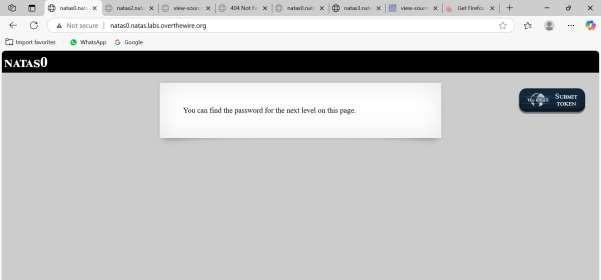
Name: Virali Gada

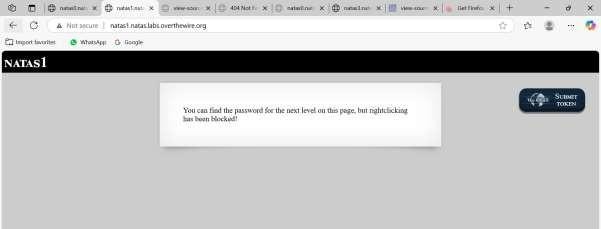
Intern id:303

NATAS REPORT



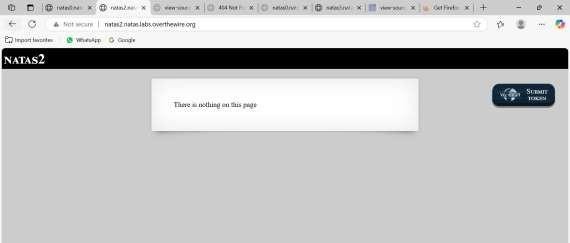
* + Sign-in in natas0 > Right click > click on View source file >In index you will get the password for natas1 > 0nzCigAq7t2iALyvU9xcHlYN4MlkIwlq

### Natas1

****

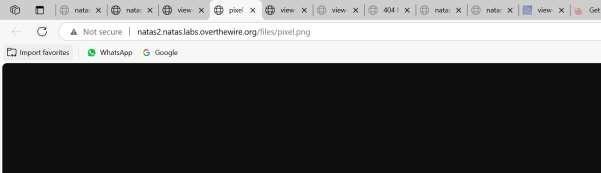
* + Sign-in into natas1 > Go to 3 dots > select more tools> select developer tools> In index you will get the password for natas2 > TguMNxKo1DSa1tujBLuZJnDUlCcUAPlI

### Natas2

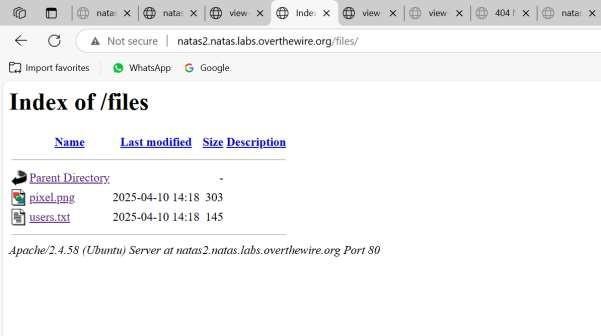


* + Sign-in into natas2 > Right click > select view source file > click on

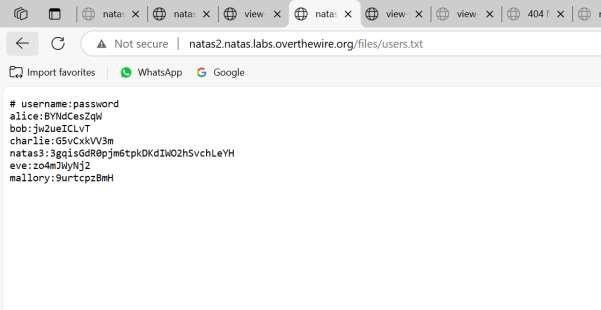
"[files/pixel.png](http://natas2.natas.labs.overthewire.org/files/pixel.png)" > this page will open.



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

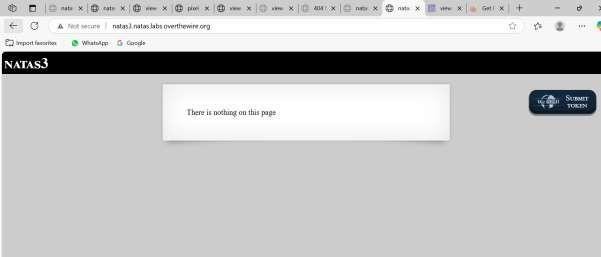


* + This page will visible > go to users.txt

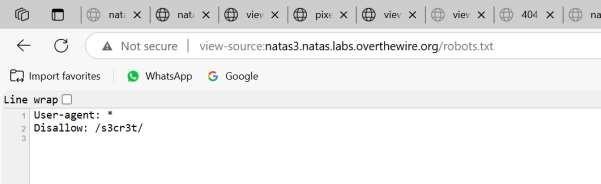


* + The password will show for natas3 > 3gqisGdR0pjm6tpkDKdIWO2hSvchLeYH

### 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

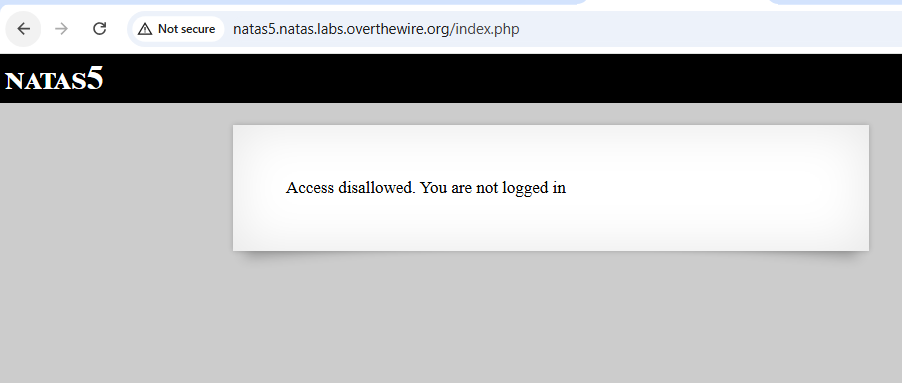
****

* + 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](https://reqbin.com/)

