Et l'homme créa le Code

Éric BERTHOMIER
eric.berthomier@free.fr

17 octobre 2020

Version 1.0 - Version Stagiaire

Préambule

Ce cours est une adaptation libre des pages web suivantes :

- **Linux Assembly and Disassembly an Introduction)** [1]
- **CWE-77 : Improper Neutralization of Special Elements used in a Command ('Command Injection')** [2]

1. http://lucifer.phiral.net/linuxasmone.htm
2. http://cwe.mitre.org/data/definitions/77.html

De la source à l'exécutable

bonjour.c

```
main() {
   write (1,"Hello, World!\n", 14);
   return 0;
}
```

Compilation de bonjour.c

```
$ gcc -c bonjour.c
$ gcc -o bonjour bonjour.o
```

./bonjour

```
Hello, World!
```

Ce que contient l'exécutable

```
    .file   "bonjour.c"
    .section    .rodata
.LC0:
    .string "Hello, World!\n"
    .text
    .globl main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    andl    $-16, %esp
    subl    $16, %esp
    movl    $14, 8(%esp)
    movl    $.LC0, 4(%esp)
    movl    $1, (%esp)
    call    write
    movl    $0, %eax
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident "GCC: (Debian 4.7.2-5) 4.7.2"
    .section    .note.GNU-stack,"",@progbits
```

Explications

- Le code assembleur a été obtenu par

```
gcc bonjour.c -S -o bonjour.s
```

- L'assembleur peut devenir exécutable par l'appel de

```
gcc bonjour.s -o bonjour
```

  où `bonjour.s` est le code assembleur.
- L'assembleur n'est donc pas le langage de la machine mais quelque chose de proche ...

De inexécutable au code

- Le code est exécuté par le système, il se doit donc d'être compris par le système.
- Il est possible de lire ce que va lire le système à lexécution du programme, cette manipulation s'appelle désassembler un programme.
- Pour désassembler notre programme :

```
objdump -d bonjour
```

Ex Machina (1/3)

```
bonjour:    file format elf32-i386


Disassembly of section .init:

080482bc <_init>:
 80482bc:  55                    push   %ebp
 80482bd:  89 e5                 mov    %esp,%ebp
 80482bf:  53                    push   %ebx
 80482c0:  83 ec 04              sub    $0x4,%esp
 80482c3:  e8 00 00 00 00        call   80482c8 <_init+0xc>
 80482c8:  5b                    pop    %ebx
 80482c9:  81 c3 a4 13 00 00     add    $0x13a4,%ebx
 80482cf:  8b 93 fc ff ff ff     mov    -0x4(%ebx),%edx
 80482d5:  85 d2                 test   %edx,%edx
 80482d7:  74 05                 je     80482de <_init+0x22>
 80482d9:  e8 22 00 00 00        call   8048300 <__gmon_start__@plt>
 80482de:  58                    pop    %eax
 80482df:  5b                    pop    %ebx
 80482e0:  c9                    leave
 80482e1:  c3                    ret

Disassembly of section .plt:

080482f0 <__gmon_start__@plt-0x10>:
 80482f0:  ff 35 70 96 04 08     pushl  0x8049670
 80482f6:  ff 25 74 96 04 08     jmp    *0x8049674
 80482fc:  00 00                 add    %al,(%eax)
    ...
```

Ex Machina (2/3)

```
08048300 <__gmon_start__@plt>:
 8048300:  ff 25 78 96 04 08     jmp    *0x8049678
 8048306:  68 00 00 00 00        push   $0x0
 804830b:  e9 e0 ff ff ff        jmp    80482f0 <_init+0x34>

08048310 <__libc_start_main@plt>:
 8048310:  ff 25 7c 96 04 08     jmp    *0x804967c
 8048316:  68 08 00 00 00        push   $0x8
 804831b:  e9 d0 ff ff ff        jmp    80482f0 <_init+0x34>

08048320 <write@plt>:
 8048320:  ff 25 80 96 04 08     jmp    *0x8049680
 8048326:  68 10 00 00 00        push   $0x10
 804832b:  e9 c0 ff ff ff        jmp    80482f0 <_init+0x34>

Disassembly of section .text:

08048330 <_start>:
 8048330:  31 ed                 xor    %ebp,%ebp
 8048332:  5e                    pop    %esi
 8048333:  89 e1                 mov    %esp,%ecx
 8048335:  83 e4 f0              and    $0xfffffff0,%esp
 8048338:  50                    push   %eax
 8048339:  54                    push   %esp
 804833a:  52                    push   %edx
 804833b:  68 50 84 04 08        push   $0x8048450
 8048340:  68 60 84 04 08        push   $0x8048460
 8048345:  51                    push   %ecx
 8048346:  56                    push   %esi
```

