HackerFrogs Afterschool
Beginner Cybersecurity Skills

Web App Exploitation Basics with Natas CTF: Part 2

Class:	Web App Exploitation
Workshop Number:	AS-WEB-02
Document Version:	1.0
Special Requirements:	Completion of AS-WEB-01

Table of Contents

Introduction

Welcome to HackerFrogs Afterschool! HackerFrogs Afterschool is a a series of online workshops meant to teach cybersecurity skills and concepts to beginners who may not be familiar with the field of cybersecurity. This lesson is the second lesson for web app exploitation, which is a very prominent and in-demand specialization in cybersecurity. This lesson will cover the following learning objectives:

- Interact with the web applications using web browser software.
- Read relevant information from web page source code
- Learn about Local File Inclusion (LFI) exploitation
- Learn about OS Command Injection exploitation

Workshop Guide Format Notes

When following instructions in this document, there will be instructions to type specific commands in the command line interface (CLI) window. These commands will be displayed in red, and will have a different `font` applied to them, as well as a light grey background color. For example:

```
this is a sample command text line
```

However, if the input is meant to be put into the web browser or other non-CLI program, it will be colored red, and **bolded** but will not be in a different font nor will it have a light-grey highlight. For example:

This is a sample user input text line

If there are code snippets in this document, they look similar to the format above, except that the text will be black. For example:

```
this is a sample code snippet line
```

Part 1 – Sourcecode Analysis With Natas 6

Step 1

In our web browser, navigate to the following webpage:

http://natas6.natas.labs.overthewire.org

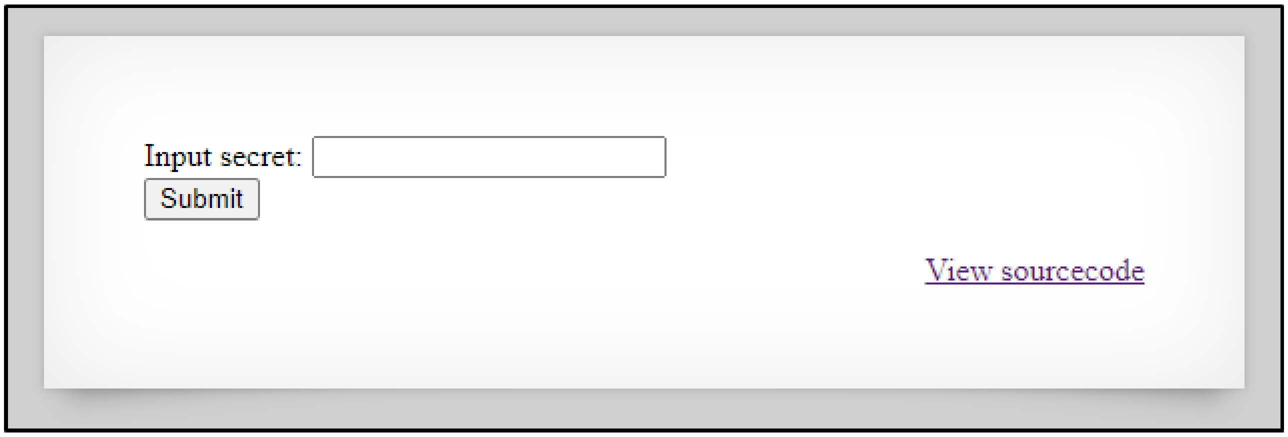
When prompted, enter the following username and password:

Username: **natas6**
Password: obtained from Natas level 5

SPECIAL PASSWORD NOTE: The Natas game servers change the passwords for all of the levels once every few months. So it's likely that the passwords provided in this document are not valid. Therefore it's important for us to keep a list of our own passwords as we work through the Natas CTF.

Step 2

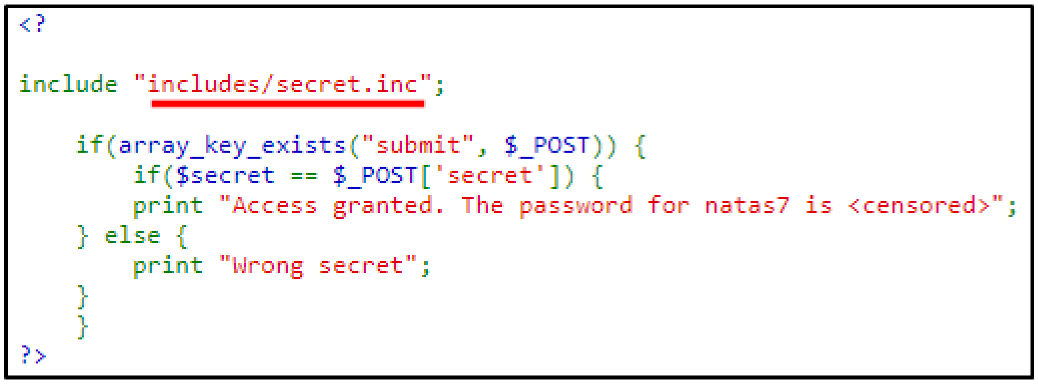
Once logged in, we see the following:



It seems like we need to supply the app with some secret value to progress. We should take a look at the sourcecode through the link.

