Elimination Round

HackToday

Kategori Forensic

TABLE OF CONTENTS

TLDR;

Tidak lebih & tidak kurang, berikut adalah kumpulan coretan kegabutan dan eksperimental dari pengerjaan soal HackToday 2020 kategori *forensic*

Kudos to *@deomkicer* atas kesediaannya membuka akses *guest ^o^*

Arsip soal dapat diakses pada https://s.id/PenyisihanHackToday2020

Semoga bermanfaat

11 Agustus 2020

nama disamarkan

Harta-Karun

Seorang penggemar harta akhirnya insaf setelah menonton drama pengingat dosa, ia pun mengadakan sebuah sayembara untuk menemukan harta yang telah ia simpan di suatu tempat. Para peserta hanya diberikan gambar peta untuk menemukan Location dari harta tersebut. Apakah kamu yang menjadi juara?

for attachment

Diberikan sebuah berkas citra bernama `peta.png`. Kemudian menggunakan bantuan binwalk, kita peroleh informasi sebagai berikut

```
$ binwalk peta.png

DECIMAL         HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0               0x0             PNG image, 512 x 512, 8-bit/color RGBA, non-interlaced
153             0x99            Zlib compressed data, best compression
34170           0x857A          Zip archive data, at least v2.0 to extract, name: MapLv2/
34239           0x85BF          Zip archive data, at least v2.0 to extract, uncompressed size: 612,
name: MapLv2/ke.txt
34635           0x874B          Zip archive data, at least v2.0 to extract, uncompressed size: 561,
name: MapLv2/lo.txt
34980           0x88A4          Zip archive data, at least v2.0 to extract, uncompressed size: 1377,
name: MapLv2/sy.txt
35767           0x8BB7          Zip archive data, at least v2.0 to extract, uncompressed size: 693,
name: MapLv2/en.txt
36537           0x8EB9          End of Zip archive, footer length: 22
```

Tampak terlihat terdapat berkas yang disembunyikan, yang mana berisi folder `MapLv2` dengan beberapa txt file. Untuk mempermudah proses ekstraksi, kita gunakan kembali perintah binwalk dengan argumen `-e` atau `--extract`

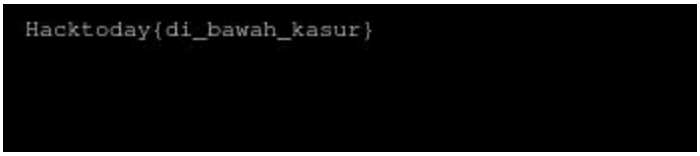
```
$ binwalk -e peta.png
$ cd _peta.png.extracted/MapLv2 && ls
en.txt  ke.txt  lo.txt  sy.txt
```

Pada tahap ini, kita menjumpai sekumpulan txt file berisikan sebuah *hex string*. Kita bisa saja berasumsi bahwa *hex string* ini merupakan hasil hexdump dari binary file yang memuat nilai dari flag.

Kendati demikian, kita perlu menebak *order* yang benar agar *binary file* yang dihasilkan dapat dikenali dan dibaca oleh sistem. Untuk itu, kita lakukan permutasi sederhana seperti yang dapat dilihat di bawah ini

```
from itertools import permutations
from binascii import unhexlify
from PIL import Image

def open_file(path):
   with open(path) as f:
       return f.read()[:-1]

filename = 'en.txt ke.txt lo.txt sy.txt'.split()
contents = dict(zip(filename, map(open_file, filename)))
perms = permutations(contents.keys(), 4)

for p in perms:
   content = ''
   for file in p:
       content += contents[file]

   with open('flag.png','wb') as flag:
       content = content.replace(' ', '')
       flag.write(unhexlify(content))

   try:
       Image.open('flag.png').show()
       break
   except IOError as e:
       continue
```

```
$ python3 solve.py
lo.txt ke.txt sy.txt en.txt
```



Flag:hacktoday{di_bawah_kasur}

Daun Singkong

`tanam-tanam ubi tak perlu dibajak.`

`for` `attachment`

Diberikan berkas `zip` bernama `daunsingkong.zip`. Setelah melakukan ekstraksi, ditemukan beberapa berkas sebagai berikut

```
$ unzip daunsingkong.zip
$ tree -a
.
|-- README.md
|-- __MACOSX
|   `-- daunsingkong
|       `-- ._.DS_Store
|-- daunsingkong
|   |-- .DS_Store
|   |-- .bash_history
|   `-- flag.7z
`-- daunsingkong.zip

3 directories, 6 files
```

Berdasarkan bagan di atas, dapat diketahui bahwa terdapat berkas flag.7z yang nampaknya dienkripsi menggunakan suatu kata sandi.

