CMP-6045B Developing Secure Software – Report 2: Allocated System Testing

Group UG-4:
100263580 – Steven Diep
100251448 – Sam Humphreys
100225776 – Martin Siddons
6214363 – Chris Sutton

1 Introduction

The process of cybersecurity involves playing the attacker as well as the defender. The 'red team' role is useful for highlighting security weaknesses within an application, which we will use here to test the site built by group UG-3. When searching for vulnerabilities, a list of tests is required to find those which are common and applicable to the site's functionality.

2 Analysis of Testing Approach

For **account enumeration**, we will be looking to test if the error for incorrect password and non-existent username are the same. We will check if the time taken for the system to reply to a login is consistent for existing and non-existing users. We will look for hints of valid usernames in URLs, page titles and HTML packets. When resetting the password, we will see if the site tells us if an account doesn't exist. We will test if login names and display names are different.

To test for **session hijacking**, we will look for client information to see if sessions have been implemented, and if there is a time limit on the session. We will look to see if a session key is returned in a URL, header or cookie in an unencrypted form, and also see if changing the source IP address is possible between requests.

For **SQL injection**, we will test input forms, `POST` requests, `GET` requests, cookies and headers for vulnerabilities. We will try piggybacking queries, tautologies, abusing character encodings, illegal queries and `UNION` statements.

For testing **XSS**, we will check to see if certain characters (`& < > " ' % * + , - / ; =` ^|) are removed or encoded from input forms. From there, we will look at injecting scripts into the page to carry out both persistent and reflected attacks.

We will look to see if the site is using a **CSRF** token hidden on each form once logged in. We will also look to see if the user is forced to re-authenticate during a session.

We will check for a limit on login attempts for the possibility of **brute-forcing** the password. We will look inside the **database** to see what information an attacker could extract from it.

3 Evidence of System Testing

Upon starting the site we noticed that there is no option for a user to create an account, and since the developers have not given a visitor access to a guest account, we would be unable to test the internal pages of the site. It also appeared that the user search functionality was broken as it did not respond to any queries.

We found through testing for **account enumeration** that error for incorrect password and non-existent username are the same. Login name is the same as display name, so we were able to find all usernames from the front page posts. Timing is different for existing and non-existing usernames - correct usernames take longer to process. URL for the user page is the same as the login name. Password reset link only says it sends an email, which is secure. The URL for user page returns whether a user exists, if user does not exist it returns `No such user`.

We were unable to test **session hijacking** this since there is no way to create an account, therefore there is no way to find if a session is implemented. However, checking the code we can see default sessions appear to be implemented and that `PERMANENT_SESSION_LIFETIME` is set to fifteen, meaning the session expires fifteen minutes after closing the browser.

We tested for **SQLi**, on the Username field on the Login form and the same on the Reset Password form. We attempted `'OR 1=1--` for tautology, `0x27OR 1=1--` for single character encoding and all for hex encoding, and illegal queries (`AND CONVERT (char, no)`) to force an SQL error. We would have used a `UNION` call if there was an input that sent back a result to the browser. The same attacks were also submitted as a URL request to `/loginfail/?error=[query]` and `[sitename]/[query]/`, but the site did not respond to these attacks. We wanted to try submitting cookies with SQLi, however there were no cookies on the site. We attempted to submit our own requests outside the browser - by using Wireshark we could see what a submitted form looks like (figure 1), we then edited the header and submitted it with the Linux curl command, however this was not successful.

Due to being unable to create an account, we could not access the ability to post, which would be the only way we could push data directly into the database and do a **stored XSS** attack. For a **reflected** attack, we attempted to post a standard attack `<script>alert(1)</script>` on both the previously mentioned search boxes, as well as the previously mentioned URL fields with no luck. Testing `&<>"'%*+,-/;=^|` showed that the server escapes all characters (figure 2). However, we were able to perform a reflected XSS attack by using HTML encoding to defeat the escaping of normal quotations. By submitting `<img src="https://duckduckgo.com/i/b2e38b1b.png"onmouseover=alert""(&quot;abcdef&quot;)">` to the password reset, the page returned with the error in figure 3, which tells us an exception occurred and the site is vulnerable to SQLi and Stored XSS should we have the ability to post. We decided to use information in the server code to break into an account to test this on the create post page, and found that it submitted to the database. Using this technique we also could demonstrate redirecting to an attacker controlled website with the content: `<img src="https://duckduckgo.com/i/b2e38b1b.png"onmouseover="window.open(&quot;http://www.google.com&quot;,&quot;_blank&quot;)">`.