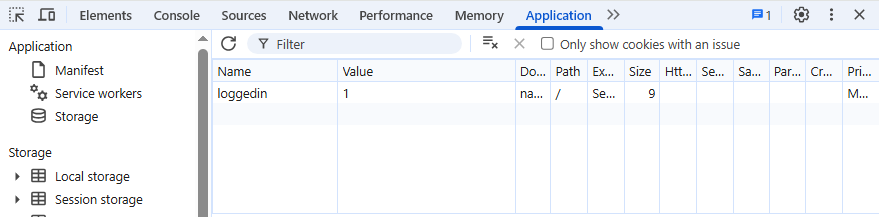
Select curl and Paste the code which you copyed to run box and run it.

* + Find the password in code for natas5 is 0n35PkggAPm2zbEpOU802c0x0Msn1ToK .

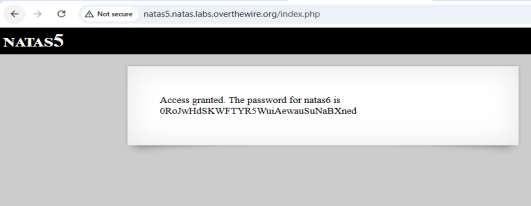
### Natas5

****

* + 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.

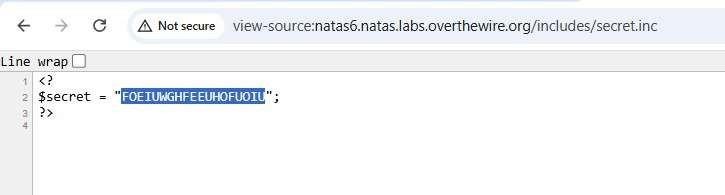


### Natas6

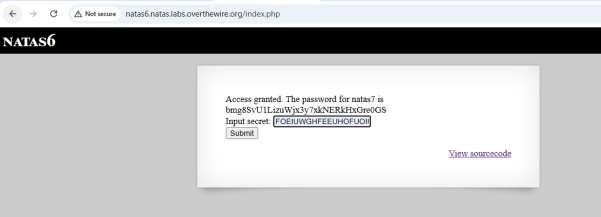
* + Sign-in natas6 > it will ask input secret



* + 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=aboutpage to this [http://natas7.natas.labs.overthewire.org/index.php?page=/etc/natas\_w](http://natas7.natas.labs.overthewire.org/index.php?page=/etc/natas_webpass/natas8) [ebpass/natas8](http://natas7.natas.labs.overthewire.org/index.php?page=/etc/natas_webpass/natas8).](http://natas7.natas.labs.overthewire.org/index.php?page=about)
  + The below page has contain the password.



### Natas8

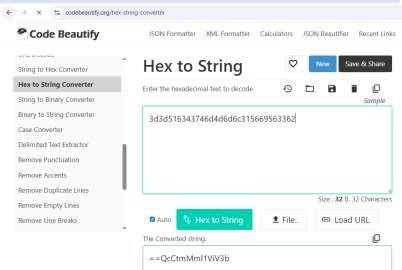
* + Sign in to natas8. It is asking the input secret.



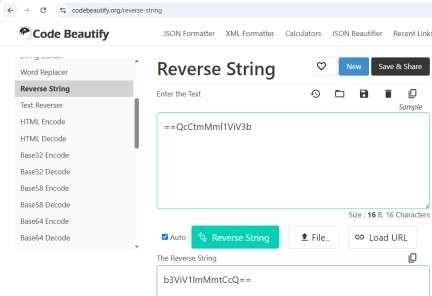
* + Click view sourcecode > it contains all instructions as given below:



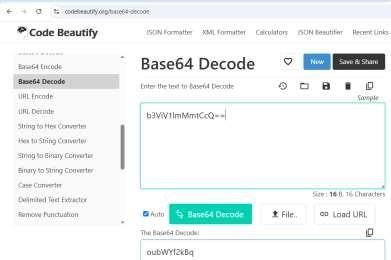
* + Convert the given encodedsecret to string as below:



* + 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.

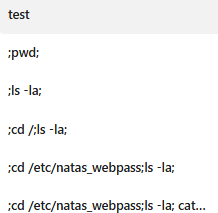


### Natas9

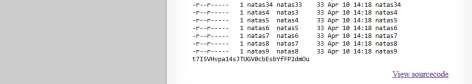
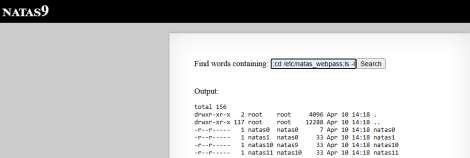
* + Sign in natas9



* + Search each an every option given below.



* + After all search this will appear as given in image.



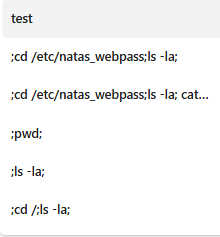
* + The password will appear for natas10 .

### Natas10

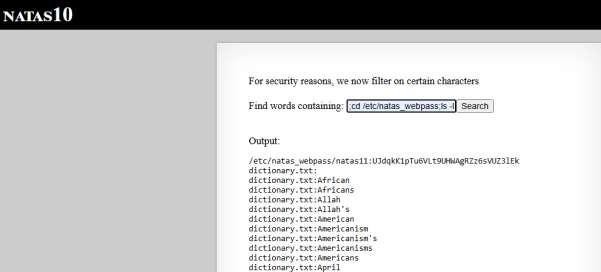
* + Sign in natas10



* + Search each an every option given below.