Oleh karena itu, kita lakukan *backtracing* dengan melihat log yang ada, seperti `.DS_Store` yang menyimpan data nama file & folder; serta `.bash_history` yang menyimpan history command yang dijalankan oleh terminal.

```
$ cd daunsingkong
$ head -6 .bash_history
ls
man ls
man 7z
vimtutor
date
7z a flag flag.png -p`ls|tail -n 13|head -n 11|head -n 7|tail -n 5|tail -n 3|tail -n 2|head -n 1`
```

Secara garis besar, dapat diketahui bahwa kata sandi berasal dari hasil eksekusi perintah `ls| tail -n 13| head -n 11| head -n 7| tail -n 5| tail -n 3| tail -n 2| head -n 1`

Selanjutnya dengan mengekstrak informasi dari `.DS_Store`, kita dapat me-reproduce perintah di atas dengan data filename yang diperoleh.

```
$ git clone https://github.com/gehaxelt/Python-dsstore
$ python3 Python-dsstore/main.py .DS_Store | uniq > data.txt

$ cat data.txt
Count:  62
daunsingkongmiripdaunapa
flag.7z
flag.png
inginkucumbuubicilembu
ladangubihabisdimakanbabi
marikitabercocoktanam
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pilemkartun
puccukkubi
pucukubiitudaunsingkong
singkongenak
siskaenyatigakali
takperludibajak
Tanamtanamubi

$ cat data.txt | tail -n 13|head -n 11|head -n 7|tail -n 5|tail -n 3|tail -n 2|head -n 1
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
```

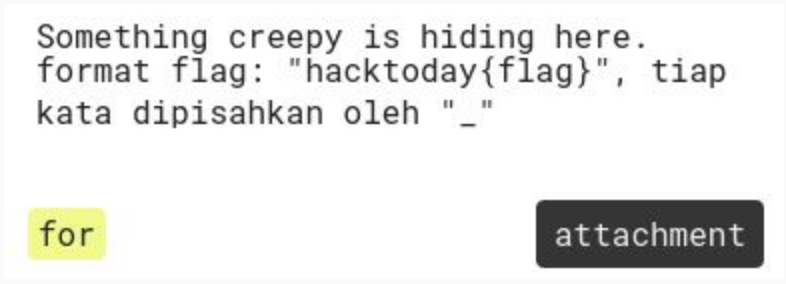
Hasilnya, kita peroleh sebuah kata sandi untuk mengekstrak berkas `flag.7z`, yaitu `pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf`

```
$ 7z x -ppertanianindonesiakanlebihbaikjikapetaninyatidakmainctf flag.7z
```

hacktoday{DS_Store_h4ve_ur_f0lder_nam3___}

Flag:hacktoday{DS_Store_h4ve_ur_f0lder_nam3___}

Stegosaurus



Diberikan berkas `zip` bernama `bendera.zip`. Setelah melakukan ekstraksi, ditemukan berkas `bendera.txt` yang berisikan sebuah paragraf teks disertai beberapa `white space`.

```
$ unzip bendera.zip
$ head bendera.txt
Stegosaurus (/ˌstɛgəˈsɔːrəs/[1]), from Greek stegos (στέγος) which means roof and sauros (σαῦρος)
which means lizard, is a genus of herbivorous thyreophoran dinosaur. Fossils of this genus date to the
Late Jurassic period, where they are found in Kimmeridgian to early Tithonian aged strata, between 155
and 150 million years ago, in the western United States and Portugal. Of the species that have been
classified in the upper Morrison Formation of the western US, only three are universally recognized; S.
stenops, S. ungulatus and S. sulcatus. The remains of over 80 individual animals of this genus have been
found. Stegosaurus would have lived alongside dinosaurs such as Apatosaurus, Diplodocus, Brachiosaurus,
Allosaurus, and Ceratosaurus; the latter two may have preyed on it.
```

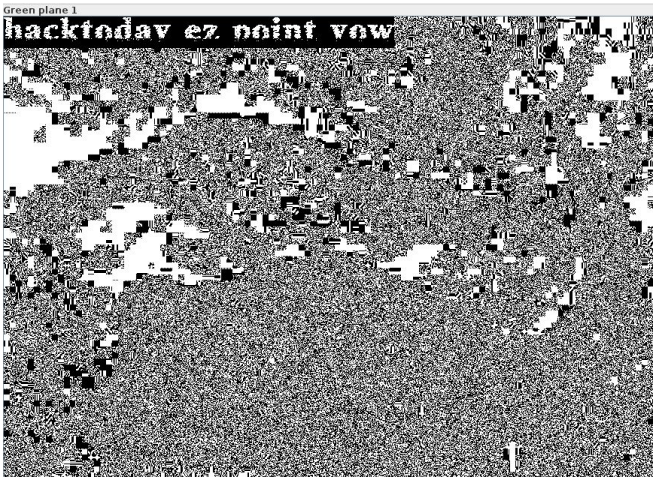
Merujuk pada judul soal & pola `white space` yang ada, kita dapat berasumsi bahwa terdapat informasi yang disembunyikan menggunakan metode `white space steganography`. Di sini kita menggunakan bantuan `stegsnow` untuk menggali informasi tersembunyi pada `bendera.txt`

```
$ stegsnow -C bendera.txt
https://drive.google.com/file/d/17abT2zPLrVUZJLQ-PbW3qU9L__pihQtJ/view?usp=sharing
```

Hasilnya diperoleh sebuah `gdrive link` yang merujuk pada berkas citra `pokeslow.png` seperti berikut ini:

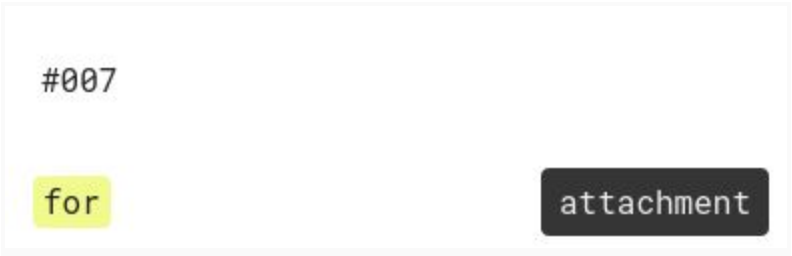
```
$ gdown --id 17abT2zPLrVUZJLQ-PbW3qU9L__pihQtJ -O pokeslow.png
```