None of the forms we could access appear to include a **CSRF** token, so it would be possible to embed a request on another site and have it executed if you knew a user had an active session. This would be done by partially filling in a form URL and having a validated user click the link to submit a request from them with the details we entered.

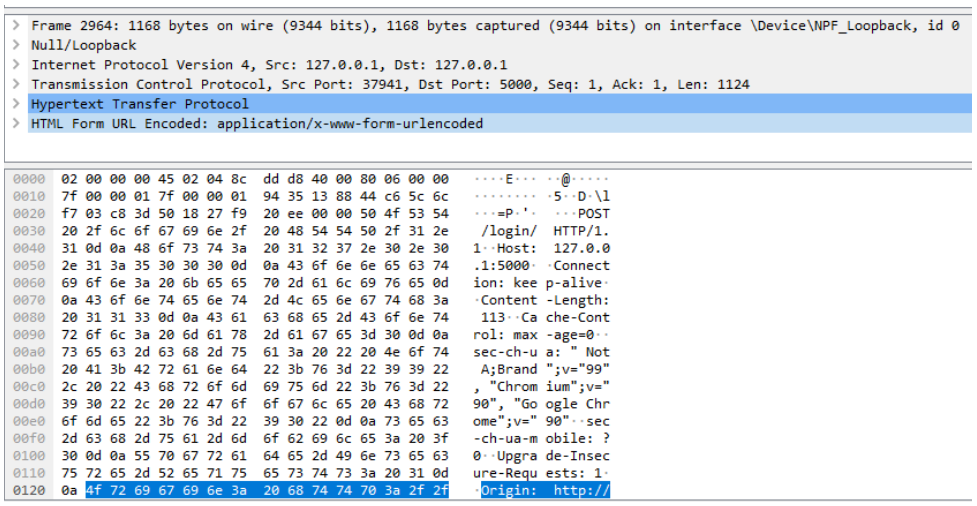
Looking at **further vulnerabilities**, the site uses a graphical password system that uses 3 (X,Y) coordinates sent in the `POST` request. The size of the image reveals the range of possible X values as 0-1500 and possible Y values as 0-1000. There is no restriction on the number of logins, therefore we attempted to brute force entry using Burp Suite as shown in Figures 4, 5, 6. We determined the maximum number of combinations to be $(1500 \times 1000)^3 = 3.375 \times 10^{15}$ as a worst case. As this would take too long to iterate, so we did not continue. The Database in figure 7 shows that every field aside from the username appeared to be hashed. Although great for security, we did question the usability of having information such as name and email hashed rather than encrypted, as they are now unable to be returned.

4 Conclusion

The inability to log in easily stymied our investigation however our testing revealed several vulnerabilities with the site, and some serious usability problems. As we have proven, it takes only one incorrect setting to bypass the rest of a site's security.