Step 3

Click on the **View sourcecode** link, we see the function that handles the **Input secret** form. Interestingly enough it references a file at the endpoint **includes/secret.inc** (endpoint underlined in red).



```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for natas7 is <censored>";
    } else {
        print "Wrong secret";
    }
    }
?>
```

CONTEXT

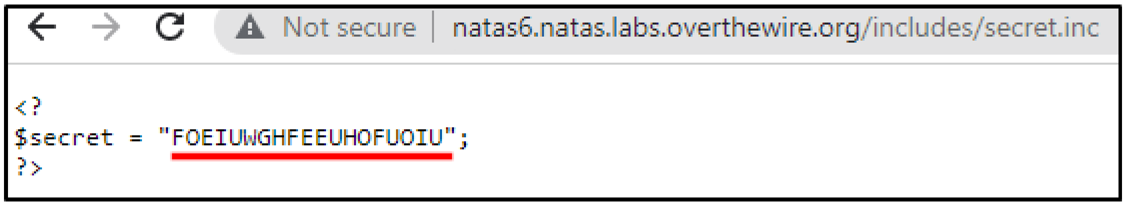
If we have access to the sourcecode of a particular program or function, (assuming that we are familiar with the language the program is written in) we can understand how the program / function works. In this case, the PHP code here takes code from another file (**secret.inc**) and if the user submits a POST request and the content of the POST request is equal to the **$secret** variable, then the password for natas7 is returned. Since the value of the $secret variable is not available in this code, we have to assume that it is contained in the secret.inc file.

Step 4

Access the secret.inc file by navigating to the following link:

http://natas6.natas.labs.overthewire.org/includes/secret.inc

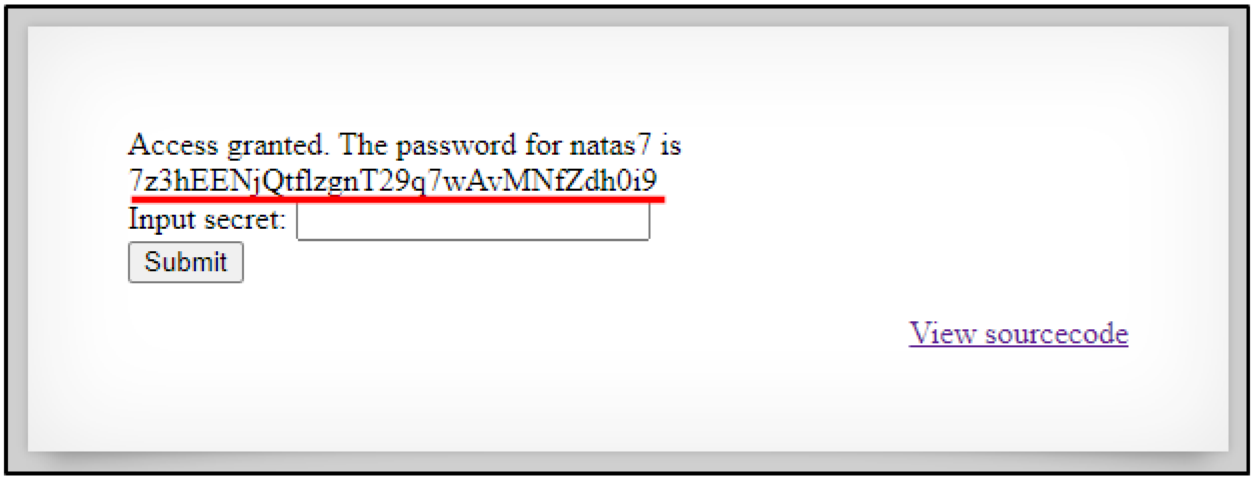
The webpage that is returned contains the value of the $secret variable (variable underlined in red)



```
<?
$secret = "FOEIUWGHFEEUHOFUOIU";
?>
```

Step 5

Copy the value of $secret (all letters between the double quotes), then return to the main natas6 webpage, paste the $secret value into the **Input secret** field and click on the **Submit** button. The password for natas7 should be in the output (password underlined in red).


Access granted. The password for natas7 is 7z3hEENjQtflzgnT29q7wAvMNfZdh0i9

Part 2 – Local File Inclusion (LFI) With Natas 7

Step 1

In the web browser, navigate to the following webpage:

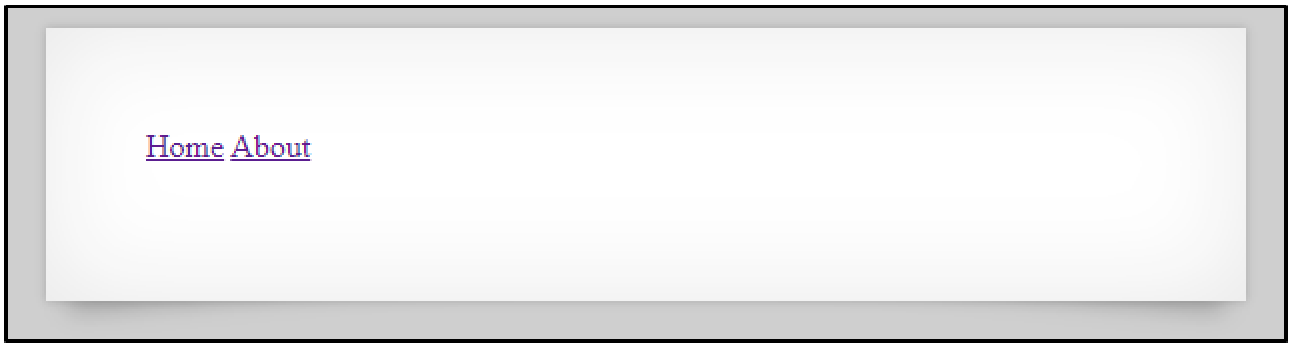
http://natas7.natas.labs.overthewire.org

When prompted, enter the following username and password:

Username: **natas7**
Password: obtained from Natas level 6

Step 2

Upon successful login, we can see the following message:



If we hover our mouse cursor over the Home and About links, at the bottom of the screen, the URLs that they will go to are displayed at the bottom of the screen, and the names of the webpages are included in the URL as the **page** parameter (**page=home** and **page=about**). This indicates that files from the web server are directly referenced by the webpage, and thus this app could be vulnerable to Local File Inclusion (LFI).

CONTEXT

We'll describe LFI in further detail at the end of this level, but in a nutshell, LFI is a vulnerability where any webserver file can be accessed through an insecure web interface.

We mentioned URL parameters in the previous step. These are variables attached to the end of URLs, and can be identified by the following example (the parameter portion of the url is colored in blue):

https://example.com/test.html?webpage_background_color=blue&webpage_font_color=red

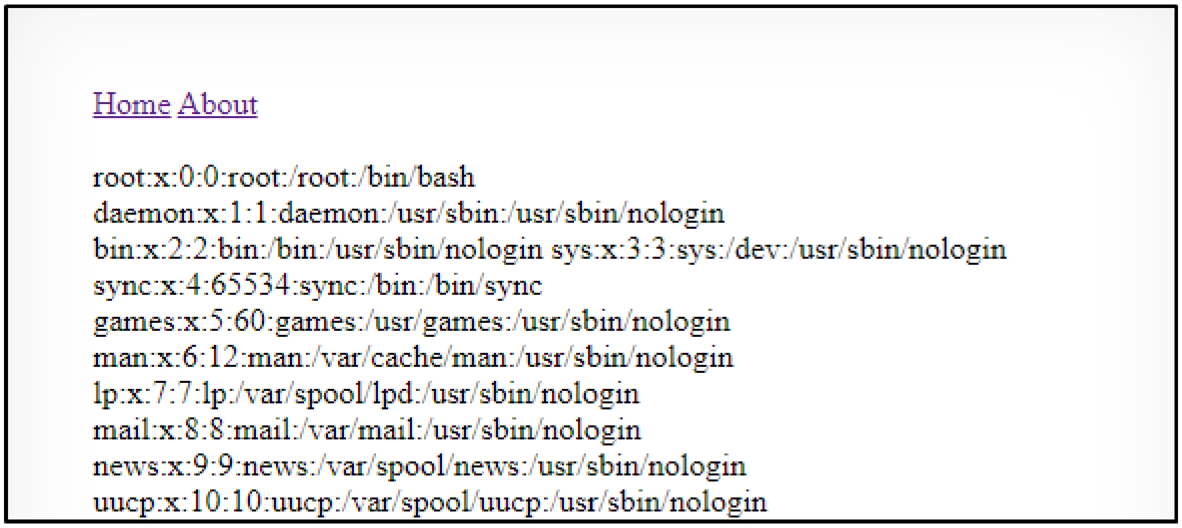
In the URL above, the webserver is changing the color of the webpage background and font color depending on parameters sent to it. The ? (question mark) starts the parameter section of the URL, followed by the parameter name (also known as the key), then the = (equals sign), then the parameter value. If there are multiple parameters in a URL, they are separated by the & (ampersand) symbol.

Step 3

We will test the application for LFI vulnerability by attempting to access the webserver's local user list. Navigate to the following URL:

http://natas7.natas.labs.overthewire.org/index.php?page=../../../../../etc/passwd

The beginning of the output should look similar to this:


Home About

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

CONTEXT

Let's break down the method we used to access the Linux user list file (**passwd**).

The URL we accessed was quite standard up until the **?** mark, which denotes the beginning of a URL parameter. The parameter name is **page**, and takes the name of a file on the webserver. The string of **../** characters is Linux syntax which instructs the filesystem to go up one level in the directory structure. The five repetitions of this should bring us to the top level of the file system. Then we access the **passwd** file in the **etc** directory. The reason why we need to go up to the top of file system is because by default, a website can only access files in its working directory (which is often several layers down from the top of the file directory)

The fact that we are able to access the passwd file is evidence that the app is vulnerable to LFI.

