



Web Application Penetration Testing Report

Product Name: Security Shepherd

Product Version: v3.0

Test Completion: 27th April 2021

Lead Penetration Tester: Che Yeu Wan

Prepared for: Vsevolods Caka

Consultant Information

Name: Che Yeu Wan

Email: che.wan@ucdconnect.ie

Location: University College Dublin, Belfield, Dublin 4

Manager: Mark Scanlon

Manager email: mark.scanlon@ucd.ie

Sensitive Information

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 to this document.

Table of Contents

CONSULTANT INFORMATION	2
SENSITIVE INFORMATION	3
TABLE OF CONTENTS	4
EXECUTIVE SUMMARY	5
SCOPE	6
TEST CASES.....	7
FINDINGS - INITIAL STEPS.....	8
1. FINDINGS - CROSS SITE SCRIPTING (XSS)	9
2. FINDINGS - SQL INJECTION	11
3. FINDINGS - FAILURE TO RESTRICT URL ACCESS	13
4. FINDINGS - CROSS SITE REQUEST FORGERY	15
5. FINDINGS - INSECURE DIRECT OBJECT REFERENCE	16
6. FINDINGS - INSECURE CRYPTOGRAPHIC STORAGE	17
7. FINDINGS - SESSION MANAGEMENT	18
8. FINDINGS - POOR DATA VALIDATION	20
9. FINDINGS - SECURITY MISCONFIGURATION.....	22
10. FINDINGS - NOSQL INJECTION.....	23
LIST OF VULNERABILITIES.....	25
RECOMMENDATIONS AND CONCLUSIONS	26

Executive Summary

Lead Tester: Che Yeu Wan

Number of days testing:

Testing Start Date: 2nd April 2021

Testing End Date: 24th April 2021

Application Name: Security Shepherd

Application Version: v3.0

Release Date: 29th April 2021

Project Contact: Nikita Pavlenko

OWASP Top 10: 7/10

Total Defects: 10

Severity	# of Defects
Critical	3
High	3
Medium	4
Low	0

Scope

Challenges:

1. Cross Site Scripting - Challenge 5 [CWE-79]
2. SQL Injection - Challenge 4 [CWE-89]
3. Failure to Restrict URL Access - Challenge 3 [CWE-285]
4. Cross Site Request Forgery - Challenge 7 [CWE-352]
5. Insecure Direct Object Reference - Challenge 2 [CWE-639]
6. Insecure Cryptographic Storage - Challenge 3 [CWE-326]
7. Session Management - Challenge 8 [CWE-287]
8. Poor Data Validation - Challenge 2 [CWE-190]
9. Security Misconfiguration - Challenge Cookie Flag [CWE-200]
10. Injection - Challenge NoSQL [CWE-943]

Time Frame:

- Challenges and tests were conducted over a period of 23 days spanning from April 2nd - 24th.
- Report was completed over a period of 5 days spanning from April 25th - 29th.

List of user roles:

- BurpSuite
 - Username: admin
 - Password: password
- SecurityShepherd
- FireFox

List of URLs:

- Security Shepherd:
<https://github.com/OWASP/SecurityShepherd/releases/tag/v3.0>
- OWASP: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- BurpSuite: <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

Test Cases

1. **Cross Site Scripting (XSS):**
 - Testing For Stored Cross Site Scripting (OTG-INPVAL-002)
2. **SQL Injection:**
 - Testing for SQL injection (Improper NeutrOTG-INPVAL-005)
3. **Failure to Restrict URL Access:**
 - Testing For Bypassing Authorization Schema (OTG-AUTHZ-002)
4. **Cross Site Request Forgery:**
 - Testing for CSRF (OTG-SESS-005)
5. **Insecure Direct Object Reference:**
 - Testing for Insecure Direct Object References (OTG-AUTHZ-004)
6. **Insecure Cryptographic Storage:**
 - Testing For Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001)
7. **Session Management:**
 - Testing For Bypassing Session Management Schema (OTG-SESS-001)
8. **Poor Data Validation:**
 - Testing for Buffer Overflow (OTG-INPVAL-014)
9. **Security Misconfiguration:**
 - Testing for Cookies Attributes (WSTG-SESS-02)
10. **NoSQL Injection:**
 - Testing for NoSQL Injection (https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.6-Testing_for_NoSQL_Injection)

Findings - Initial Steps

Initial Steps (For all challenges):