A Appendix A

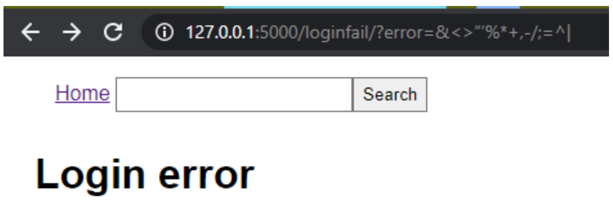
A.1 Figures



```
> Frame 2964: 1168 bytes on wire (9344 bits), 1168 bytes captured (9344 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 37941, Dst Port: 5000, Seq: 1, Ack: 1, Len: 1124
> Hypertext Transfer Protocol
> HTML Form URL Encoded: application/x-www-form-urlencoded

0000  02 00 00 00 45 02 04 8c  dd d8 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  94 35 13 88 44 c6 5c 6c   ········ ·5··D·\l
0020  f7 03 c8 3d 50 18 27 f9  20 ee 00 00 50 4f 53 54   ···=P·'·  ···POST
0030  20 2f 6c 6f 67 69 6e 2f  20 48 54 54 50 2f 31 2e    /login/  HTTP/1.
0040  31 0d 0a 48 6f 73 74 3a  20 31 32 37 2e 30 2e 30   1··Host:  127.0.0
0050  2e 31 3a 35 30 30 30 0d  0a 43 6f 6e 6e 65 63 74   .1:5000· ·Connect
0060  69 6f 6e 3a 20 6b 65 65  70 2d 61 6c 69 76 65 0d   ion: kee p-alive·
0070  0a 43 6f 6e 74 65 6e 74  2d 4c 65 6e 67 74 68 3a   ·Content -Length:
0080  20 31 31 33 0d 0a 43 61  63 68 65 2d 43 6f 6e 74    113··Ca che-Cont
0090  72 6f 6c 3a 20 6d 61 78  2d 61 67 65 3d 30 0d 0a   rol: max -age=0··
00a0  73 65 63 2d 63 68 2d 75  61 3a 20 22 20 4e 6f 74   sec-ch-u a: " Not
00b0  20 41 3b 42 72 61 6e 64  22 3b 76 3d 22 39 39 22    A;Brand ";v="99"
00c0  2c 20 22 43 68 72 6f 6d  69 75 6d 22 3b 76 3d 22   , "Chrom ium";v="
00d0  39 30 22 2c 20 22 47 6f  6f 67 6c 65 20 43 68 72   90", "Go ogle Chr
00e0  6f 6d 65 22 3b 76 3d 22  39 30 22 0d 0a 73 65 63   ome";v=" 90"··sec
00f0  2d 63 68 2d 75 61 2d 6d  6f 62 69 6c 65 3a 20 3f   -ch-ua-m obile: ?
0100  30 0d 0a 55 70 67 72 61  64 65 2d 49 6e 73 65 63   0··Upgra de-Insec
0110  75 72 65 2d 52 65 71 75  65 73 74 73 3a 20 31 0d   ure-Requ ests: 1·
0120  0a 4f 72 69 67 69 6e 3a  20 68 74 74 70 3a 2f 2f   ·Origin:  http://
```

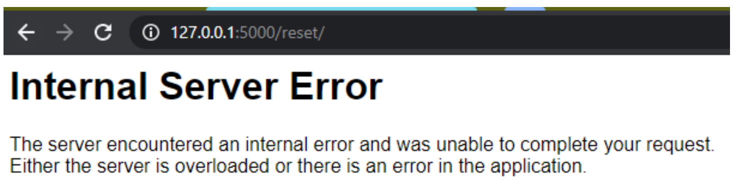
Figure 1: An image showing the output of Wireshark from a login request to the server.



127.0.0.1:5000/loginfail/?error=&<>"'%*+,-/;=^|

Login error

Figure 2: Login Error received when trying to insert restricted characters into the form. Note how the characters are not printed below the title as expected.