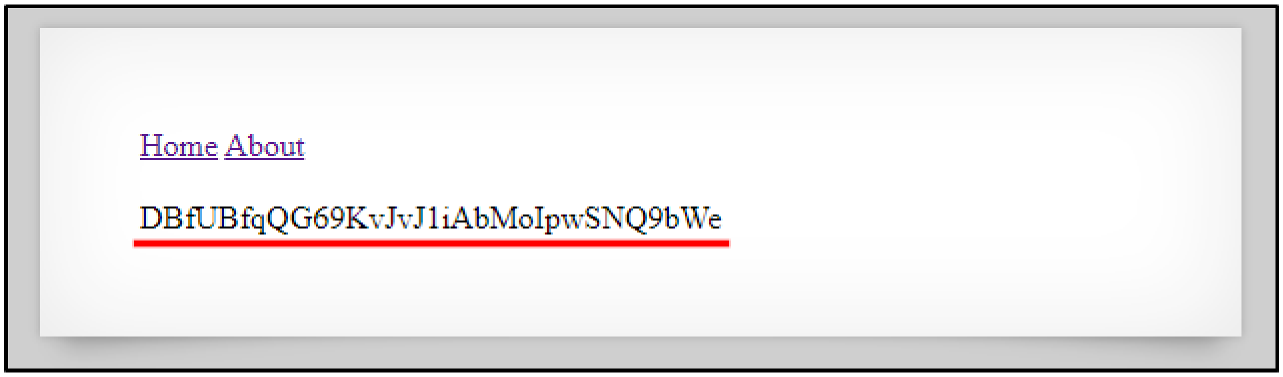
Step 4

Now that we know the app is vulnerable to LFI, let's access the password file for natas8. From the natas CTF description page, we know that each level's password is stored on the webserver in the /etc/natas_webpass/natasNUMBER directory where NUMBER is replaced with the level of the CTF in question, e.g., the password for natas8 would be located in the /etc/natas_webpass/natas8.

With that in mind, navigate to the following URL, taking advantage of the LFI vulnerability:

http://natas7.natas.labs.overthewire.org/index.php?page=../../../../../etc/natas_webpass/natas8

We should see output like the following (natas8 password underlined in red):



Home About

DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe

What is Local File Inclusion (LFI)?

LFI is a vulnerability in web apps where attackers can access any file on the webserver (not just files normally available on the website). Using LFI, attackers can gain access to sensitive webserver files, and possibly use LFI as the first step in a multi-step exploitation attack.

In order for an LFI attack to be successful, the following requirements must be met:

- The webserver must be misconfigured so that any file can be accessed via URL parameters (not just the intended ones)
- The locations of sensitive files on the webserver must be known (although many systems keep sensitive files in default locations, which are well-known). This requires identification of webserver's operating system (Windows, Linux, Mac), and the type (e.g., Ubuntu Linux, Red Hat Linux, etc)
- The webserver software user account must have read access to the sensitive files.

Part 3 – Further Sourcecode Analysis With Natas 8

Step 1

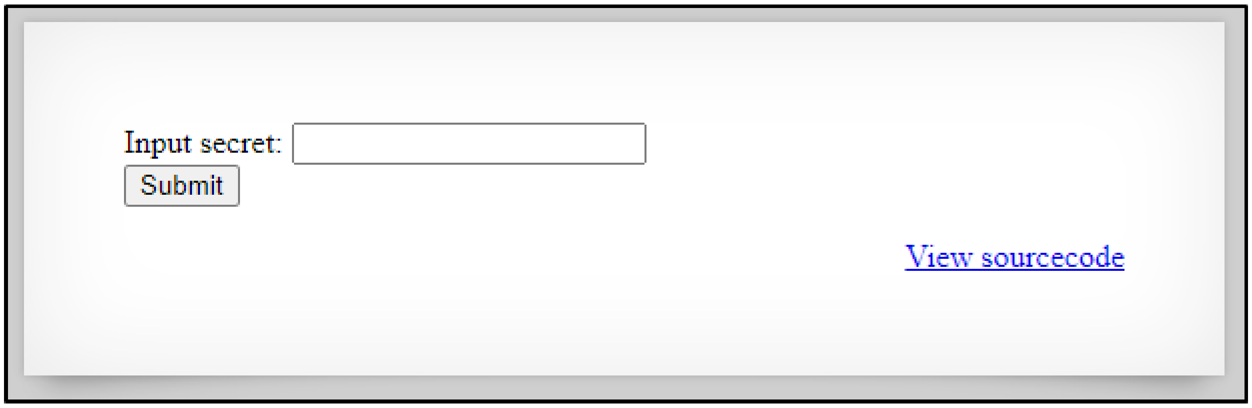
In the web browser, navigate to the following webpage:

http://natas8.natas.labs.overthewire.org

When prompted, enter the following username and password:

Username: **natas8**
Password: obtained in Natas level 7

We see the following after we login in:



It looks like we'll have to analyze the sourcecode again.