* + After all search this will appear as given in image.



* + The password will appear for natas11.

### Natas11

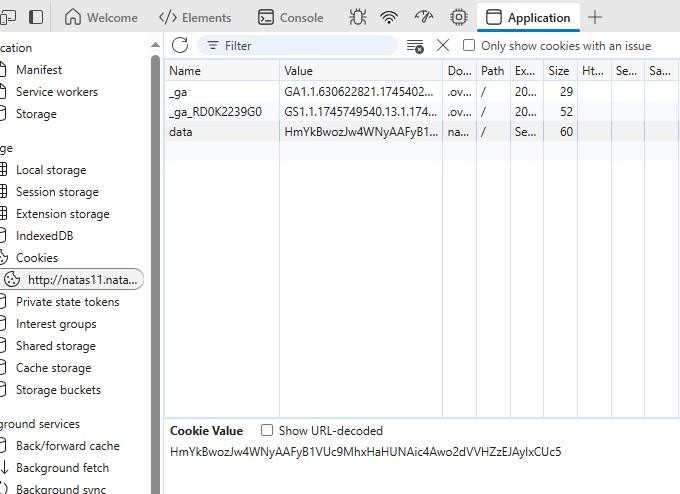
* + Sign in natas11 .



* + 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

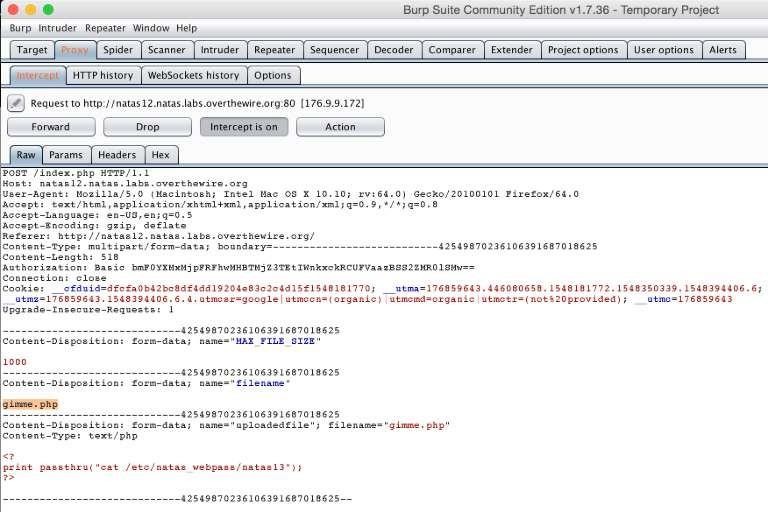


### Natas12

* + - 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

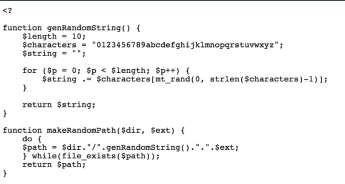


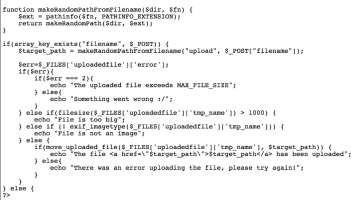
### Natas13

* + - sign in natas13

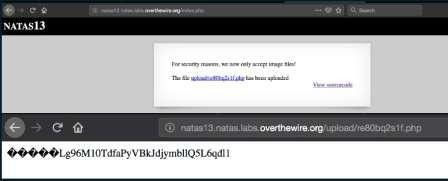


* + - View the source page and we can find that this challenge is an upgraded version of Natas 12. In this challenge, the EXIF image type is checked, so we need to add a header to our php file and make it look like a JPEG file



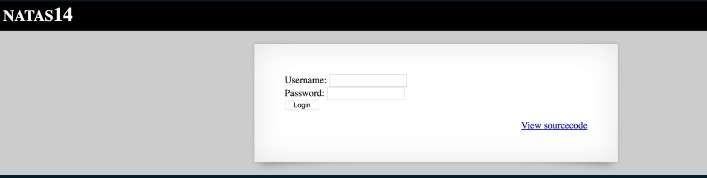


* + - We can create a php file like this:\0xFF\0xD8\0xFF\0xE0<?php echo exec('cat /etc/natas\_webpass/natas14');?>
    - Upload this php using the similar way in Natas 12 and get the password.

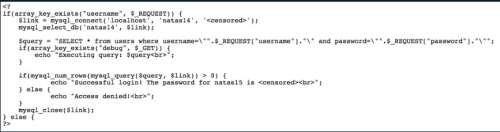


### Natas14

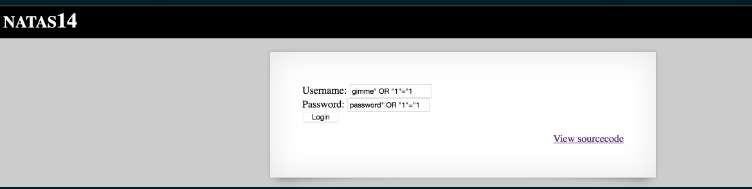
* + Sign in natas14



* + Simple SQL injection.