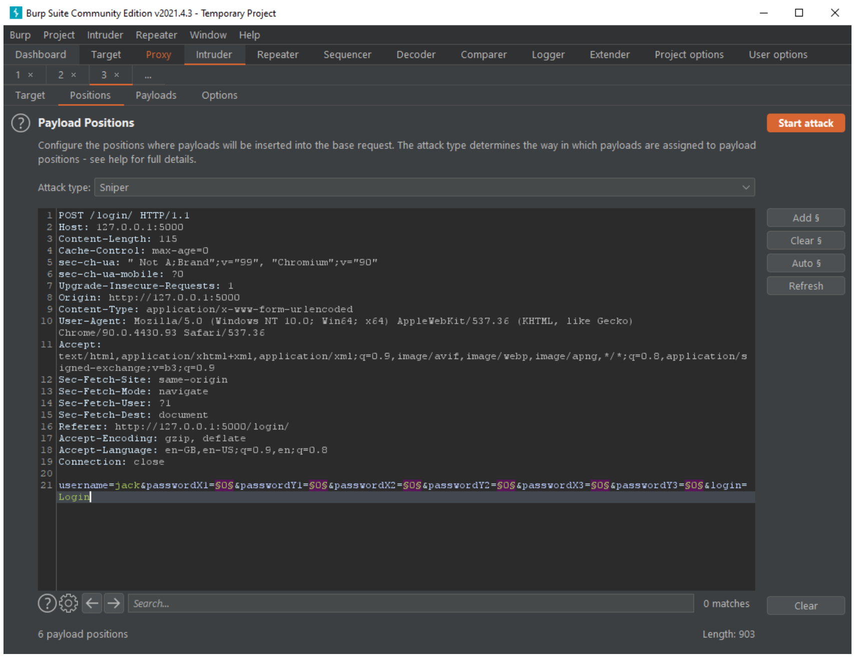


127.0.0.1:5000/reset/

Internal Server Error

The server encountered an internal error and was unable to complete your request.
Either the server is overloaded or there is an error in the application.

Figure 3: Error sent back from the server indicating an exception was generated on the server during processing, showing the SQLi was successful.



Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
POST /login/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Length: 115
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/90.0.4430.93 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:5000/login/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Connection: close

