole purpose of permitting the recipient to evaluate the recommendations submitted. In consideration of receipt of this document, the recipient agrees to maintain the enclosed information in confidence and not reproduce or otherwise disclose the information to any person outside the group directly responsible for evaluation of its contents.

Warning

Sensitive Information

This document contains confidential and sensitive information about the security posture of the OWASP Security Shepherd Application. This information should be classified. Only those individuals that have a valid need to know should be allowed access this document.

Index

1. Executive Summary

2. Scope

3. Test Cases

4. Findings with Cross Site Scripting

5. Findings with Cross Site Request Forgery

6. Findings with Broken Cryptography

7. Conclusions

Executive Summary

Lead Tester: Sanad Masannat

Number of Days testing: 20 days

Test Start date: 25/03/2025

Test End date: 17/04/2025

Project Information

Application Name: Security Shepherd

Application Version: v3.0

Release Date: October 24, 2015

Project Contact: Mark Scanlon

Findings

OWASP Top 10:

- A01:2021 - Broken Access Control
- A02:2021 - Cryptographic Failures
- A03:2021 - Injection

Total Defects:

Severity	**Number of Defects**
Critical	1
High	1
Medium	1
Low	0

What Defects were Found:

Vulnerability	**CVSS Score**
Cryptographic Failures	9.1
Cross Site Request Forgery	7.1
Cross Site Scripting	5.4

Scope

Roles Used:

- Admin (Administrator role)
- Test User 1 (basic role)

List of URLS:

- https://cwe.mitre.org/data/definitions/79.html
- https://cwe.mitre.org/data/definitions/327.html

- https://cwe.mitre.org/data/definitions/352.html

Levels and Challenges Attempted:

- Cross Site Scripting Challenge 4
- Cross Site Request Forgery Challenge 2
- Broken Cryptography Challenge 3

Testing Started: 26/03/2025

Testing Ended: 18/04/2025

Report Writing Started: 25/03/2025

Report Writing Ended: 17/04/2025

Test Cases

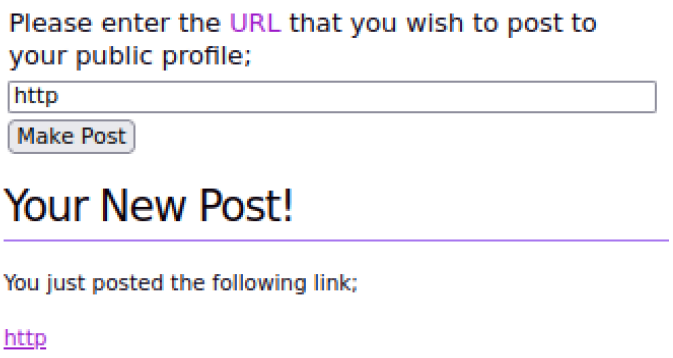
For XSS:

From OWASP testing Guide:

- (OTG-INPVAL-001) Testing for Reflected Cross Site Scripting

Scripts Used and output:

- http



- <script>alert(‘XSS)</script>

- <IMG SRC="#" ONERROR="alert('XSS')"/>

- <IFRAME SRC="javascript:alert('XSS');"></IFRAME>

- http" OnERRor=alert('XSS')

For CSRF:

From OWASP testing Guide:

- (OTG-SESS-005) Testing for Cross Site Request Forgery (CSRF)

Script Used:

- <form name="testingForm"
  action="https://192.168.1.200/user/csrfchallengetwo/plusplus"
  method="POST">
  <input type="hidden" name="userId" value="Our User ID" />
  <input type="submit"/>
  </form>
  <script> document. testingForm.submit(); </script>

For Broken Cryptography:

From OWASP testing Guide:

- (OTG-CRYPST-001) Testing for Weak SSL/TLS Ciphers, Insufficient

  Transport Layer Protection

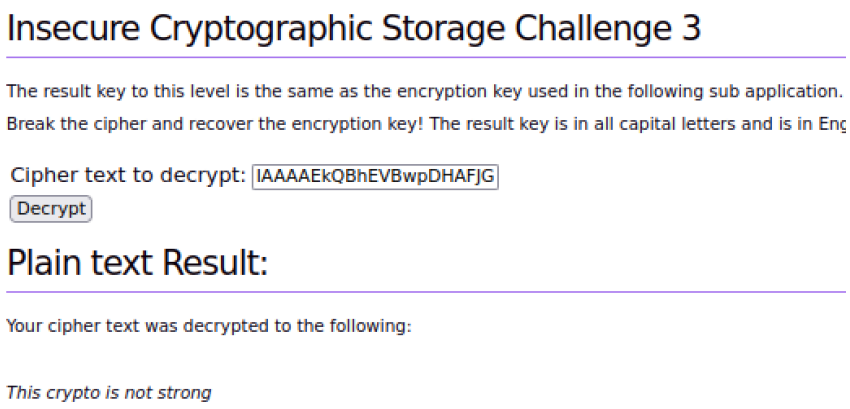
Python Code Used:

```
cipher_b64 = "IAAAAEkQBhEVBwpDHAFJGhYHSBYEGgocAw=="
plaintext = "This crypto is not strong".encode()
cipher_bytes = base64.b64decode(cipher_b64)
key = bytes([p ^ c for p, c in zip(plaintext, cipher_bytes)])
print("Recovered key:", key)
```

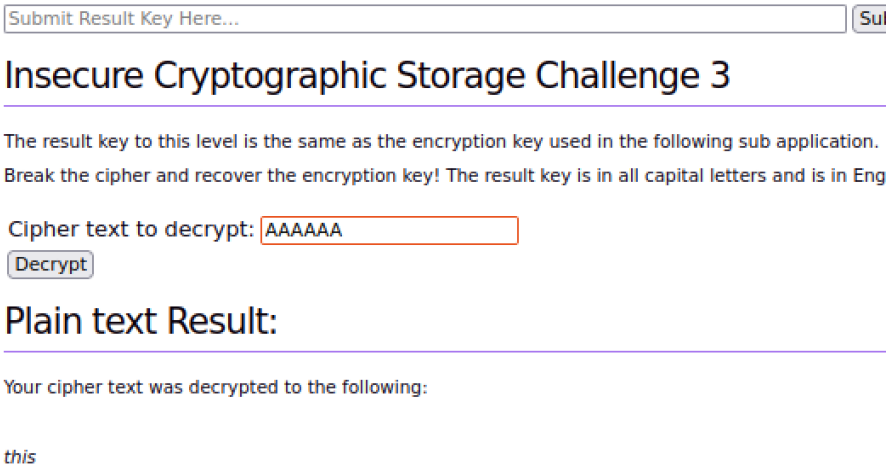
Output: Recovered key: 'thisisthesecurityshepherd'

Cipher Text Used and outputs:

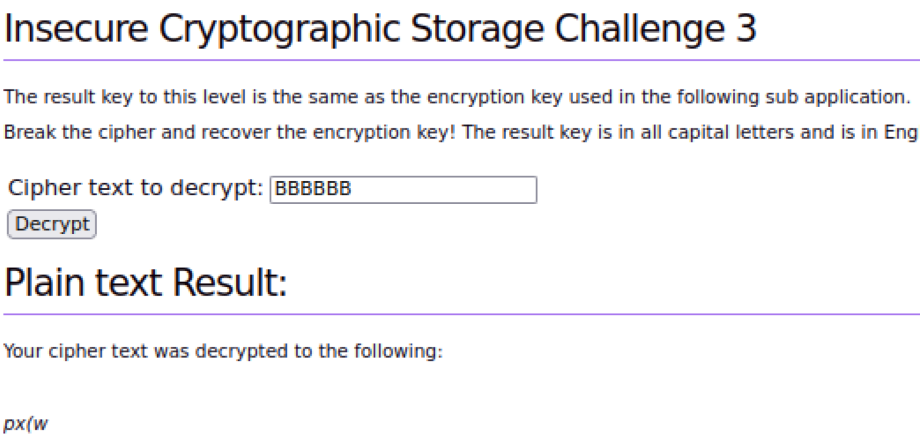
- IAAAAEkQBhEVBwpDHAFJGhYHSBYEGgocAw==



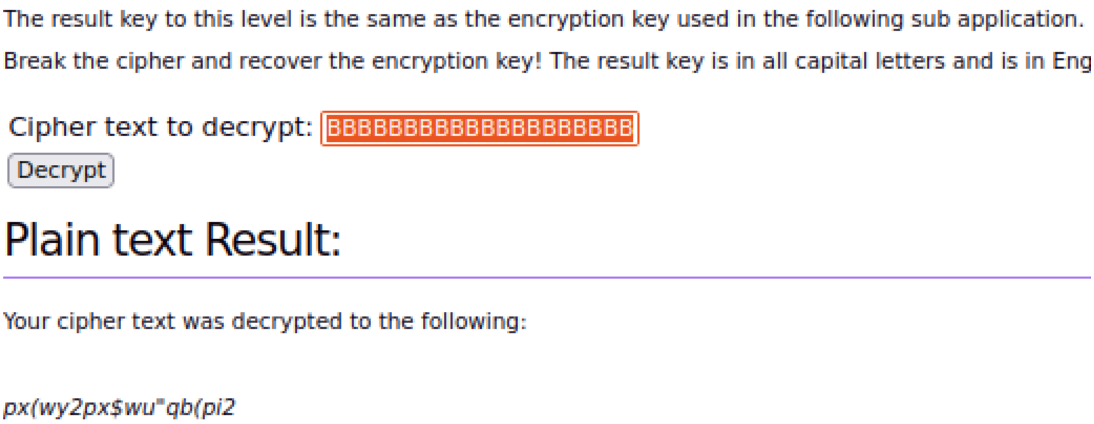
- AAAAAA



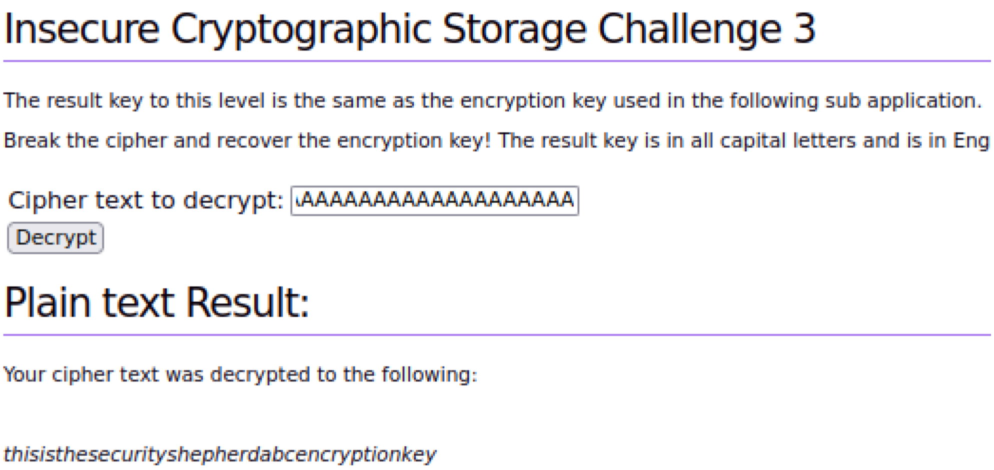
- BBBBB



- BBBBBBBBBBBBBBBBBBBBBBBB



- AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAA



Findings

Critical: Use of a Broken or Risky Cryptographic Algorithm [CWE-327]

A risky or broken cryptographic algorithm is when we use an algorithm to encrypt sensitive information such as a password or an encryption key however as the algorithm used is weak or outdated, improperly handle keys or no hashing is done on passwords, leaves the sensitive information vulnerable to malicious attacks which can steal and use the data.

Steps to reproduce:

1. Got to Security Shepherd https://192.168.1.200
2. Navigate to the Challenges section and look for Insecure Cryptographic Storage
3. Once there select Challenge 3
4. Type in 55 capital letter A's into the decryption box
5. You should see that we get the key which is thisisthesecurityshepherdabcencryptionkey

CVSS Score 9.1

Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High

Availability	None

Mitigation

For this challenge, we find out that they encryption tactic here is a cyclic XOR key as the key itself is only 44 characters but to find the full key, we needed 55 A's. XOR encryption alone is very weak and prone to cracking easily. As such, we should aim to use stronger cryptographic methods such AES or RSA. AES also uses XOR in one of its layers so it would already build upon what we have already. It is usually better to use industry standard cryptography methods and ciphers as they are more secure than custom ones. For this challenge specifically, we are given a cipher text and a plain text which would allow us to gain a bit more information about the key, so a simple solution is to avoid giving people such information so that one is not able to glean any information about encryption method or key. For most cryptography issues, other than using standard encryption methods, it's important to have and use transient keys so the keys are not prone to Man-in-the-Middle attacks and to discard any sensitive information after use.