Merujuk kembali pada konteks `steganography`, penggalian informasi flag dilanjutkan dengan bantuan `stegsolve`. Hasilnya diperoleh:



`Flag:hacktoday{ez_point_yow}`

Nothosaurus



Diberikan berkas `zip` bernama nothosauruss.zip. Setelah dilakukan ekstraksi diperoleh beberapa berkas yang nampaknya merupakan bagian-bagian dari `zip file`

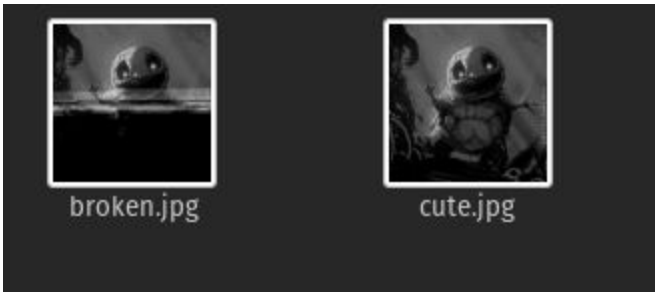
```
$ unzip nothosauruss.zip
$ ls
again  be  ill  nothosauruss.zip  okay  today

$ file *
again:          data
be:             data
ill:            data
nothosauruss.zip: Zip archive data, at least v2.0 to extract
okay:           Zip archive data, at least v2.0 to extract
today:          data
```

Seperti biasa kita perlu menebak *order* yang benar agar `zip file` tetap menjadi entitas file yang valid. Berikut kembali kita lakukan permutasi sederhana sebagai berikut:

```
from itertools import permutations
from zipfile import ZipFile

def open_file(name):
   with open(name,'rb') as f:
       return f.read()

header = open_file('okay')
parts  = 'again be ill today'.split()
perms  = permutations(parts, 4)
contents = dict(zip(parts, map(open_file, parts)))

for p in perms:
   content = b''.join(contents[_] for _ in p)

   with open('flag.zip', 'wb') as f:
       f.write(header + content)
```

```
    try:
        z = ZipFile('flag.zip')
        z.extractall('.')
        print('okay ' + ' '.join(p))
        break
    except:
        continue
```

```
$ python solve.py
okay today ill be again

$ ls
again  be  flag.zip  ill  nothosaurs  nothosauruss.zip  okay  solve.py  today

$ ls nothosaurs
broken.jpg  cute.jpg
```



Hasilnya diperoleh folder `nothosaurs` yang berisi dua buah `JPEG file` seperti yang terlihat pada gambar di atas. Secara sekilas, kita dapat berasumsi bahwa kedua berkas terlihat identik, yang mana hanya dibedakan oleh struktur byte `JPEG data`.

Untuk membuktikannya, kita lakukan sedikit static analysis dengan bantuan `wc` & `diff`

```
$ xxd -p -c1 cute.jpg > cute
$ xxd -p -c1 broken.jpg > broken
$ wc -c cute.jpg broken.jpg
 73591 cute.jpg
 73591 broken.jpg

$ diff cute broken | grep '[<>]' | head -4
< ac
> 68
< bb
> 61
```

Berdasarkan hasil tersebut, dapat diketahui bahwa dua buah file memiliki `byte size` yang sama, yang mana hanya dibedakan oleh beberapa byte string saja. Dari sini, kita akan mencoba untuk melihat isi dari byte string untuk melihat adakah korelasi dengan flag yang dicari

```
$ diff cute broken | grep '>' | xxd -r -p
hacktoday{broken_image}
```

`Flag:hacktoday{broken_image}`

babyVol

`I command you to find the flag!`

`for` `attachment`

Diberikan sebuah berkas bernama `dump.7z`. Setelah melakukan ekstraksi, kita mendapati sebuah memory dump. Dari sini dapat kita uraikan skema pengerjaan menggunakan `volatility` seperti berikut init

OS Profile detection

Sebelum melakukan memory analysis, kita perlu mengetahui detail `os profile` dari memory dump agar dapat menjalankan plugin yang disediakan oleh volatility. Oleh karena itu, kita jalankan plugin `kdbgscan` untuk memperoleh *suggested os profile*

```
$ vol -f dump kdbgscan
**************************************************
Offset (P)                      : 0x284f0a0
KDBG owner tag check            : True
Profile suggestion (KDBGHeader) : Win7SP1x64
PsActiveProcessHead             : 0x2885b90
PsLoadedModuleList              : 0x28a3e90
KernelBase                      : 0xfffff8000265e000
```