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



* + And we get a password for natas15

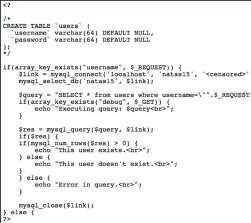


### Natas15

* + Sign in natas15



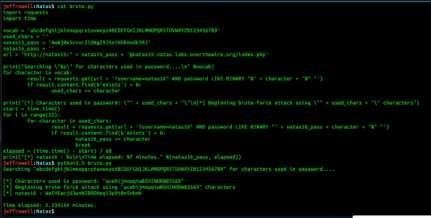
* + SQL injection



* + 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

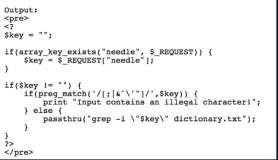
/etc/natas\_webpass/natas17) 1 1)

1.Find the common letter in the result (numbers will get no result)

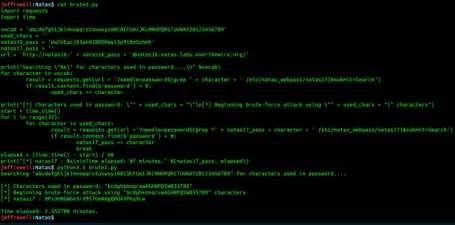
Get all numbers: a{$(($(expr substr $(cat /etc/natas\_webpass/natas17) 1 1)-6))}

* + Get the case of the letters: a{$(($(expr index $(expr substr $(grep -i Englishing dictionary.txt) 2 4) $(expr substr $(cat

/etc/natas\_webpass/natas17) 21 1))+0))}

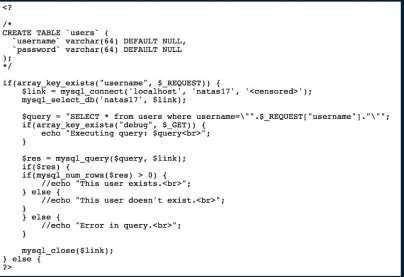


* + Final result: natas17:8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw



### Natas17

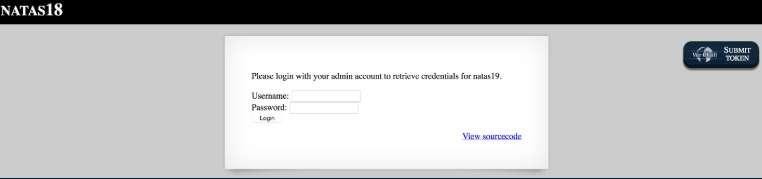
* + Sign in natas17
  + MySQL Injection Code: N0tAC0ntent” union select 1,if ((select password from users where CHAR\_LENGTH(password)=32 limit 0,1) like binary “%”,sleep(2),1)



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

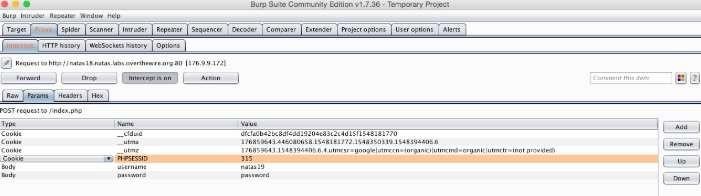


### Natas18

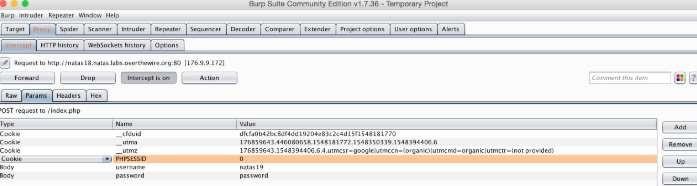
****

* + 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,

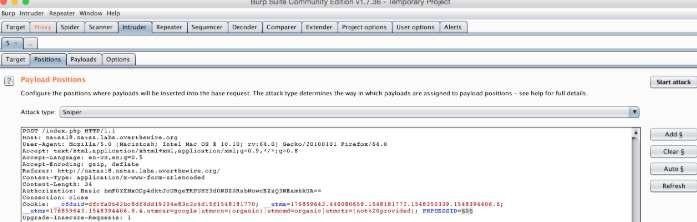
> my\_session\_start() checks for PHPSESSID in the cookie, > Burp Suite is used to intercept and analyze the PHPSESSID



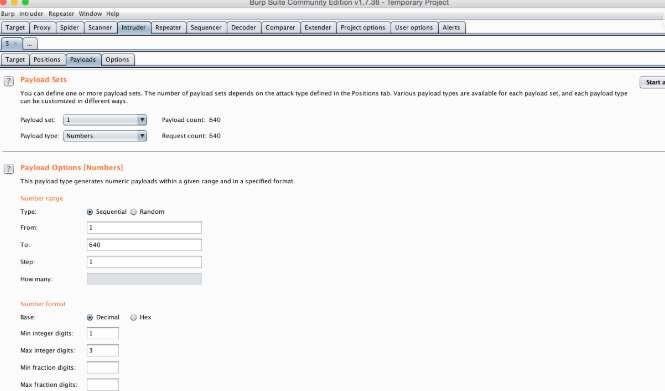
* + The **PHPSESSID** is used as the cookie, so there are 640 possible session IDs to brute-force using **Burp Suite** intruder attack, targeting the ID that returns an admin response, starting with changing the **PHPSESSID** to 0

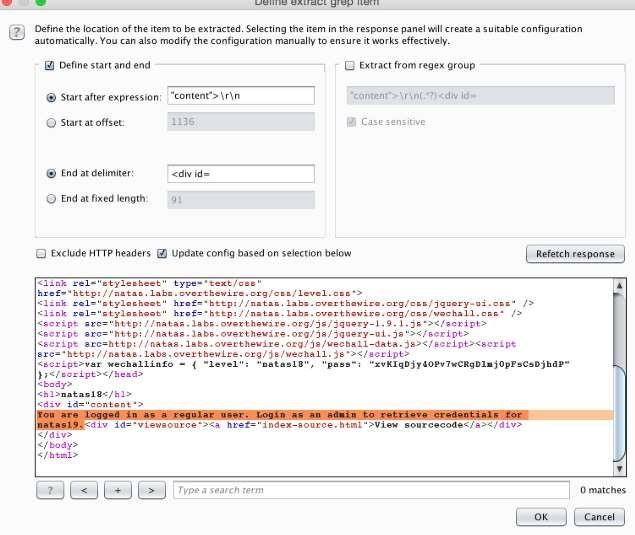


* + After clicking "send to intruder," the **Intruder** tab activates, where we select the **Sniper attack**, clear any pre-selected values, and only choose the **PHPSESSID** to target.