username=jack&passwordX1=§0§&passwordY1=§0§&passwordX2=§0§&passwordY2=§0§&passwordX3=§0§&passwordY3=§0§&login=
Login
```

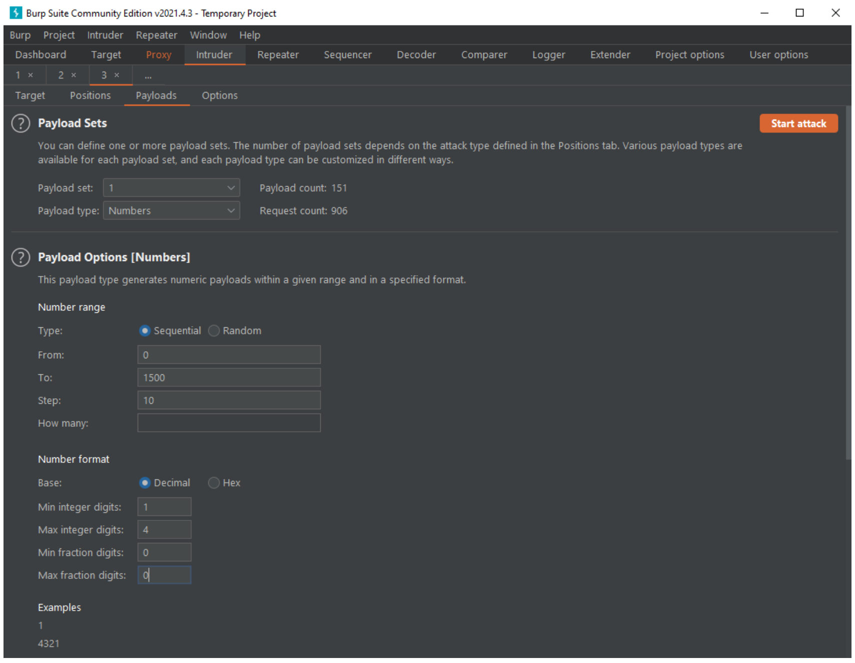
0 matches

6 payload positions

Length: 903

Burp Suite Community Edition v2021.4.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options
Target Positions Payloads Options
Start attack
Add §
Clear §
Auto §
Refresh
Clear

Figure 4: Figure of Burp Suite showing the HTTP request with the variables being changed in the payload.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload count: 151

Payload type: Numbers

Request count: 906

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: [✓ Sequential] Random

From: 0

To: 1500

Step: 10

How many:

Number format

Base: [✓ Decimal] Hex

Min integer digits: 1

Max integer digits: 4

Min fraction digits: 0

Max fraction digits: 0

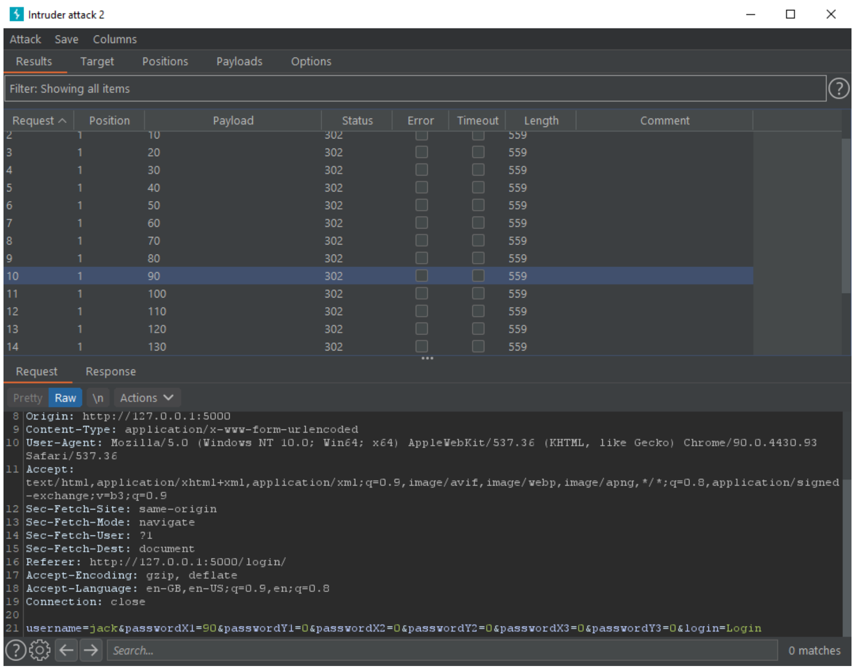
Examples

1

4321

Burp Suite Community Edition v2021.4.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options
Target Positions Payloads Options
Start attack

Figure 5: Figure showing the payload in Burp Suite to be sent to the server.

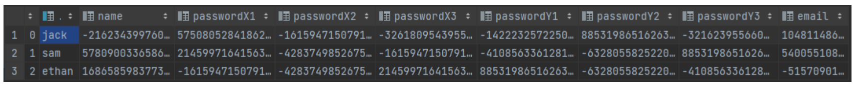


Request	Position	Payload	Status	Error	Timeout	Length	Comment
2	1	10	302	☐	☐	559	
3	1	20	302	☐	☐	559	
4	1	30	302	☐	☐	559	
5	1	40	302	☐	☐	559	
6	1	50	302	☐	☐	559	
7	1	60	302	☐	☐	559	
8	1	70	302	☐	☐	559	
9	1	80	302	☐	☐	559	
10	1	90	302	☐	☐	559	
11	1	100	302	☐	☐	559	
12	1	110	302	☐	☐	559	
13	1	120	302	☐	☐	559	
14	1	130	302	☐	☐	559	

```
Origin: http://127.0.0.1:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed
-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:5000/login/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Connection: close

username=jack&passwordX1=90&passwordY1=0&passwordX2=0&passwordY2=0&passwordX3=0&passwordY3=0&login=Login
```

Figure 6: Figure showing Burp Suite executing the attack on the server.



	.	name	passwordX1	passwordX2	passwordX3	passwordY1	passwordY2	passwordY3	email
1	0	jack	-216234399760…	57508052841862…	-1615947150791…	-3261809543955…	-1422232572250…	88531986516263…	-321623955660…	104811486…
2	1	sam	5780900336586…	21459971641563…	-4283749852675…	-1615947150791…	-4108563361281…	-6328055825220…	8853198651626…	540055108…
3	2	ethan	1686585983773…	-1615947150791…	-4283749852675…	21459971641563…	88531986516263…	-6328055825220…	-410856336128…	-51570901…

Figure 7: A look at the SQLite database as seen from within PyCharm.