Process Enumeration

Kemudian berdasarkan hasil deteksi `os profile (Win7SP1x64)`, kita jalankan plugin `pslist` untuk mengetahui `running process` saat memory dump dibuat

```
$ vol -f dump --profile=Win7SP1x64 pslist
Offset(V)          Name            PID     PPID   Thds   Hnds     Sess   Wow64  Start
------------------ -------------- ------ ------ ------ -------- ------ ------ -------------------
..
..
0xfffffa800283db30 explorer.exe   1316   1272    32      693     1       0      2020-07-29 02:55:14
0xfffffa8002976b30 notepad.exe     2036   1316    2       63      1       0      2020-07-29 02:55:30
0xfffffa8002b1eb30 SearchIndexer. 1124    436    14      565     0       0      2020-07-29 02:55:34
0xfffffa8002c34b30 wmpnetwk.exe   1852    436    16      427     0       0      2020-07-29 02:55:39
0xfffffa8002c43b30 svchost.exe     960    436    12      352     0       0      2020-07-29 02:55:40
0xfffffa8002d9bb30 WmiPrvSE.exe   2164    544    7       119     0       0      2020-07-29 02:55:49
0xfffffa8002e8d9e0 cmd.exe         2380   1316    1       19      1       0      2020-07-29 02:56:00
```

History Enumeration

Dari hasil `process enumeration`, diketahui bahwa terdapat instance `cmd.exe` yang berjalan, yang mana menandakan adanya aktivitas `command-line` pada terminal. Untuk melihat `history command-line`, kita jalankan plugin `cmdscan` seperti berikut. Hasilnya kita menjumpai informasi terkait flag yang diminta

```
$ vol -f dump --profile=Win7SP1x64 cmdscan
**************************************************
CommandProcess: conhost.exe Pid: 2388
CommandHistory: 0x12eab0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 2 LastAdded: 1 LastDisplayed: 1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #0 @ 0x12dde0: dir
Cmd #1 @ 0x133680: hacktoday{yOUv3__folll0wed_My_c0mm4ND_f3ry_w3LL__}
Cmd #15 @ 0xf0158:
Cmd #16 @ 0x12dc20:
```

Alternatives

Terdapat alternatif opsi yang dapat kita gunakan untuk mendapatkan flag dalam representasi `plain-text`, salah satunya dengan membaca memory dump sebagai string dengan `little-endian` byte ordering. Hal ini dapat dilakukan mengingat default byte ordering pada sistem operasi Windows umumnya mengadopsi representasi `little-endian`

```
$ strings -ae l dump | grep -oP 'hacktoday{.*}' -m2
hacktoday{d1Z__iz_n0T_d4_Fl4g_nOr_Da_command_???}
hacktoday{yOUv3__folll0wed_My_c0mm4ND_f3ry_w3LL__}
```

`Flag:hacktoday{yOUv3__folll0wed_My_c0mm4ND_f3ry_w3LL__}`

babyVol V2

Now with extra process™

for attachment

Kali ini kita diberikan sebuah berkas bernama `dump.7z`. Setelah melakukan ekstraksi, kita mendapati sebuah memory dump. Dari sini dapat kita uraikan skema pengerjaan menggunakan `volatility` seperti berikut ini:

OS Profile detection

Pada tahap ini, seperti biasa kita perlu menemukan base `os profile` yang sesuai dengan memory dump yang diberikan. Hal ini umumnya dapat diselesaikan menggunakan plugin `kdbgscan`. Akan tetapi, pada kasus ini proses scanning justru membutuhkan waktu yang relatif lama dan tidak membuahkan hasil.

Setelah mempelajari behavior `volatility` lebih lanjut, kecenderungan ini dikonfirmasi pada isu #412. Sebagaimana tertera pada isu tersebut, dijelaskan bahwa `kdbgscan` umumnya dikhususkan untuk sampel memori Windows. Hal ini secara tidak langsung juga menjelaskan alasan mengapa secara default volatility hanya menyediakan `os profile` dengan basis sistem operasi Windows.

```
$ vol --info | grep Profile -m5
Profiles
VistaSP0x64             - A Profile for Windows Vista SP0 x64
VistaSP0x86             - A Profile for Windows Vista SP0 x86
VistaSP1x64             - A Profile for Windows Vista SP1 x64
VistaSP1x86             - A Profile for Windows Vista SP1 x86
```

Berdasarkan argumen tersebut, dapat dipastikan bahwa sampel memori didump pada sistem operasi non-Windows. Meskipun demikian, `volatility` masih dapat menganalisis sampel memori jika dan hanya jika user menyediakan `os profile` yang bersesuaian.

Information gathering

Pada tahap ini, kita perlu mencari informasi mengenai `os version` dan `kernel version`. Hal ini dilakukan agar kita dapat membuat sebuah `os profile` yang bersesuaian dengan sampel memori. Adapun berikut ini adalah keyword yang digunakan dalam proses penelusuran beserta hasilnya:

```
$ strings dump | grep 'VERSION=' -A3 -m1
VERSION="20.04 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04 LTS"

$ strings dump | grep 'BOOT_IMAGE=' -m1
BOOT_IMAGE=/boot/vmlinuz-5.4.0-42-generic root=UUID=a86eb3dc-3d09-40e6-8da6-985d3a1ab9b3 ro quiet splash
```

Berdasarkan hasil penelusuran di atas, dapat diketahui bahwa `os version` dan `kernel version` pada sampel memori secara berturut-turut adalah `20.04 LTS (Focal Fossa)` dan `5.4.0-42-generic`.