High: Cross Site Request Forgery in URL Validation [CWE-352]

Cross Site Request Forgery (CSRF) is when a user is tricked by an attacker into making an unintentional request to a web server leading to capture or the exposing of crucial data. CSRF is usually done via, but not limited to a URL, image load, XML or even an HttpRequest, etc.

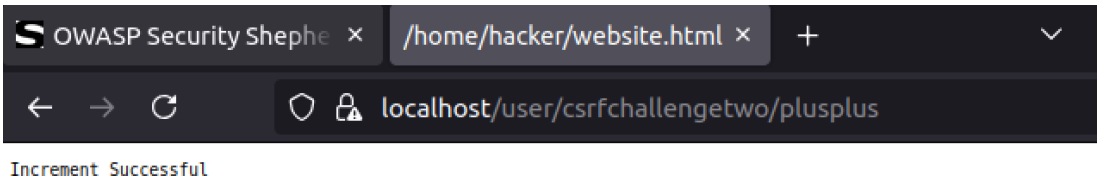
Steps to reproduce:

1. Download and run Burp Suite https://portswigger.net/burp/download.html (making sure you have Oracle Java Installed)
2. Utilising Firefox set the system proxy to route traffic through Burp - "Open Menu" button in the right-hand corner -> Advanced -> Network (tab) -> Connection "Settings Button" -> Manual proxy configuration. The default for Burp is 127.0.0.1 with a port of 8080
3. Got to Security Shepherd https://192.168.1.200
4. Create another 2 users and one new class assign them to said class
5. Open another Security Shepherd Session and log in as the basic user
6. Navigate to the Challenges section and look for CSRF
7. Once there select Challenge 2
8. Let Burp Suite Capture Requests by using Intercept mode
9. Create an HTML document which contains the following script:

```
<html>
<form name="csrfForm" action="https://localhost/user/csrfchallengetwo/plusplus" method="POST">
<input type="hidden" name="userId" value="Replace this with your User ID"/>
<input type="submit"/>
</form>
<script>
document.csrfForm.submit();},3000);
</script>
</html>
```

The user ID is given to us in the challenge or one can use BurpSuite to capture it using the same form

10. Insert the newly created HTML page on the first user's session form.
11. The second user should then click on the link and should see below screenshot



12. Successfully completed a CSRF attack

CVSS Score 7.1

Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	Required
Scope	Unchanged
Confidentiality	Low
Integrity	High
Availability	None

Mitigation

For this challenge, we were able to use a malicious HTML script to gain force a user to increment our own counter by putting said HTML into a comment box and letting the user interact with this. To prevent this, we could prevent only accepting post requests which is

important as in this case, we are attacking the user via a POST request. For more general CSRF mitigation methods, it is important to log off all applications after using them and to change passwords for them more often. Another important mitigation method is to generate per-session nonces and use the same origin policy HTTP header.

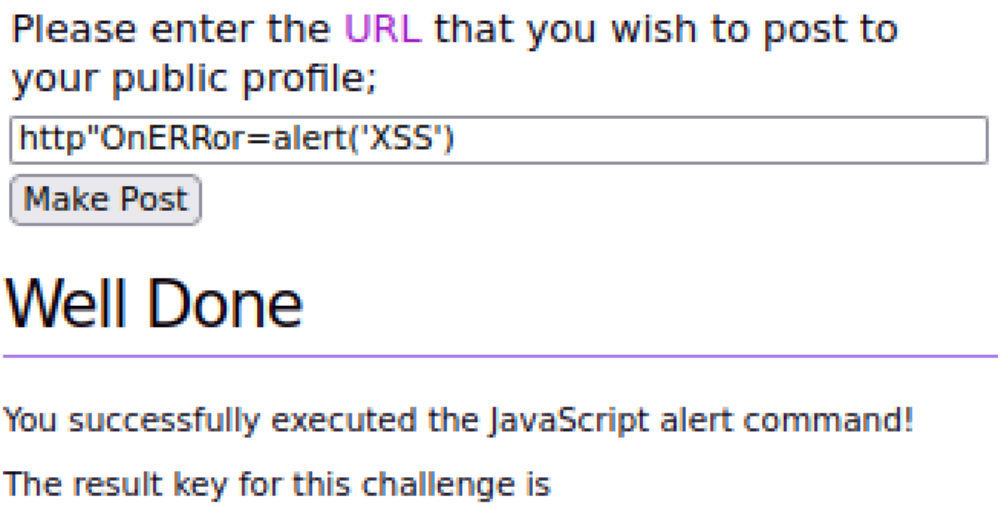
Medium: Cross-Site Scripting in URL Validation [CWE-79]

