Natas – Stage by stage explanation

Level 0 -> 1

After initially logging in to natas 0, by going to address `http://natas0.natas.labs.overthewire.org` and entering natas0 for both fields, we are greeted by a page which says "You can find the password for the next level on this page". A quick right click -> inspect element reveals the password as an HTML comment.

The password for natas 1 is `gtVrDuiDfck831PqWsLEZy5gyDz1clto`

Level 1 -> 2

Again, we are told that the password is somewhere on the page, but right clicking has been blocked this time. No bother; we can simply hit f12 to pull up the developer pane. The password is stored as an HTML comment again.

The password for natas 2 is `ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi`

Level 2 -> 3

Here, we are told that there is nothing on this page. A quick inspect element is, at first, promising - it reveals a .png file. However, this file seems to just be a single white pixel, and is of no use. We can see that it's stored in the /files folder - navigating to this (by adding /files/ to the end of the url) we find a file called users.txt. This contains the password for natas3.

The password for natas 3 is `sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14`

Level 3 -> 4

As before, we are told that there is nothing on this page. A quick inspect element reveals to us an HTML comment, `<!-- No more information leaks!! Not even Google will find it this time... -->`. This suggests that this might have something to do with the robots.txt file; these exist in websites to tell a search engine to ignore certain pages. Sure enough, navigating to this file, we find out that natas3 wants google to ignore the file `/s3cr3t/`. Navigating here (again, via URL) we are greeted once more with the users.txt file.

The password for natas4 is `Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ`

Level 4 -> 5

Arriving on the next page, we are greeted with `Access disallowed. You are visiting from "" while authorized users should come only from "http://natas5.natas.labs.overthewire.org/"`

Somehow, we need to convince the webpage that we came from natas5. This can be done with the funky program BurpSuite – this comes installed with kali linux, and otherwise can be downloaded pretty easily. Once installed, it needs to be configured with your browser - this can be done pretty easily; you just need to add it as a proxy. Once this is done, navigate to the proxy tab on burp, turn intercept on and then refresh the page. The details of the HTTP request we just sent should appear. The `Referer` attribute should be set to natas4 - if we simply change this to natas5 and then forward the request then we all set:). Doing so gives us the password.

The password for natas 5 is `iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq`

Level 5 -> 6

On arrival, we are told that access wasn't allowed and we're not allowed in. Checking our cookies (this can be done with either a browser extension or with Burp Suite), we find a conspicuously named 'loggedin'. This is currently set to 0. Changing it to 1 and refreshing the page, the text changes to say we are now logged in. This also gives us the password.

The password for natas 6 is `aGoY4q2Dc6MgDq4oL4YtoKtyAg9PeHa1`

Level 6 -> 7

Here, we are greeted with a form that asks us to input a secret. We also have a link that we can follow to view the source code. Following this, we see a block of PHP – this is code that wouldn't show up when inspecting page source in the normal way. The block of code that we see is as follows:

```
<?

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";
        }
    }
```

```
?>
```

This compares the value of the variable `$secret` to what the user enters. Unfortunately, we don't know the value of secret. However, we can notice that the code starts off with `includes/secret.inc`. By adding this to our url, we can see the contents of this file and, crucially, the value of secret. Entering this to the form gives us the password we've been hunting for.

The password for natas 7 is `7z3hEENjQtflzgnT29q7wAvMNfZdh0i9`

Level 7 -> 8

On accessing this page, we see links to `home` and `about`. On clicking these, we can observe that the URL changes - after the URL, we have `index.php?page=<page>`. We also know that all the passwords for natas are stored in `/etc/natas_webpass/natas<n>` where n is the user (we are also reminded of this as an HTML comment). Simply putting this address into the parameter serves up the password.

The password for natas 8 is `DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe`

Level 8 -> 9

Another PHP form here, checking the source code we see the following script:

```
<?

$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";
    }
}
?>
```

From this code, we can see that the script applies an encryption algorithm to the value we enter, and then compares this to the `$encodedsecret` variable. Therefore, applying the `encodeSecret` algorithm to encoded secret backwards will give us the value we need to enter.

This can either be done in the terminal, or using online tools - it doesn't matter. All we need to do is take it from hex to binary, reverse this then apply a base64 decode. If you wish to use a terminal, the following bash command will do it all in one step:

```
echo "3d3d516343746d4d6d6c315669563362" | xxd -r -p - | rev |
base64 -d
```

If we then put the output of this into the form, we get the password back.

The password for natas 9 is `W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl`

Level 9 -> 10

Here, we are again greeted by a form. It asks us to submit a search term, and offers to return words relating to this. As before, we are able to view the source code by following a link. The PHP of this is as follows:

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

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

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

The most interesting line here is `passthru("grep -i $key dictionary.txt")` – this tells us that the script is passing out input directly into the command line. As we will be aware, it is possible to execute multiple commands on the same line with linux, by using the `&` operator. So, entering `& cat /etc/natas_webpass/natas10` into the key field should give us the password - and, indeed, does.

The password for natas 10 is `nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu`

Level 10 -> 11

On reaching this page, we are met with an almost identical screen to last time, except now 'For security reasons, we now filter on certain characters'. Checking the source code, we see the following line:

```
if(preg_match('/[;|&]/',$key)) {
```

This tells us that the system isn't allowing the usual operators used to join linux commands together. Is there another way of joining commands? One would think so, but a hunt through various forums bore no fruits. However, we ARE still being allowed to access the command line directly, so maybe we could exploit `grep`. The syntax for this command is . Let's try, then, putting in `. /etc/natas_webpass/natas11` – this should match anything in the file `/etc/natas_webpass/natas11`, and print it to the screen.

This worked, and revealed the password to be `U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK`

Level 11 -> 12