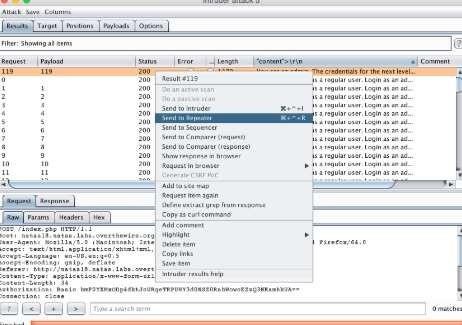


* + In the **Payload** tab, set **Payload type** to "Numbers," start 0, end 640, step 1 to brute-force **PHPSESSIDs**.

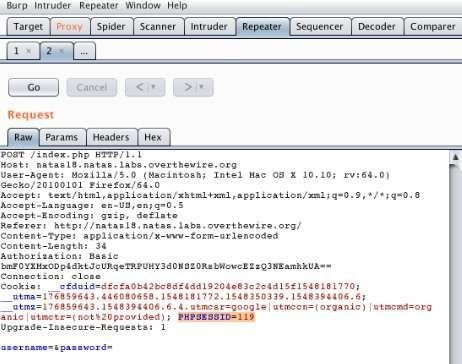




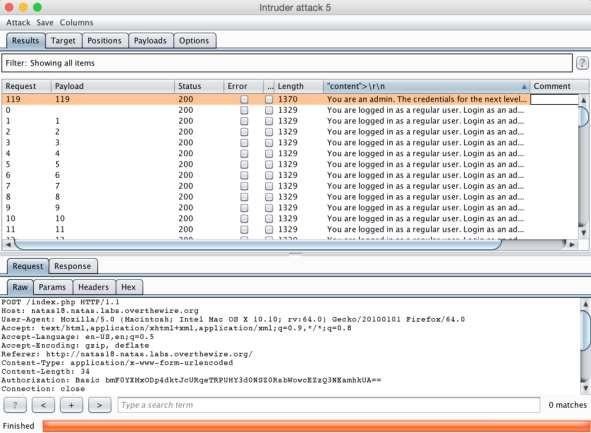
* + After setting this up, it will take about an hour to brute-force all 640

**PHPSESSIDs**, and the result screen will appear.

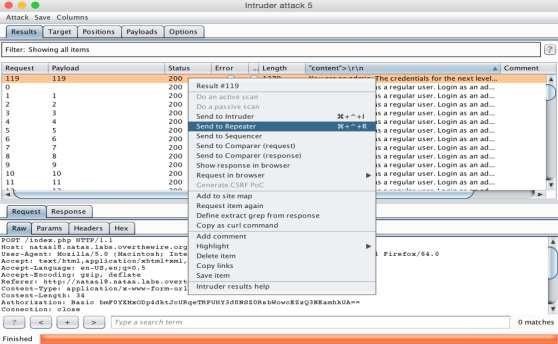
* the results by content in Intruder to find the admin PHPSESSID , right- click and send it to Repeater to replay the admin session, and pressing Go reveals the HTTP response with the next level’s password.



* 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



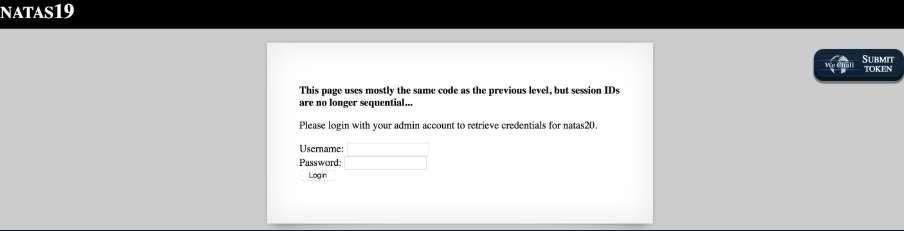
* Once found, we can right click on that row and send it to the Repeater to replay the admin log in session.



* 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

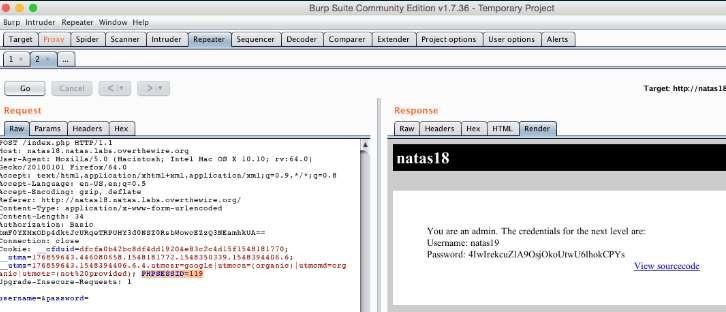


* Intercept requests in **Burp Suite**, notice **PHPSESSID** changes based on username, decode "admin" PHPSESSID to "103-admin", generate all possible **PHPSESSIDs** (0-admin to 640-admin) using a Python script, then load the hex-encoded IDs into Burp Suite and use **Intruder** to brute-force the correct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

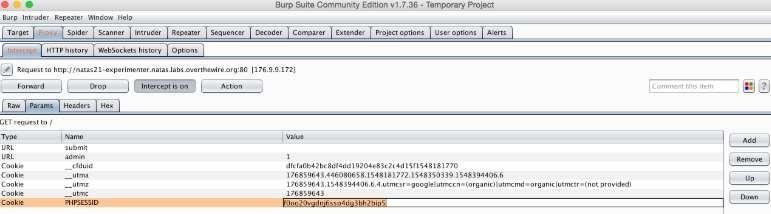


* In natas21, the source code shows a "CSS Style Experimenter" with a vulnerability: no input sanitation when setting session variables.