Cross Site Scripting is when an application accepts untrustworthy data and sends it to a browser without validation or escaping, allowing attackers to execute malicious script in the victim's browser. These scripts are usually injected into content (in this case a URL) which is then uploaded to the website.

Steps to reproduce:

1. Download and run Burp Suite https://portswigger.net/burp/download.html (making sure you have Oracle Java Installed)
2. Utilising Firefox set the system proxy to route traffic through Burp - "Open Menu" button in the right-hand corner -> Advanced -> Network (tab) -> Connection "Settings Button" -> Manual proxy configuration. The default for Burp is 127.0.0.1 with a port of 8080
3. Got to Security Shepherd https://192.168.1.200
4. Navigate to the Challenges section and look for XSS
5. Once there select Challenge 4
6. Let Burp Suite Capture Requests by using Intercept mode

7. Try individual test cases provided in XSS section earlier
8. Open BurpSuite and navigate to the request
9. Note that keyword on is encoded with HTML meaning we need to use lower case and upper case in the style of http"onerror=alert('XSS')
10. Type in http" OneERRor=alert('XSS') and should see the below



11. Successful XSS attack

CVSS Score 5.4

Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	Required
Scope	Unchanged
Confidentiality	Low
Integrity	Low
Availability	None

Mitigation

In this challenge, the idea of encoding is important as it made it more difficult to inject an XSS script. However, stronger encodings need to be used. One crucial one is to force all input to be lowercase or uppercase, not a mixture of both. As mentioned by CWE, different encodings will be needed, especially to prevent scripts which follow a similar pattern. One could also use parametrisation (something to enforce separation between data and code) or use a an “accept known good” input validation strategy here. This means we reject any input that transforms any input or isn’t part of a whitelist.

Recommendations and Conclusions

During penetration testing of the Security Shepherd application, three common types of vulnerabilities were found. Those three being: Cross-Site Scripting, Cross Site Request Forgery and Insecure Cryptographic Storage. Once identified, the application’s vulnerabilities were successfully exploited with the use of publicly available tools and techniques.

For the most critical issue (CVSS of 9.1), insecure cryptography storage, it allowed us to retrieve the key by exploiting the cipher text being displayed and after testing we did find out it was a cyclic XOR method. XOR as an encryption method alone is weak and easily exploitable so it is a critical issue to fix, either by using a different encryption technique, have proper key management or avoid displaying the cipher text.

The next issue was CSRF which scored a CVSS of 7.1. We were able to cause a user to perform a malicious action without their knowledge using a crafted HTML and injecting it in the comment box. It showed us that input was not sanitised or checked for any issues. Going forward, it is better to avoid putting these kinds of forms in GET requests, use CSRF tokens and discard them after use and more importantly, use the SameSite attribute in cookies.

Finally, we also fond Cross Site Scripting vulnerabilities in the application. The vulnerability allowed us to inject malicious JavaScript in the application as there was an improper amount of input sanitisation. To prevent this, it would be better to add further sanitisations to the user input, normalise it or even use whitelisting to prevent it.

While these were the vulnerabilities that were testing, other vulnerabilities might be present within the security such as broken authentication and authorisation. It would be important to add proper security measures to prevent these kinds of vulnerabilities as we would not want users to have access to data they are not meant to have. To do this, we could add MFA to make sure that users are who they say they are while also implementing transient session ID and keys which are later.