1. Download and run Burp Suite <https://portswigger.net/burp/download.html> (making sure you have Oracle Java Installed)
2. Utilising Firefox set the system proxy to route traffic through Burp - "Open Menu" button in the right hand corner -> Advanced -> Network (tab) -> Connection "Settings Button" -> Manual proxy configuration. The default for Burp is 127.0.0.1 with a port of 8080
3. Download the Security Shepherd VM https://github.com/OWASP/SecurityShepherd/releases/download/v3.1/owaspSecurityShepherd_v3.1_VM.zip then install and run using Virtual Box (or any other VM application)
4. Got to Security Shepherd <https://192.168.0.248> using Firefox
5. Confirm that Burp can see and capture requests and turn off intercept in Burp
6. If you already have login details setup for Security Shepherd, login using those details. If not login using the user role outlined on Page 6

1. Findings - Cross Site Scripting (XSS)

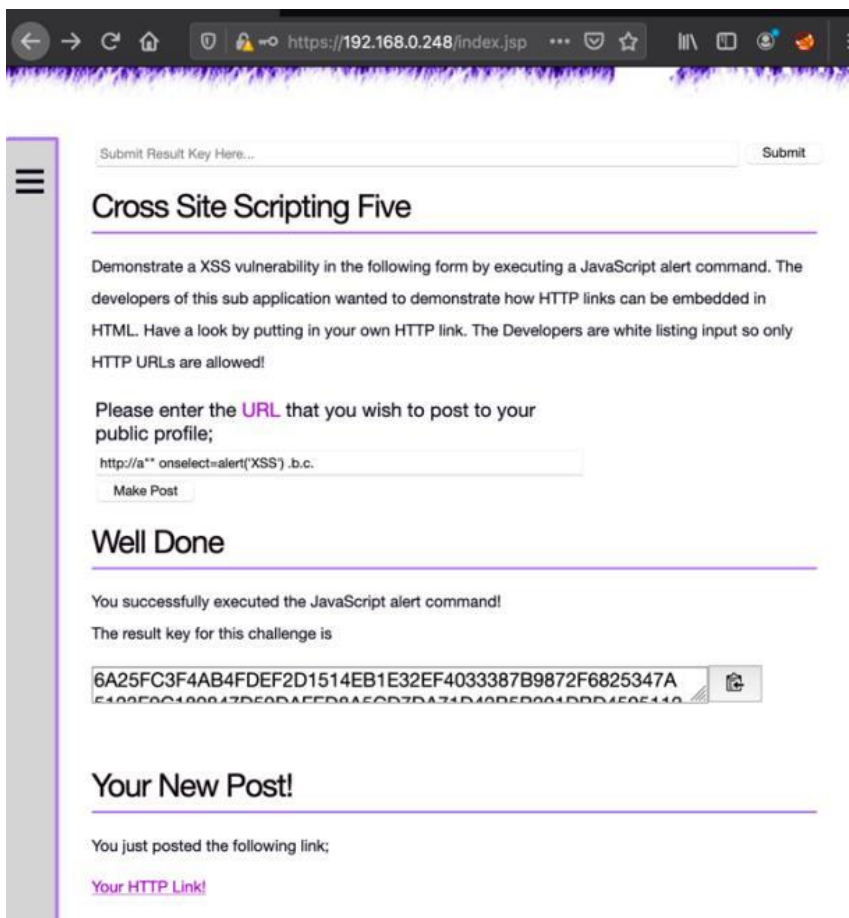
Critical: Cross Site Scripting (XSS) [CWE-79]

Cross Site Scripting(XSS) happens when an application uses untrusted data to show on browser without sufficient validation. If untrusted data contains client side scripts the browser will execute the script when it is interpreting the page.

Attackers can hijack user sessions, change original web sites or redirect users to malicious sites by leveraging XSS attacks to execute scripts in a victims browser. XSS is the most widespread vulnerability found in modern web applications due to the variety of attack vectors available

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> XSS -> Cross Site Scripting 5
3. To complete this challenge we must execute a JavaScript alert command
4. Enter the following as the URL **http://a"" onselect=alert('XSS') .b.c.**
5. Click Make Post and the result key will be shown below
6. Submit the result key to complete the challenge



CVSS Score 9.0 (Critical)

Attack Vector	Network
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

Mitigation

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

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.

2. Findings - SQL Injection

Critical: SQL Injection [CWE-89]

SQL Injection is when malicious data is sent to a server and the server trusts the data without a proper examination of the data. The malicious data would be treated as a normal command and executed with the privilege of corresponding server side application.