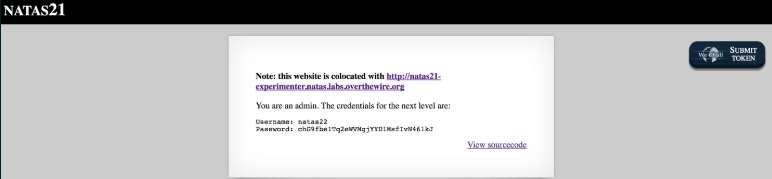
* we trigger the session to set admin=1.

Using Burp Suite, we intercept the PHPSESSID after submitting the payload, then go back to the main site: <http://natas21.natas.labs.overthewire.org/>



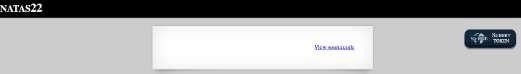
* We replace the PHPSESSID with the admin one f0oo20vgdnj6sso4dg3bh2bip5

After forwarding the request, the page confirms admin access and shows the **natas22 password**

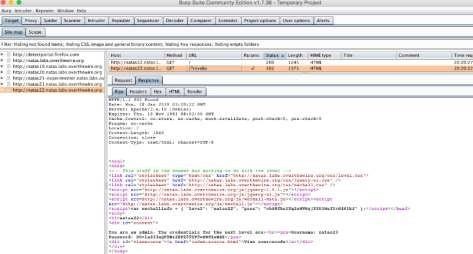
****

### 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.



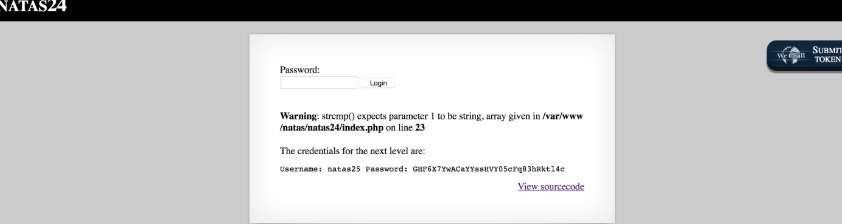
### Natas 24:

* 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.



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.

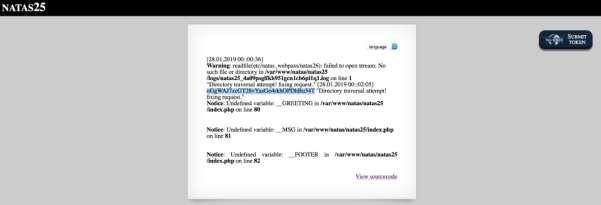
1. There is another problem when a directory traversal is detected, it calls “logRequest()” and logs the error message. We can take advantage of the $\_SERVER\*“HTTP\_USER\_AGENT”+ value by replacing it with a call to readfile(“/etc/natas\_webpass/natas26) using Burp

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

1. 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!



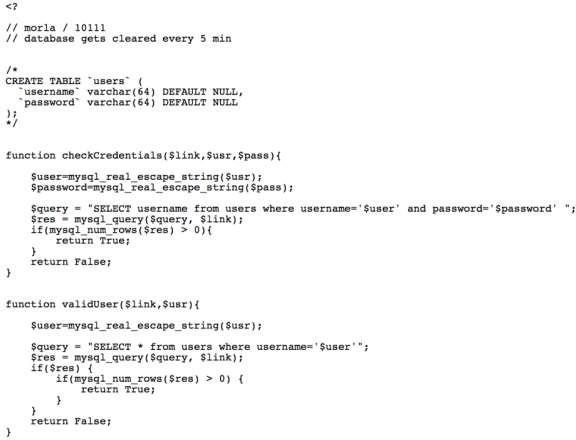
### Natas26:

****

* + 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.



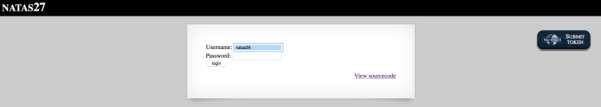
* + The “dumpData()” function makes another call to a mySQL function called “mysql\_fetch\_assoc()”.
  + Corresponds internal data pointer ahead in the database to point at the next row.

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



* + INSERT INTO users (username,password) values (‘given username’,’given password’)
  + SELECT \* from users WHERE username=’natas’
  + SELECT \* from users WHERE username=’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:

****

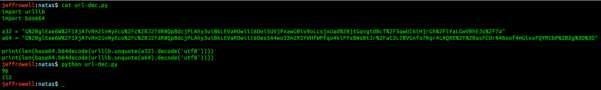
* + When I enter something like ‘natas29’ into the search box, a blank page is returned with no jokes.
  + When I enter a random letter like ‘a’, I noticed that the URL has some very long query value for the “query” key which looks like some sort of encrypted string or hash value.



* + 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.

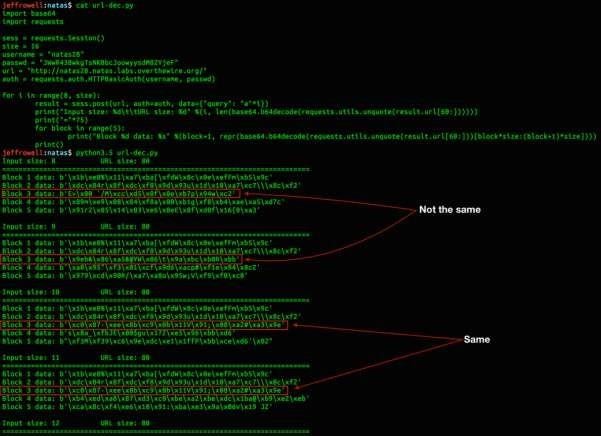


* + First piece of the input (which appears to always be the same) is encrypted, then the next portion of the input is being encrypted, so it is safe to say that there is some sort of block encryption going on behind the scenes.