Build custom Volatility OS Profile using Docker Image

Setelah memperoleh informasi yang dibutuhkan, tiba saatnya dilakukan pembuatan `os profile` sebagaimana tercantum pada dokumentasi berikut ini. Berdasarkan literasi tersebut, diketahui bahwa kita memerlukan `sistem operasi yang sama` dengan `sampel memori` yang digunakan. Kendati demikian, hal tersebut pastinya akan membutuhkan resource waktu yang lebih lama sehingga menjadikannya kurang affordable.

Alternatifnya, kita dapat mensimulasikan proses tersebut pada sebuah `Docker container` dengan `base Image` yang memiliki spesifikasi `os version` & `kernel version` yang sama. Berikut ini langkah-langkah yang dilakukan:

a. Build a docker image based on given os version & kernel version

   Pada tahap ini, definisikan sebuah docker image yang berasal dari Docker Hub ubuntu. Kemudian lakukan instalasi kernel sesuai versi yang diberikan. Terakhir lakukan kompilasi `dwarf.module` dan satukan dengan `kernel system.map` ke dalam sebuah `zip archive`.

Proses di atas dapat dicapai dengan mem-build Dockerfile sebagai berikut

```
$ vim Dockerfile
```

```
FROM ubuntu:20.04

RUN apt update
RUN apt -y install linux-tools-5.4.0-42-generic linux-headers-5.4.0-42-generic
linux-modules-5.4.0-42-generic
RUN apt -y install zip git build-essential dwarfdump

RUN git clone https://github.com/volatilityfoundation/volatility.git
RUN sed -i 's/$(shell uname -r)/"5.4.0-42-generic"/' volatility/tools/linux/Makefile
RUN cd volatility/tools/linux/ && make
RUN zip /Ubuntu20.04.zip volatility/tools/linux/module.dwarf /boot/System.map-5.4.0-42-generic
```

```
$ docker build -t volatility:20.04 .
```

b. Copy os profile from Docker container

Setelah proses sebelumnya berhasil dijalankan, langkah selanjutnya adalah menyalin os profile dari `Docker Image`. Untuk itu, kita perlu menjalankan instance container untuk menginisiasi `Docker volume`

```
$ docker run --name profile volatility:20.04
$ docker cp profile:/Ubuntu20.04.zip Ubuntu20.04.zip
```

Terakhir, masukkan os profile ke dalam directory `plugins/overlays/linux/`. Hasilnya adalah sebagai berikut:

```
$ cp Ubuntu20.04.zip
/usr/local/lib/python2.7/dist-packages/volatility-2.6.1-py2.7.egg/volatility/plugins/overlays/l
inux/

$ vol --info | grep Profile -m6
Profiles
LinuxUbuntu20_04x64                     - A Profile for Linux Ubuntu20.04 x64
VistaSP0x64                             - A Profile for Windows Vista SP0 x64
VistaSP0x86                             - A Profile for Windows Vista SP0 x86
VistaSP1x64                             - A Profile for Windows Vista SP1 x64
```

Process Enumeration

Setelah selesai berurusan dengan pembuatan `os profile`, kini kita dapat melakukan eksplorasi lebih lanjut terhadap sampel memori. Langkah pertama yang kita lakukan adalah mengenumerasi `running process` saat Ubuntu dijalankan, menggunakan plugin `linux_psaux`.

```
$ vol -f dump --profile=LinuxUbuntu20_04x64 linux_psaux
Pid     Uid     Gid     Arguments
1       0       0       /sbin/init splash
2       0       0       [kthreadd]
3       0       0       [rcu_gp]
4       0       0       [rcu_par_gp]
..
..
1908    0       0       [uas]
1951    0       0       [jbd2/sdb2-8]
1952    0       0       [ext4-rsv-conver]
1996    1000    1000    ./fl4g
```

Hasilnya tampak bahwa terdapat executable program `fl4g` yang dijalankan

Mounted Device Enumeration

Sebelumnya, telah diketahui bahwa terdapat program bernama `fl4g`. Akan tetapi, kita belum mengetahui path saat program dijalankan sehingga perlu dilakukan penelusuran lebih lanjut. Kali ini kita jalankan plugin `linux_mount` untuk mengetahui `mounted device` yang tersedia.

```
$ vol -f dump --profile=LinuxUbuntu20_04x64 linux_mount  | grep '^/dev/'
/dev/sda5               /                                       ext4        rw,relatime
/dev/sda1               /boot/efi                               vfat        rw,relatime
/dev/sdb2               /media/hektod/casper-rw                 ext3        rw,relatime,nosuid,nodev
/dev/loop2              /snap/core18/1705                       squashfs    ro,relatime,nodev
/dev/fuse               /run/user/1000/doc                      fuse        rw,relatime,nosuid,nodev
/dev/loop0              /snap/gtk-common-themes/1506            squashfs    ro,relatime,nodev
/dev/loop3              /snap/snap-store/433                    squashfs    ro,relatime,nodev
/dev/loop1              /snap/gnome-3-34-1804/24                squashfs    ro,relatime,nodev
/dev/loop4              /snap/snapd/7264                        squashfs    ro,relatime,nodev
/dev/sdb1               /media/hektod/UBUNTU 20_0               vfat        rw,relatime,nosuid,nodev
```

Hasilnya, kita menjumpai dua buah `USB flash drive (/dev/sdb1,2)` dengan nama drive secara berturut-turut, yaitu `UBUNTU 20_0` & `casper-rw`.