These are high severity attacks that can lead to confidential data leakage directory traversal, file execution and fully controlled by attackers. This attack makes systems wide open to attackers. It can be initiated by anyone who can connect to the system through data they pass to the application.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Injection -> SQL Injection 4
3. To complete this challenge we must sign in as admin
SQL Injection 4

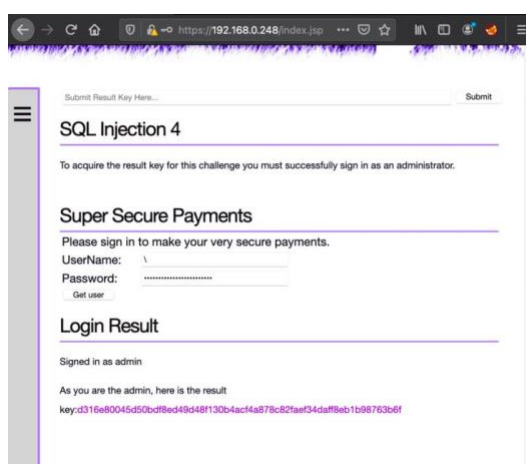
To acquire the result key for this challenge you must successfully sign in as an administrator.

Super Secure Payments

Please sign in to make your very secure payments.

UserName:
Password:

- 4.
5. For UserName we use enter \
6. For Password we enter **or username="admin";--**
7. As shown below in the screenshot, this successfully signs you in and the Result Key is shown below
8. Submit the result key and the challenge will be completed



CVSS Score 10.0 (Critical)

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

Mitigation

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.

3. Findings - Failure to Restrict URL Access

High: Failure To Restrict URL Access [CWE-285]

Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Failure To Restrict URL Access -> Challenge 3
3. Turn on intercept in Burp and Click on "Admin Only" button
4. We look up what the value is for currentPerson and we see that it is a Base64 encoding for aGuest.
5. We check the source code and looking at it we find a form with a url


```
document.cookie="currentPerson=YUdlZXN0";
var ajaxCall = $.ajax({
  type: "POST",
  url: "e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4UserList",
  async: false
});
```
- 6.
7. We go back into BurpSuite and send a post request using the URL found above.
8. We then see on the challenge page that a list has come up

Admin Only Button

```
aGuest
manager
sean
root
administrator
privileged
seanduggan
markdenihan
mark
superuser
megauser
hyperuser
godzilla
kinguser
rootuser
adminuser
shepherd
```

- 9.
10. Since the currentPerson was aGuest, we need to change the currentPerson value to something that will allow us to request a list for admins.
11. We use base64 encoding for the value "or"1"="1 as this will allow us to make use of the fact that this parameter could be vulnerable to SQL injection. Forward on the interception after changing the value of currentPerson.

Failure to Restrict URL Access Challenge 3

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

Admin Only Button

aGuest
manager
sean
root
administrator
privileged
seanduggan
markdenihan
mark
superuser
MrJohnReillyTheSecond
megauser
hyperuser
godzilla
kinguser
rootuser
adminuser
shepherd

- 12.
13. We compare the two lists and we see that **MrJohnReillyTheSecond** is exclusive to this second list.
14. We use base64 encoding for this username and insert what we get as the value for **currentPerson**, and we forward again.
15. Challenge is complete, copy and paste the result key into the box.

CVSS Score 7.2 (High)

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

Mitigation

Divide the software into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

4. Findings - Cross Site Request Forgery

Medium: Cross Site Request Forgery [CWE-352]

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> CSRF-> Challenge 7
TODO

CVSS Score 5.4 (Medium)

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

Mitigation

Do not use the GET method for any request that triggers a state change.

Ensure that the application is free of cross-site scripting issues as most CSRF defences can be bypassed using attacker-controlled script.

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

5. Findings - Insecure Direct Object Reference

High: Insecure Direct Object Reference [CWE-639]

Insecure Direct Object Reference is where a malicious user can change the value of a parameter to bypass the authorisation of the application. The application fails to verify the authorisation for a target object in this case the 'username' parameter. Changing the value of the username will bypass the authorisation and allow an attacker to see other users details in the system for example the admin user.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Insecure Direct Object Reference -> Challenge 2
3. Turn on intercept in Burp.
4. Go through every user in the list and intercept each HTTP request. Make note of each userId's MD5 value.
5.

```
userId%5B%5D=6512bd43d9caa6e02c990b0a82652dca  n26L.Iq#2B#2D=8LJd6d2Lc669J2\9293eqq6952d3
userId%5B%5D=c81e728d9d4c2f636f067f89cc14862c  userId%5B%5D=eccbc87e4b5ce2fe28308fd9f2a7baf3
userId%5B%5D=e4da3b7fbbce2345d7772b0674a318d5
```
6. Looking at all the values we see that we get 2, 3, 5, 7, 11. These are all prime numbers so as a guess we take 13 and use MD5 hashing. We get **c51ce410c124a10e0db5e4b97fc2af39**
7. We use this as the userId value and forward the HTTP request. We then get the result key and the challenge is complete.