* + The first part of the URL is static and will never change… The

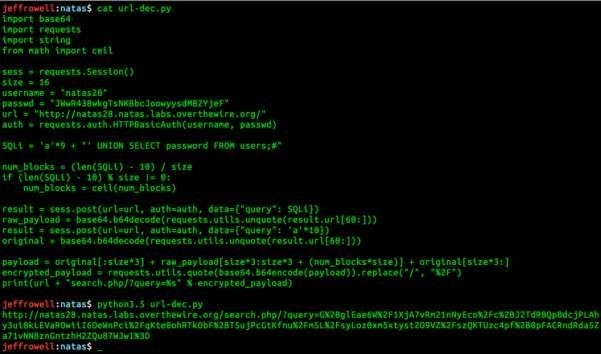
“http://…overthewire.org/?query=&#8221; is 60 characters long, so I just grab whatever is after that which will correspond to the value for the “query”.



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.



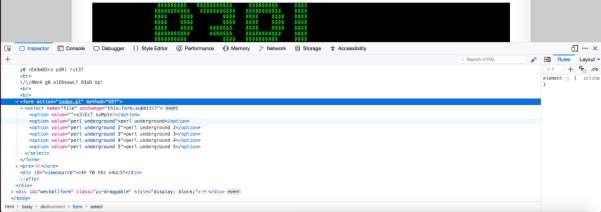
* + SELECT text FROM jokes WHERE text LIKE ‘%,query-%’
  + ‘UNION SELECT password FROM users;#



### Natas29:



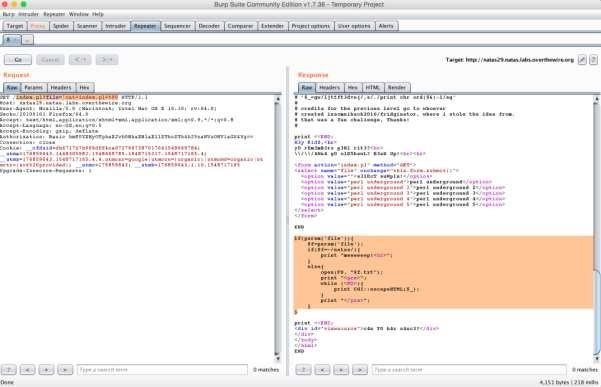
* + - Using (Command+alt+i) in Firefox allows me to still view the inspection window.



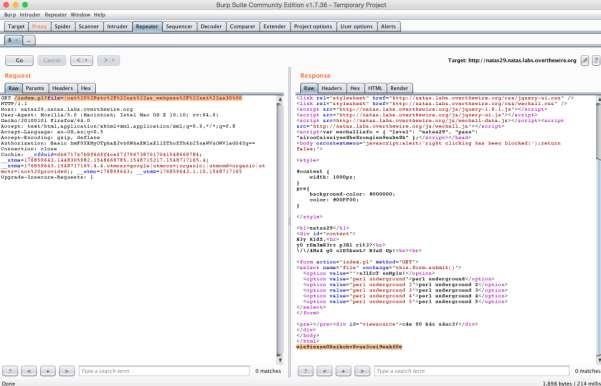
* + - <http://natas29.natas.labs.overthewire.org/index.pl>?
    - file=perl+underground