Step 2

Click on the **View sourcecode** link

Let's review the sourcecode:

```
1  <?
2
3  $encodedSecret = "3d3d516343746d4d6d6c315669563362";
4
5  function encodeSecret($secret) {
6      return bin2hex(strrev(base64_encode($secret)));
7  }
8
9  if(array_key_exists("submit", $_POST)) {
10     if(encodeSecret($_POST['secret']) == $encodedSecret) {
11     print "Access granted. The password for natas9 is <censored>";
12     } else {
13     print "Wrong secret";
14     }
15 }
16 ?>
```

The variable we want to pay attention to is **$encodedSecret** (line 3). Its value is a result of a few different operations performed on the **$secret** variable (which we don't have access to). However, we can reverse engineer the $secret variable by performing reverse operations on the value of the $encodedSecret variable.

These operations are all on line 6, and the last operation on the $secret variable is a binary to hexidecimal operation (`bin2hex`), then a string reverse (`strrev`), then a base64 encode (`base64_encode`). So to return the original $secret variable, we have to take the value of $encodedSecret, then:

1) hexidecimal to binary
2) string reverse
3) base64 decode

We can use an online PHP console to perform these operations.

Step 3

Navigate to the following URL:

https://onlinephp.io/

Step 4

On the Natas sourcecode page, copy the entirety of the PHP code, from the **<?** bracket until the **?>** bracket, then highlight all of code on the onlinephp webpage, then paste over it with the Natas PHP code. DO NOT copy the code from the previous page of this document, as it will not work unless the line numbers are deleted manually afterwards.

Step 5

On line 2 of the code, modify name of the **$encodedSecret** variable, changing it to $secret, then on line 4, change the name of the **encodeSecret** function to **decodeSecret**. Then highlight and delete all of the code from line 9 until line 15.

Step 6

Now we're going to reverse all of the operations on line 6 of the code. Replace the content of line 6 with the following code:

```
    return base64_decode(strrev(hex2bin($secret)));
```

Step 7

Lastly, we'll include a print instruction so we can output the result of the **decodeSecret** function on line 8:

```
print(decodeSecret($secret));
```

Step 8

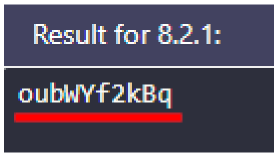
Review the code. It should look like the following:

```
<?

$secret = "3d3d516343746d4d6d6c315669563362";

function decodeSecret($secret) {
    return base64_decode(strrev(hex2bin($secret)));
}
print(decodeSecret($secret));
?>
```

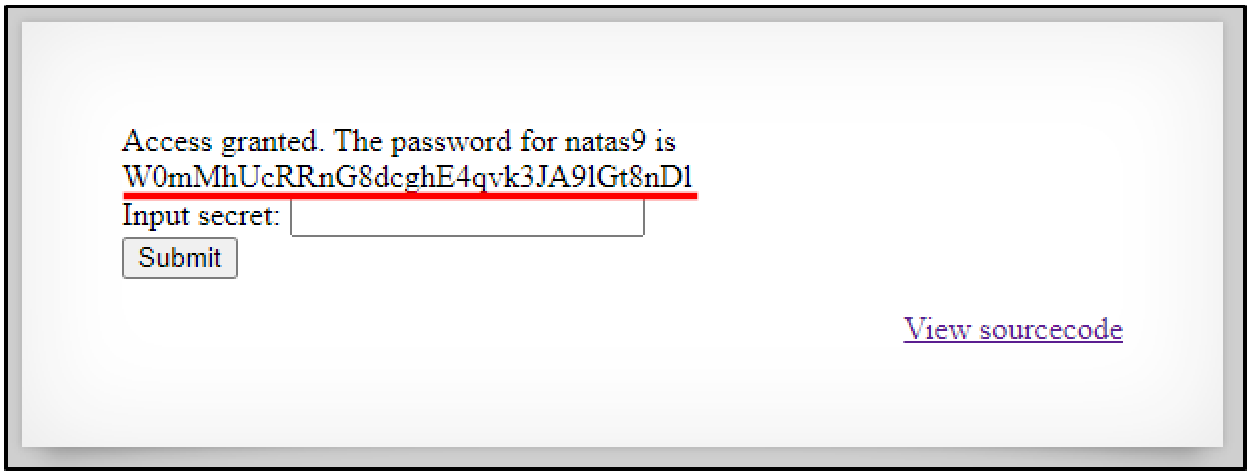
Click on the white **Execute Code** button below the code window, and the secret value we're looking for should appear in the **Result** window (secret underlined in red).



```
oubWYf2kBq
```

Step 5

Copy the value derived in the last step into the main natas8 webpage, then click the **Submit** button. The output should look like the following (natas9 password underlined in red)



Access granted. The password for natas9 is W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl

Part 4 – OS Command Injection With Natas 9

Step 1

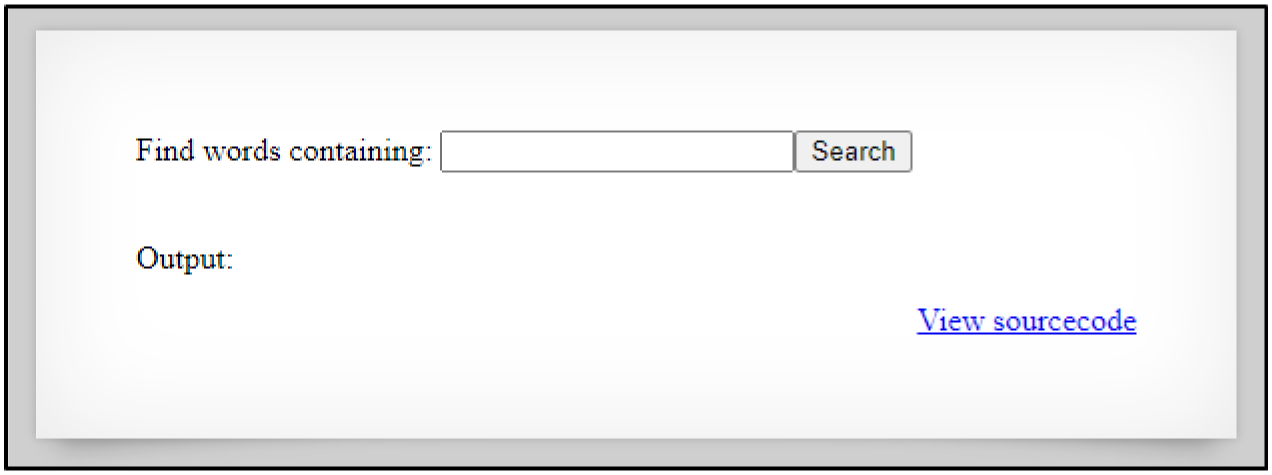
In the web browser, navigate to the following webpage:

http://natas9.natas.labs.overthewire.org

When prompted, enter the following username and password:

Username: **natas9**
Password: obtained from Natas level 8

On login, we see the following webpage:



We have access to the sourcecode again, so we should look at that.

Step 2

Click on the View sourcecode link. Let's take a look at the code for the web app search function:

```
<?
$key = "";

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

if($key != "") {
    passthru("grep -i $key dictionary.txt");
}
?>
```

This is more PHP code, and if the user input is not empty (line 8), then the PHP **passthru** command is used to send the webserver an operating system (OS) command (line 9). In this case, the OS command being passed is a search with the Linux **grep** command, with the contents of a file called dictionary.txt as the file being searched.

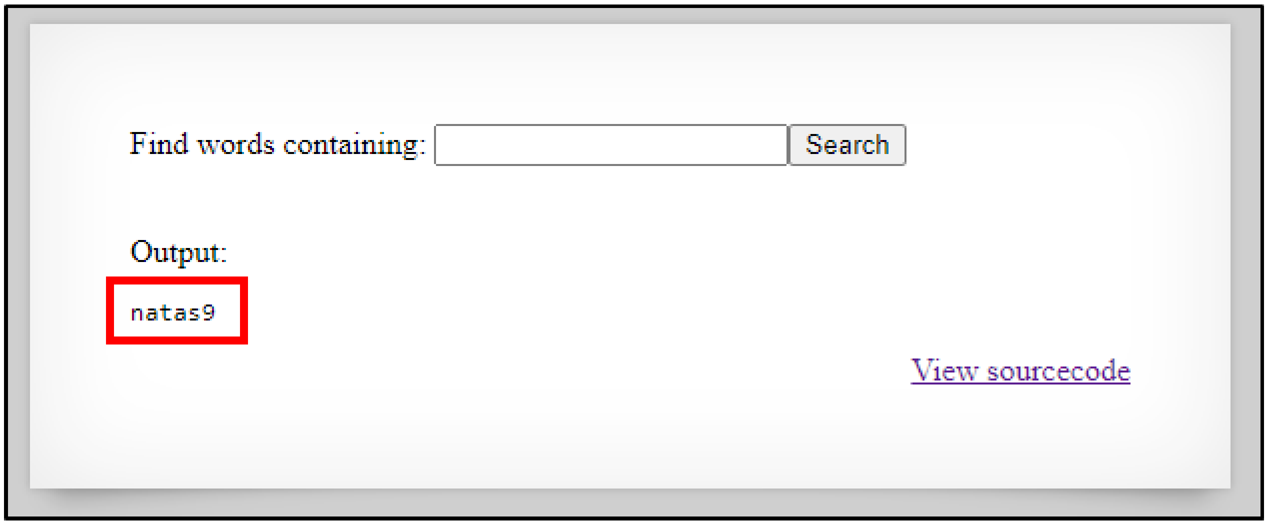
But what happens if we add an extra Linux command to the user input?

Step 3

On the main natas9 page, supply the following input to the search field, then click on the **Search** button:

; whoami #

We should get the following output:



```
natas9
```

We've managed to get the username of the local webserver account, which confirms that the webpage is vulnerable to OS Command Injection.

CONTEXT

Let's break down the Linux command we sent to the webpage:

First, the semicolon (**;**) terminates the initial command (which was "**grep -i**").
Second, the **whoami** command queries the system for the current user account name.
Finally, the hash sign (#) comments out the rest of the command so it doesn't interfere with our output.