CVSS Score 7.2 (High)

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

Mitigation

For each and every data access, ensure that the user has sufficient privilege to access the record that is being requested. Make sure that the key being used cannot be controlled externally and that any tampering of the key can be detected.

Use encryption to stop the key being guessed and to ensure that it has not been tampered with

6. Findings - Insecure Cryptographic Storage

High: Insecure Cryptographic Storage [CWE-326]

Insecure Cryptographic Storage isn't a single vulnerability, but a collection of vulnerabilities that all have to do with making sure that your most important data is encrypted when it needs to be. This includes, but isn't limited to, making sure you are encrypting the correct data, making sure you have proper key storage and management, making sure that you are not using known bad algorithms and making sure you are not implementing your own cryptography, which may or may not be secure.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Insecure Cryptographic Storage -> Challenge 3
3. TODO

CVSS Score 7.5 (High)

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

Mitigation

Use a certified encryption scheme that is considered to be strong by experts in the field of data encryption. Another way to prevent this is to only store necessary data online and if possible keep sensitive data offline instead of online. This must be stored offline where the user cannot directly access it.

7. Findings - Session Management

Critical: Session Management [CWE-287]

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Session Management -> Challenge 8
3. Turn on intercept in Burp.

```
Cookie: challengeRole=LmH6nmbC; JSESSIONID=6D27D5A83FBAC29AF449E2387BC78F1F; token=135636708303766387706989682725427765061; JSESSIONID3="SgINJWrlrvk6fC7JTLsx/A=="  
Connection: close  
  
returnUserRole=false&returnPassword=false&adminDetected=false
```

4. We click the button and check what is intercepted. What we see is that there is an adminDetected parameter. We set this to **true**.
5. Now we take the value of **challengeRole** as search online for what the value is. We find that **LmH6nmbC** decrypts to **guest** using **ATOM-128**.
6. Using ATOM-128 encryption we can try the word **admin**. However the response is **INVALID ROLE DETECTED**
7. We then try superuser using ATOM-128 encryption to get **nmHqLjQknIHs**. We try this and forward the request.
8. The result key is then given and challenge is completed.

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

2UMX99zPVN3soMOZEW8f3PNEzCGsryqk4C6TsQ+LnCDZtiq4M6mFDG
YtLazKcQShKbxAk7fUtoChCfZweY25clYc0Bldeb7/SEdals7Dl5DuEnrEK



CVSS Score 10.0 (Critical)

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

Mitigation

Use an authentication framework or library such as the OWASP ESAPI Authentication feature

8. Findings - Poor Data Validation

Medium: Poor Data Validation [CWE-190]

An integer overflow or wraparound occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value may wrap to become a very small or negative number. While this may be intended behaviour in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected. This is especially the case if the integer overflow can be triggered using user-supplied inputs. This becomes security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviours such as memory allocation, copying, concatenation, etc.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Poor Data Validation -> Challenge 2





Poor Validation Two

If you can buy trolls for free you'll receive the key for this level!

Super Meme Shopping

Use this shop to buy whatever old memes you like!

Picture Cost Quantity

	\$45	<input type="text"/>
	\$15	<input type="text"/>
	\$3000	<input type="text"/>
	\$30	<input type="text"/>

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

- 3.
4. Start off with a negative number for Quantity beside the Troll meme to test for data validation for negative numbers.
5. We see that there is data validation as the total comes to \$0.
6. We now try an extremely large positive number to test boundary issues. For example using 99999999, We can see that this works and that the total comes to a negative number, showing that there is a integer buffer overflow.

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**

7. We see that this works and that the result key is given. Challenge is completed.

CVSS Score 4.3 (Medium)

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

Mitigation

Ensure that all protocols are strictly defined, such that all out-of-bounds behaviour can be identified simply, and require strict conformance to the protocol.

Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

If possible, choose a language or compiler that performs automatic bounds checking.

Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range.

Use unsigned integers where possible. This makes it easier to perform validation for integer overflows. When signed integers are required, ensure that the range check includes minimum values as well as maximum values.

9. Findings - Security Misconfiguration

Medium: Security Misconfiguration Cookie Flag [CWE-200]

The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.