Berdasarkan temuan tersebut, kita dapat berasumsi bahwa `/dev/sdb1` merupakan bootable drive, sedangkan `/dev/sdb2` merupakan storage drive yang menyimpan program `fl4g`

History Enumeration

Untuk membuktikan asumsi-asumsi di atas, kali ini kita jalankan plugin `linux_bash` dengan tujuan menampilkan `.bash_history` dari sampel memori.

```
$ vol -f dump --profile=LinuxUbuntu20_04x64 linux_bash
Pid      Name                 Command Time                   Command
-------- -------------------- ------------------------------ -------
1763     bash                 2020-07-23 11:16:22 UTC+0000   history
1763     bash                 2020-07-23 11:16:22 UTC+0000   cd /media/hektod/casper-rw/
1763     bash                 2020-07-23 11:16:22 UTC+0000   ls
1763     bash                 2020-07-23 11:16:22 UTC+0000   uname -r
1763     bash                 2020-07-23 11:16:22 UTC+0000   ls
1763     bash                 2020-07-23 11:16:27 UTC+0000   whoami
1763     bash                 2020-07-23 11:16:38 UTC+0000   )
1763     bash                 2020-07-23 11:16:38 UTC+0000   uname -r
1763     bash                 2020-07-23 11:16:57 UTC+0000   touch flag.txt
1763     bash                 2020-07-23 11:17:11 UTC+0000   gedit flag.txt
1763     bash                 2020-07-23 11:19:05 UTC+0000   ;s
1763     bash                 2020-07-23 11:19:06 UTC+0000   ls
1763     bash                 2020-07-23 11:19:41 UTC+0000   cd /media/hektod/casper-rw/
1763     bash                 2020-07-23 11:19:42 UTC+0000   ls
1763     bash                 2020-07-23 11:20:08 UTC+0000   ./fl4g
```

Hasilnya, dapat dipastikan bahwa program `fl4g` berada pada directory `/media/hektod/casper-rw`

File Acquisition

Setelah mendapatkan absolute path dari program `fl4g`, kita dapat memanfaatkan plugin `linux_find_file` untuk mencari `offset` dan melakukan recovery file

```
$ vol -f dump --profile=LinuxUbuntu20_04x64 linux_find_file -F "/media/hektod/casper-rw/fl4g"
Inode Number                 Inode File Path
---------------- ---------------- ---------
12 0xffff93a57a2c6bf0 /media/hektod/casper-rw/fl4g

$ vol -f dump --profile=LinuxUbuntu20_04x64 linux_find_file -i 0xffff93a57a2c6bf0 -O fl4g
```

Flag Discovery

Pada tahap ini, kita lakukan analisis terhadap program `fl4g`. Hasilnya ditemukan flag dengan menyeleksi string tanpa null byte character

```
$ file fl4g
fl4g: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, BuildID[sha1]=e84af12bf607bc8be74a262e92656aaf88474d55, for GNU/Linux
3.2.0, not stripped

$ ./fl4g
Disassemble me!

$ sed 's/\x0//g' fl4g | grep -oP 'hacktoday{.*}'
hacktoday{jU5tt__4_f3w_s1mPl33_pr0CE5s35s}
```

Alternatives

Mengingat informasi flag tersimpan dalam representasi `plain-text`, proses pencarian dapat juga dilakukan secara langsung pada file `dump` seperti sebelumnya

```
$ sed 's/\x0//g' dump | grep -oP 'hacktoday{.*}' -m1
hacktoday{jU5tt__4_f3w_s1mPl33_pr0CE5s35s}
```

`Flag:hacktoday{jU5tt__4_f3w_s1mPl33_pr0CE5s35s}`

Kataware Doki

So we don't forget when we wake up.
Let's write our names on each other.

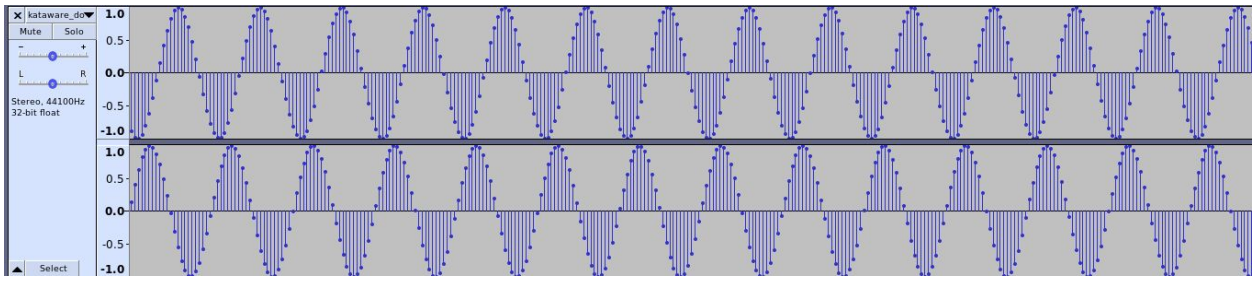
for attachment

Diberikan sebuah berkas `zip` bernama kataware_doki.zip. Setelah melakukan ekstraksi, diperoleh stereo `WAV file` sebagaimana berikut ini:

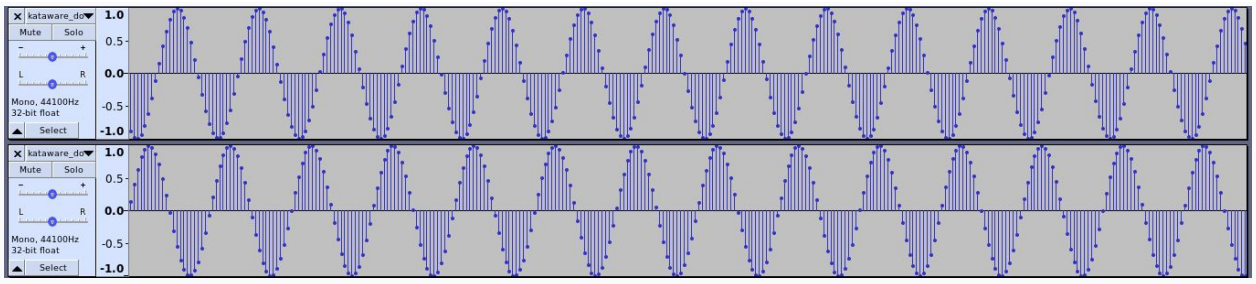
```
$ unzip kataware_doki.zip
$ file kataware_doki.wav
kataware_doki.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, stereo 44100 Hz

$ mediainfo kataware_doki.wav
General
Complete name                            : kataware_doki.wav
Format                                   : Wave
File size                                : 19.4 MiB
Duration                                 : 1 min 55 s
Overall bit rate mode                    : Constant
Overall bit rate                         : 1 411 kb/s

Audio
Format                                   : PCM
Format settings                          : Little / Signed
Codec ID                                 : 1
Duration                                 : 1 min 55 s
Bit rate mode                            : Constant
Bit rate                                 : 1 411.2 kb/s
Channel(s)                               : 2 channels
Sampling rate                            : 44.1 kHz
Bit depth                                : 16 bits
Stream size                              : 19.4 MiB (100%)
```

Audio Analysis

Pada tahap ini, kita lakukan analisis menggunakan bantuan `Audacity`. Hasilnya diketahui bahwa `waveform` dari kedua channel tidak identik satu sama lain. Dari sini kita dapat berasumsi bahwa terdapat 2 buah informasi yang disembunyikan pada masing-masing `audio channel`.



Berdasarkan asumsi tersebut, kita segmentasikan `audi file` ke dalam representasi `mono channel` menggunakan opsi `Split Stereo to Mono`



SSTV decoding

Setelah beberapa saat dilakukan percobaan, diasumsikan bahwa `audio waveform` satisfiable dengan `transmisi SSTV`. Untuk membuktikan asumsi tersebut, terlebih dahulu kita lakukan konfigurasi module `null output` dengan perintah:

```
$ pactl load-module module-null-sink sink_name=virtual-cable
26
```

Lalu lakukan setup system audio menjadi `null output` untuk menginisiasi interface `virtual-cable`. Barulah kemudian dilakukan `SSTV decoding` menggunakan bantuan `QSSTV` pada masing-masing channel. Hasilnya diperoleh representasi transmisi citra sebagai berikut