So the original OS command with our injected input looks like this (injected input in red):

grep -i ; whoami #dictionary.txt

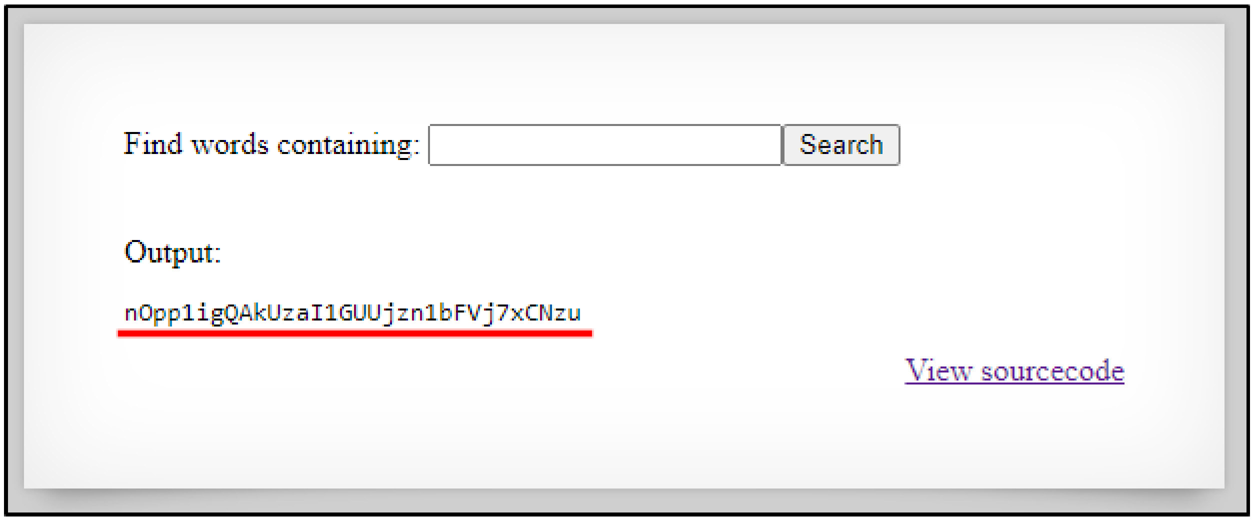
Our OS command injection ensures that only the **whoami** command is returned by the webpage.

Step 4

Use OS Command Injection to retrieve the password for natas10. Input the following command into the natas9 webpage:

; cat /etc/natas_webpass/natas10 #

The output should look like this (natas10 password underlined in red):



Output:

```
nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu
```

Before we move on to our last exercise, let's review what OS Command Injection is.

What is OS Command Injection

OS Command Injection is a web application vulnerability where users can execute arbitrary (any) operating system (OS) commands directly from a webpage. If users can only use the webpage to execute specific OS commands, then it is not considered to be vulnerable to OS Command Injection. OS Command Injection can be used by malicious attackers to access sensitive data on the webserver or even result in complete compromise of the webserver, which can then be used as a foothold system to attack other systems on the webserver's network.

Now let's proceed to our final exercise in this lesson!

Part 5 – Further OS Command Injection With Natas 10

Step 1

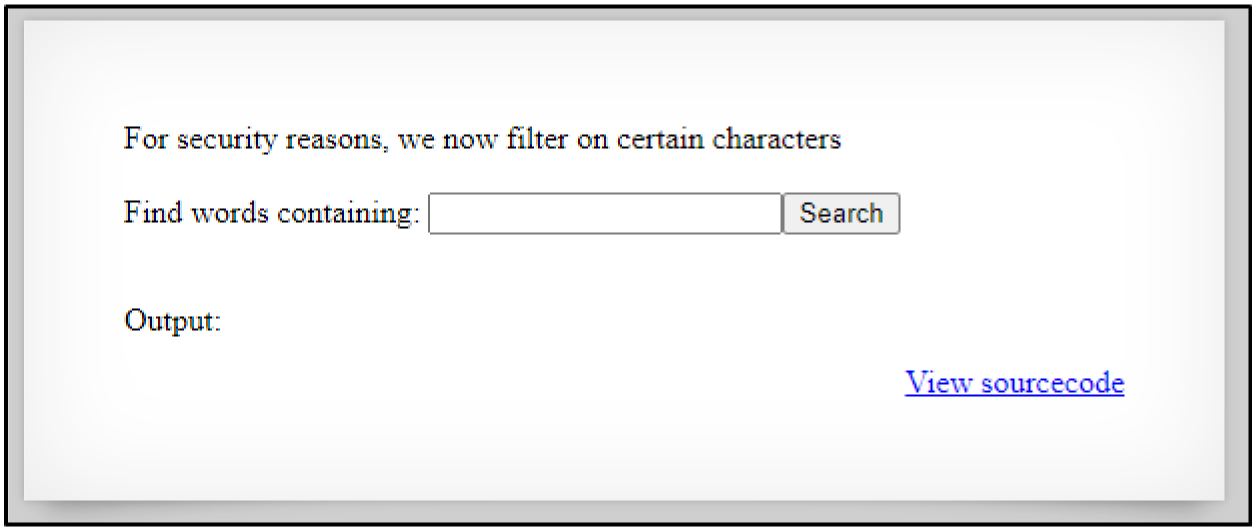
In the web browser, navigate to the following webpage:

http://natas10.natas.labs.overthewire.org

When prompted, enter the following username and password:

Username: **natas10**
Password: obtained from Natas level 9

After login, we see the following on the landing page:



For security reasons, we now filter on certain characters

Find words containing:

Output:

As usual, we should check the page source.

