#Sudo_root
#Zakizak

MSSP CTF

WEB 150 - Safe Hashing

We gave us a link and we asked us to get the secret under it.

The link given was : https://web-piratuer.c9users.io/web5/

Opening the page https://web-piratuer.c9users.io/web5/index.php~ shows this line of code

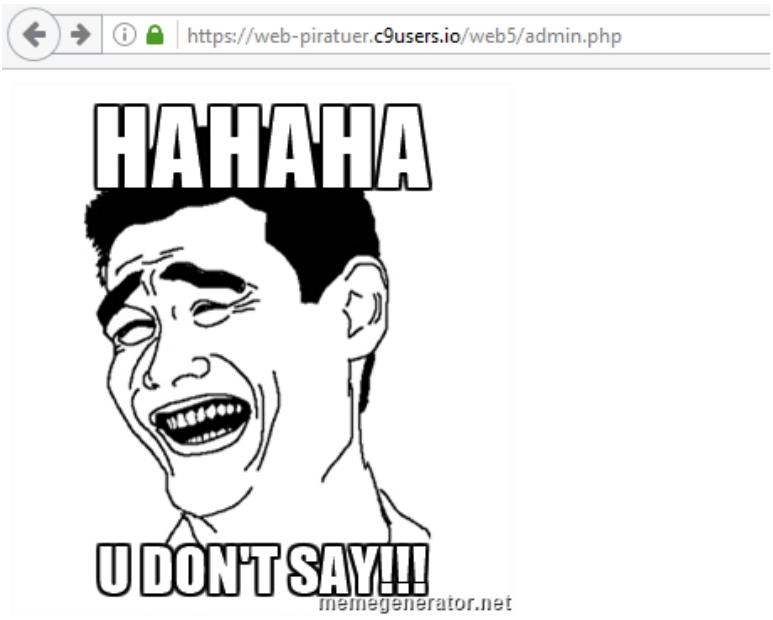
'; } else { readfile(basename($_GET['useme'])); } } ?>

From the line, above, we can see that readfile function is used and parameter 'useme' define which page should be loaded. Let's add this parameter to the end of our URL and try to load the page 'index.php' trough this manner :

https://web-piratuer.c9users.io/web5/?useme=index.php

Openning the index page this way, return the same result as the page " index.php~" loaded natively before.

Now, after a few tries of pages suceptibles to exist, we found that a page named 'admin.php' exist. Unfortunately, loading this page return us a troll image which seems to tell us that we can't load it as easily.



Showing the source code of this page reveals interesting things. There is a huge party of the source code of the page ‘admin.php’, Here it is :

```
<!--?php #Serious things begin now good luck :).
require_once("config.php"); $passme = false; if
(isset($_COOKIE["is_admin"])) { $pass = $_COOKIE["is_admin"];
$passme = unserialize($pass); $signature = $_COOKIE["mac"]; if
($signature !== hash("sha512", $SHAREDSEC . strrev($pass)))
$passme = false; } else { $passme = false; $serial =
serialize($passme); setcookie("is_admin", $serial);
setcookie("mac", hash("sha512", $SHAREDSEC . strrev($serial)));
} if ($passme) { echo "Congrats!! ".$FLAG; } else echo '<img
src="troll.jpg" alt=""-->
```

One line captivante in this source code is :

if ($signature !== hash("sha512", $SHAREDSEC . strrev($pass)))

This line is the only one which could let the variable $passme having another value than false in order to fill the last condition which is :

if ($passme) { echo "Congrats!! ".$FLAG; }

And as a consequence, getting the flag we search for.

Now we can see that there is a hash extension vulnerablity. It means that we can add arbitrary data to the end of the string, and generate a new authentication token for it.

When we visit the admin page natively for the first time, this code is executed:

```
$passme = false;
$serial = serialize($passme);
setcookie("is_admin", $serial);
setcookie("mac", hash("sha512", $SHAREDSEC . strrev($serial)));
```

From that, we see that our cookie is a serialized PHP datatype, in this case simply 'false'. The mac token is generated by prepending a secret to the reversed version of the authentication string .

Those two cookies are set. Then later, when you return, the cookies are sent back and validated:

```
if (isset($_COOKIE["is_admin"]))

{

$pass = $_COOKIE["is_admin"];

$passme = unserialize($pass);

$signature = $_COOKIE["mac"];

if ($signature !== hash("sha512", $SHAREDSEC . strrev($pass)))
$passme = false;

}
```

By looking for created cookies, I find out that my cookies for this website are:

is_admin=b:0;

mac=1505c60315b49ac1403d9cc0e2d02a435f3550b61cbf95000de3d19746b65545b361516775355aaa8fa57bf49f0a6a4e2c76667fa1ee38da5ae1681a5ccff2d

The value "b:0;" refer to boolean zero (false). The true value will be "b:1;". That's the value I want to append.

The only constraint is that we don't know the length of $SHAREDSEC, which is unfortunately important for this attack. So we have to brute-force it.

For doing that, we'll use a tool called hash_extender. However, we have to modify it a little because this challenge requires the string to be reversed before being validated.

Basically, we just have to modify the output function under the main file 'hash_extender.c' in order to reverse the string as said.

Then, we run hash_extender like this:

$ ./hash_extender --data ';0:b' -s 1505c60315b49ac1403d9cc0e2d02a435f3550b61cbf95000de3d19746b65545b361516775355aaa8fa57bf49f0a6a4e2c76667fa1ee38da5ae1681a5ccff2d9 --append ';1:b' --secret-min=1 --secret-max=32 --out-data-format=html

This tells it to try every secret length between 1 and 32 bytes. One of them should work.

Then we can automate the task entirely by mixing curl tool to the above function for each of the 32 request like this :

for i in `./hash_extender --data ';0:b' -s 1505c60315b49ac1403d9cc0e2d02a435f3550b61cbf95000de3d19746b65545b361516775355aaa8fa57bf49f0a6a4e2c76667fa1ee38da5ae1681a5ccff2d9 --append ';1:b' --secret-min=10 --secret-max=32 --out-data-format=html`; do HASH=`echo $i | sed 's/,.*//'`; DATA=`echo $i |

sed 's/.*,//'`; echo "$DATA :: $HASH"; curl -b "is_admin=$DATA;mac=$HASH" https://web-piratuer.c9users.io/web5/admin.php; echo; done

By redirecting the result into a file and by searching for the prefix of the flag mssp, we fast retrieved the whole flag as shown below:

```
%3b0%3ab%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
<img src="troll.jpg" alt="">
b%3a1%3bX%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%0C
Congrats!! mssp{Hash!nG_N0t_4lwayS_Th3_S4f3_Way_0}
Type: :: Type:
<img src="troll.jpg" alt="">
whirlpool :: whirlpool
```