The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Security Misconfigurations -> Security Misconfigurations Cookie Flag
3. TODO

CVSS Score 4.0 (Medium)

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

Mitigation

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design, and the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

Always set the secure attribute when the cookie should be sent via HTTPS only.

10. Findings - NoSQL Injection

Medium: NoSQL Injection [CWE-943]

NoSQL injection vulnerabilities allow attackers to inject code into commands for databases that don't use SQL queries, such as MongoDB. NoSQL injection attacks can be especially dangerous because code is injected and executed on the server in the language of the web application, potentially allowing arbitrary code execution.

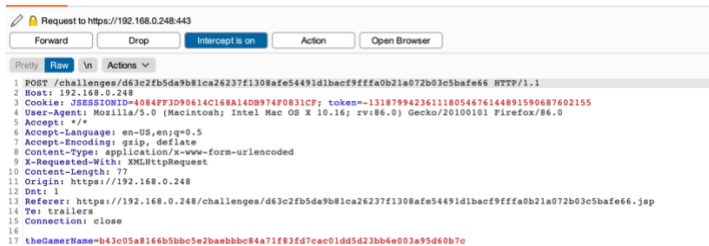
Steps to reproduce

1. Follow Initial Steps
2. Go to Challenges -> Injection -> NoSQL Challenge
3. Go to BurpSuite and turn on intercept
4. Press the "Get Gamer Info" button in the challenge page

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

Get Gamer Info

5. Looking at the captured request we see that the GamerId is what needs to be modified.



6. Modify the field by adding a ' character to the end of the GamerId.
7. We see that an error is detected and we discover that MongoDB is being used by the application.

Gamer Info

An error was detected!

com.mongodb.MongoException: Can't canonicalize query :: caused by :: Failed to call method

GamerId Name Address

8. Using the request `x'; return(true); var x = 'x`
9. Forward the request and you will see that the list of details will appear in the challenge page.

Gamer Info

GamerId	Name	Address
77b6eab74151d73c2efe561737981a1fbb2291fe3643680f5824e47f69fc9985	Omar	Baltimore
fd1b4a1d16714cee9c1320f3e7465792010d0911075b7c1002071d48e68e19b4	Stringer	Baltimore
b43c05a8166b5bbc5e2baebbbc84a71f83fd7cac01dd5d23bb6e003a95d60b7c	Jimmy	Baltimore
c09f32d4c3dd5b75f04108e5ffc9226cd8840288a62bdaf9dc65828ab6eaf86a	Marlo	Baltimore
b6c02d3459803a3e3bf99a8c5a20e2f661a8296bbeec858110d3597aaa9b830a	Clayton	Baltimore
4b54501900f31f40832a67accf8ab97150a7cb7938d814dff534e64b1e658bd	Lester	Baltimore

CVSS Score 6.5 (Medium)

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

Mitigation

Use of a firewall

Using shared libraries with valid security-testing and input sanitization. This ensures special characters are not being submitted and that the input is the type you intend for it to be instead of being user-controlled.

List of Vulnerabilities

[Critical] **10.0** Session Management

[Critical] **10.0** SQL Injection

[Critical] **9.0** Cross Site Scripting (XSS)

[High] **7.5** Insecure Cryptographic Storage

[High] **7.2** Insecure Direct Object Reference

[High] **7.2** Failure to Restrict URL Access

[Medium] **6.5** NoSQL Injection

[Medium] **5.4** Cross Site Request Forgery

[Medium] **4.3** Poor Data Validation

[Medium] **4.0** Security Misconfiguration Cookie Flag

Recommendations and Conclusions

Looking at the List of Vulnerabilities on the preceding page, they are listed in descending order based on the CVSS score. This will give an idea of what vulnerabilities pose the biggest threat to the web application and should be given the most attention

The first step I would recommend taking is to go through the Findings section of this report and reproduce the steps. This will aid in understanding and diagnosing the vulnerability which can then be mitigated by reading through the mitigations section along with reading up in more detail on the OWASP link provided.

The fact that 7 of the 10 vulnerabilities found in this application are present in the 2017 OWASP Top 10, it shows that there is a huge lack of security and a lot of security issues and vulnerabilities are present. Following through with the suggestions in the mitigation sections of each vulnerability as mentioned above will improve this vastly. Proper security testing needs to be integrated into the development/testing workflows.

With all this being said there is no impenetrable website and every web application or website will be prone to different vulnerabilities. Through proper planning and testing however, the number of vulnerabilities and the severity of these vulnerabilities can be controlled to an extent. The links provided in this report will give huge insight into the shortcomings of this web application and improvements can be made.