Step 2

Click on the **View sourcecode** link:

We will see that the code is exactly the same as the previous level, with the exception of these two lines:

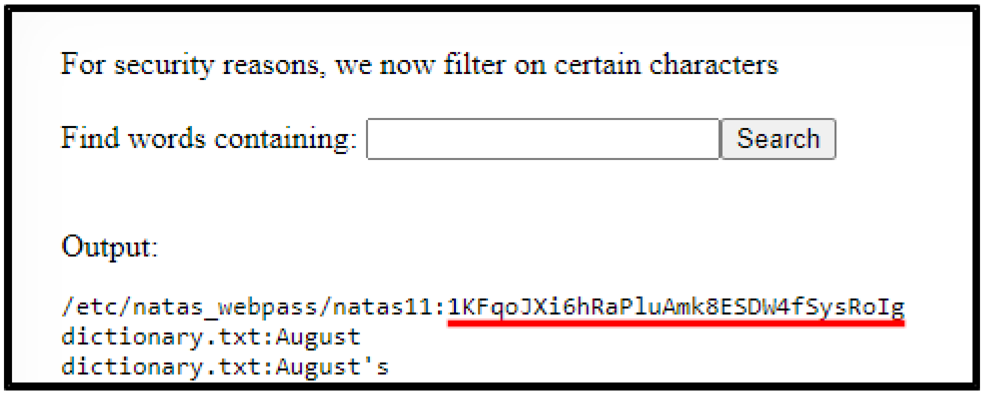
```
if(preg_match('/[;|&]/',$key)) {
    print "Input contains an illegal character!";
```

One of the special characters we used for the last OS Command Injection (the semicolon) has been filtered, so we'll have to find a different way to access the webserver for the natas11 password. Fortunately we can modify our OS Command Injection to bypass the filters in place.

Step 3

Use OS Command Injection to retrieve the password for natas11. Input the following command into the natas10 webpage search field, then click the **Search** button:

u /etc/natas_webpass/natas11


For security reasons, we now filter on certain characters

Find words containing: Search

Output:

```
/etc/natas_webpass/natas11:1KFqoJXi6hRaPluAmk8ESDW4fSysRoIg
dictionary.txt:August
dictionary.txt:August's
```

Here, we're using the web app as intended, searching for a regular string (the letter u), but the inclusion of another file (/etc/natas_webpass/natas11) means that both files will be searched for any lines containing the search query. If we take the command in its entirety, it looks like this (our user input is in red):

grep -i u /etc/natas_webpass/natas11 dictionary.txt

The grep command can be applied to multiple files in the same execution, as long as each file is separated by a space. If, the output doesn't output the Natas 11 password, try running the command again with different letters until the password appears.

Summary

While working through the Natas CTF levels 6 to 10, we learned and used our understanding of the following topics to solve various challenges.

In Natas level 6, we learned about sourcecode analysis, which is useful for discovering potential security flaws in code.

In Natas level 7, we learned about Local File Inclusion, a web app security vulnerability which allows users to potentially read any file present on the webserver.

In Natas level 8, we further explorer sourcecode analysis, allowing us to reverse engineer paswords based on the code presented to us.

In Natas level 9, we learned about OS Command Injection, an exploit where attackers can input arbitrary commands to the webserver itself through a web application, which can result in complete compromise of the webserver.

In Natas level 10, we expanded on our OS Command Injection skills, using a input filter bypass technique to perform an OS Command Injection attack.

In the next workshop, we'll be looking at the most common language used in database management (SQL), which is widely integrated into web apps which store data (which is most of them).

Extra Credit

The following exercises cover topics that were included in this workshop and are provided for an extra challenge:

picoCTF - dont-use-client-side
https://play.picoctf.org/practice/challenge/66

picoCTF – Forbidden Paths
https://play.picoctf.org/practice/challenge/270

PortSwigger – OS command injection, simple case
https://portswigger.net/web-security/os-command-injection/lab-simple

NOTE: Challenges hosted at the picoctf.org and portswigger.net websites require valid user accounts for access. Neither user account requires payment, fortunately.

Extra Research

The following articles will help us further understand some of the concepts covered in this workshop:

More local file inclusion (called "file path traversal" here):
https://portswigger.net/web-security/file-path-traversal

Portswigger's Article on OS Command Injection:
https://portswigger.net/web-security/os-command-injection

Until next time, HackerFrogs!