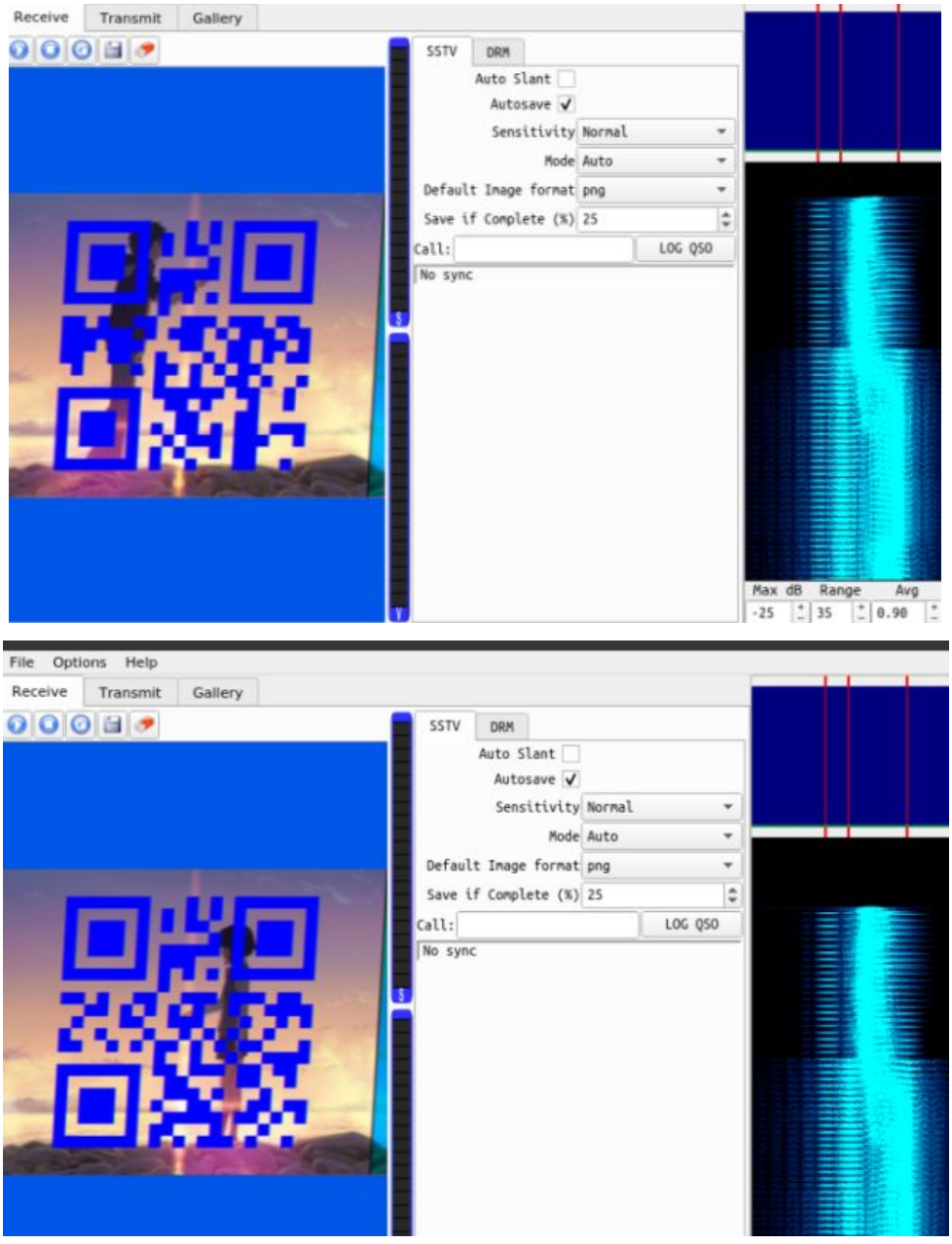


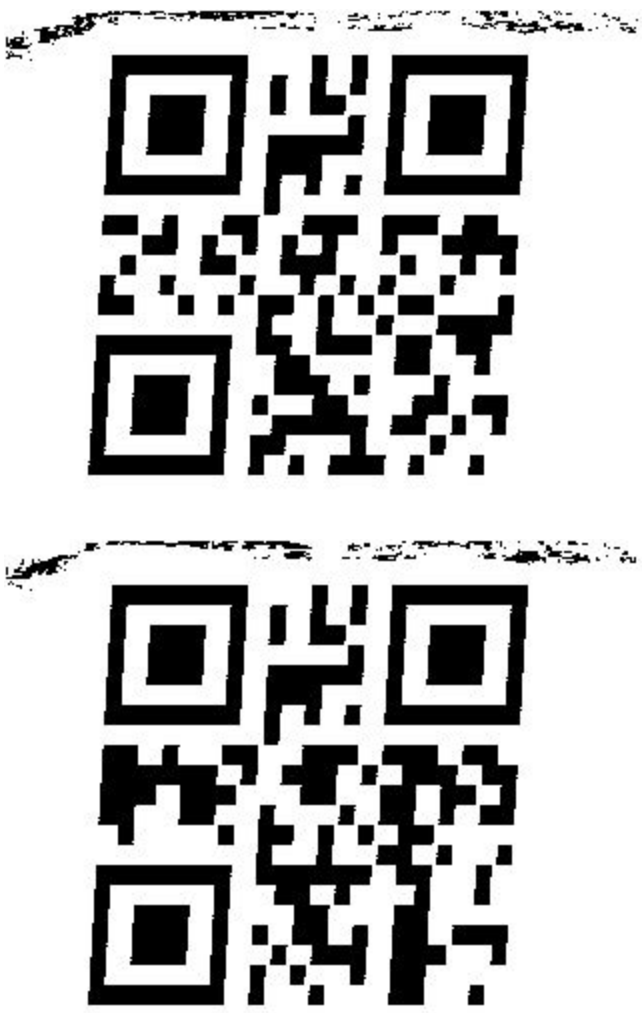
Image Pre-Processing

Berdasarkan temuan sebelumnya, terdapat dua buah `QR code` pada sebuah background anime (sebut saja `Your Name`). Kondisi demikian mengakibatkan perlunya `Image segmentation` agar entitas `foreground` & `background` dapat dipisahkan sehingga `QR code` dapat dikenali.

Pada tahap ini, kita mengambil pendekatan dengan menyeleksi `neighborhood pixel` terhadap warna (0, 0, 255) serta 40 colormap dengan frekuensi tertinggi pada citra. Adapun pada proses terakhir, kita gunakan module `pyzbar` untuk membaca `QR code` hasil segmentasi. Berikut implementasi beserta hasil eksekusi program:

```
from pyzbar.pyzbar import decode
from collections import Counter
from PIL import Image

import glob

imgs = glob.glob('*.png')
flag = ''

for num,image in enumerate(imgs):
   base = Image.open(image)
   w,h  = base.size
   mode = base.mode
   data = base.getdata()

   most_common = [_[0] for _ in Counter(data).most_common(40)]
   canvas = Image.new(mode, (w,h))
   result = []

   for pixels in list(data):
       r,g,b = pixels

       if pixels in most_common:
           result += [(0,0,0)]
       elif 0 <= r <= 10 and 0 <= g <= 10 and 23 and 250 <= b <= 255:
           result += [(0,0,0)]
       else:
           result += [(255,255,255)]

   canvas.putdata(result)
   temp = list(decode(canvas)[0])[0]
   flag += temp.decode()

   canvas.save(f"normalized/{num}.png")

print(flag)
```

```
$ python3 solve.py
hacktoday{I_shit3ru_you}
```



Flag:hacktoday{I_shit3ru_you}