* + - * 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%22as 30%00

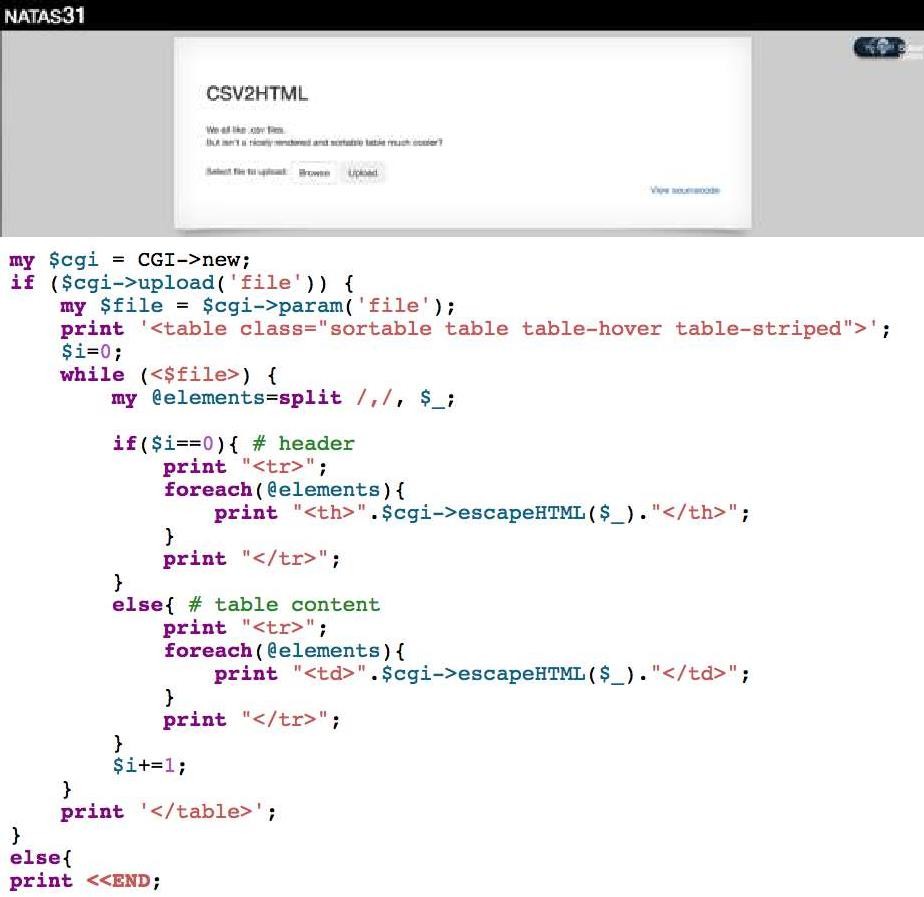


**Natas 30:**

****

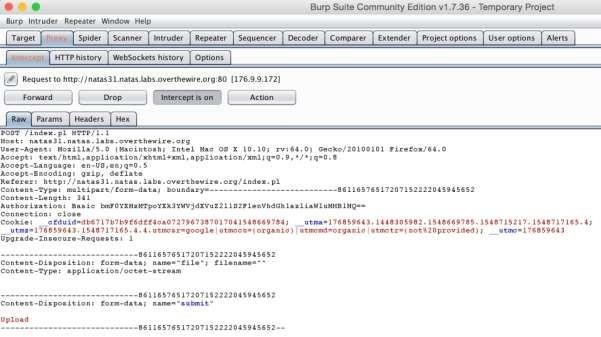
* + - * + SELECT \* FROM users WHERE username=”username” AND password=”password”
        + Use “natas31” for the username, and then an list with an SQLi attack string for the password.

**Natas 31:**

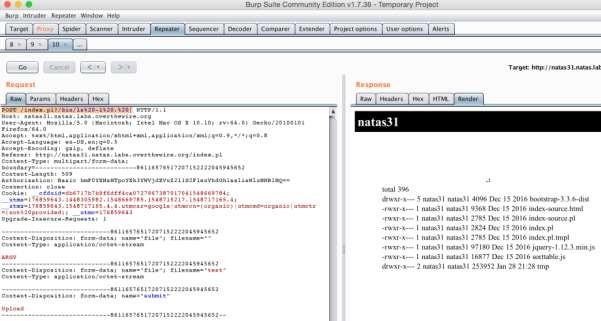
****

* + This script is relatively simple as it parses through the CSV file looking for commas, and splits the data into a tabular form.

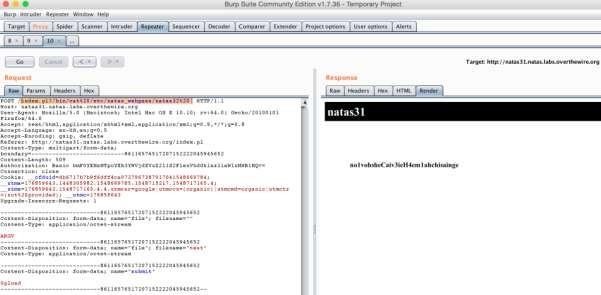
1. the line my $file = $cgi->param( 'file' ); param() will return a list of ALL the parameter values, but only the first parameter will be inserted into the file. Further, if a scalar parameter was assigned first, the $file gets assigned that scalar value rather than the value of the uploaded file descriptor. 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.



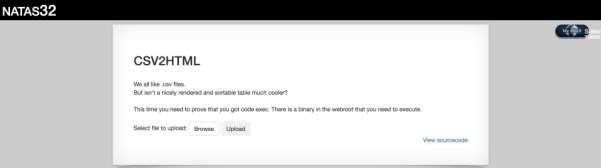
* + The next step will be to configure our exploit similar to how the exploit was configured at 23:18 here where we just need duplicate the Content-Disposition and add the string “ARGV” so that we force the <> operators to iterate through all of the values we give in ARG. Then, just need to add the command 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!

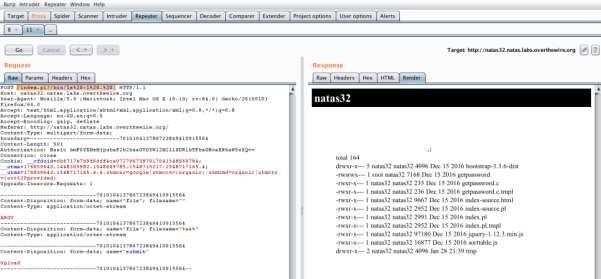


**Natas 32:**

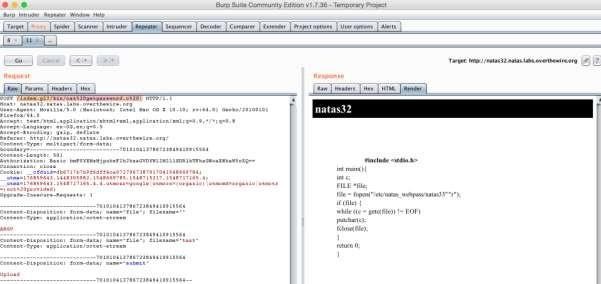
****



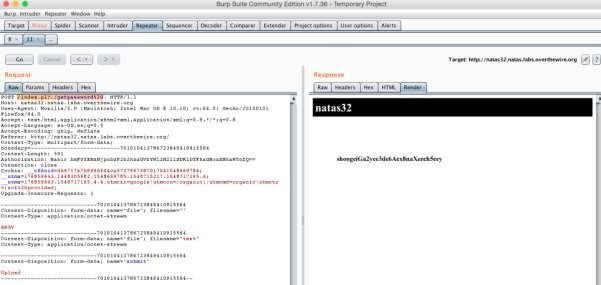
* + Send a request and intercept it using Burp to analyze if anything has changed.



* + 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!**