Ex Machina (3/3)

```
 8048347:  68 1c 84 04 08        push   $0x804841c
 804834c:  e8 bf ff ff ff        call   8048310 <__libc_start_main@plt>
 8048351:  f4                    hlt
 8048352:  90                    nop
 8048353:  90                    nop
 8048354:  90                    nop
 8048355:  90                    nop
 8048356:  90                    nop
 8048357:  90                    nop
 8048358:  90                    nop
 8048359:  90                    nop
 804835a:  90                    nop
 804835b:  90                    nop
 804835c:  90                    nop
 804835d:  90                    nop
 804835e:  90                    nop
 804835f:  90                    nop

08048360 <deregister_tm_clones>:
 8048360:  b8 8f 96 04 08        mov    $0x804968f,%eax
 8048365:  2d 8c 96 04 08        sub    $0x804968c,%eax
 804836a:  83 f8 06              cmp    $0x6,%eax
 804836d:  77 02                 ja     8048371 <deregister_tm_clones+0x11>
 804836f:  f3 c3                 repz ret
 8048371:  b8 00 00 00 00        mov    $0x0,%eax
 8048376:  85 c0                 test   %eax,%eax
 8048378:  74 f5                 je     804836f <deregister_tm_clones+0xf>
 804837a:  55                    push   %ebp
 804837b:  89 e5                 mov    %esp,%ebp
 804837d:  83 ec 18              sub    $0x18,%esp
```

Conclusion

Compilation

La simple compilation d'un programme ne protège pas son contenu.

```
.text:0040105B                  mov     esi, edx
.text:0040105D                  mov     eax, [ebp+BaseStructure]
.text:00401060                  mov     eax, [eax+PackerStructure.nSizeOfEncodeData]   1
.text:00401063                  mov     [ebp+var_counter], eax ; eax = 0x7f2d
.text:00401066                  neg     eax
.text:00401068
.text:00401068 AntiEmulationLoop:                       ; CODE XREF: start+87↓j
.text:00401068                  mov     eax, [ebp+var_counter]   2
.text:0040106B                  mov     ecx, [ebp+var_counter]
.text:0040106E                  inc     ecx
.text:0040106F                  mov     [ebp+var_counter], ecx
.text:00401072                  test    eax, eax
.text:00401074                  jz      short loc_401089 ; jump out when overflow occurs   3
.text:00401076                  lea     esi, [ecx]
.text:00401078                  inc     ecx
.text:00401079                  sub     edx, ebp
.text:0040107B                  movsx   ebx, cl
.text:0040107E                  dec     ecx
.text:0040107F                  mov     edx, ecx
.text:00401081                  cmp     eax, eax
.text:00401083                  cmp     edx, eax
.text:00401085                  sub     eax, ebx
.text:00401087                  jmp     short AntiEmulationLoop
.text:00401089 ; ---------------------------------------------------------------------------
.text:00401089
.text:00401089 loc_401089:                              ; CODE XREF: start+74↑j
```

Exécution d'un programme

Un programme peut s'exécuter **globalement** sous 3 environnements différents :

- dans une environnement utilisateur

Exécution d'un programme

Un programme peut s'exécuter **globalement** sous 3 environnements différents :

- dans une environnement utilisateur
- dans un environnement administrateur

Exécution d'un programme

Un programme peut s'exécuter **globalement** sous 3 environnements différents :

- dans une environnement utilisateur
- dans un environnement administrateur
- dans un environnement système

Exécution d'un programme

Un programme peut s'exécuter **globalement** sous 3 environnements différents :

- dans une environnement utilisateur
- dans un environnement administrateur
- dans un environnement système

Notre intérêt est bien sûr de s'attaquer aux programmes s'exécutant en environnement Système ou en Administrateur.

Illustration 1 - Les faits

Le petit programme qui suit accepte un nom de fichier en argument et affiche le contenu dudit fichier à l'utilisateur.

Illustration 1 - Les faits

Le petit programme qui suit accepte un nom de fichier en argument et affiche le contenu dudit fichier à l'utilisateur.
Le programme est installé avec un `setuid root` car il est destiné à une utilisation pédagogique.

Illustration 1 - Les faits

Le petit programme qui suit accepte un nom de fichier en argument et affiche le contenu dudit fichier à l'utilisateur.
Le programme est installé avec un `setuid root` car il est destiné à une utilisation pédagogique.
Il doit en effet permettre aux administrateurs du système **en formation** de pouvoir visualiser les fichiers et les dossiers système sans pour autant leur donner un accès en modification.

Illustration 1 - Les faits

Le petit programme qui suit accepte un nom de fichier en argument et affiche le contenu dudit fichier à l'utilisateur.
Le programme est installé avec un `setuid root` car il est destiné à une utilisation pédagogique.
Il doit en effet permettre aux administrateurs du système **en formation** de pouvoir visualiser les fichiers et les dossiers système sans pour autant leur donner un accès en modification.

badcode1.c

```c
#include <string.h>
#include <stdio.h>

int main(char* argc, char** argv) {
    char cmd [256] = "/bin/cat ";
    strcat(cmd, argv[1]);
    printf ("Execution de %s\n", cmd);
    system(cmd);
}
```

Illustration 1 - Le problème

- Du fait que le programme s'exécute avec des privilèges root, l'appel de la fonction `system()` permet d'exécuter des commandes avec des privilèges root.

Illustration 1 - Le problème

- Du fait que le programme s'exécute avec des privilèges root, l'appel de la fonction `system()` permet d'exécuter des commandes avec des privilèges root.
- Si un utilisateur spécifie un nom de fichier standard, le programme fonctionnera comme souhaité.

Illustration 1 - Le problème

- Du fait que le programme s'exécute avec des privilèges root, l'appel de la fonction `system()` permet d'exécuter des commandes avec des privilèges root.
- Si un utilisateur spécifie un nom de fichier standard, le programme fonctionnera comme souhaité.
- Cependant, si un attaquant passe une chaîne de caractère de type `;rm -rf /` en plus du nom de fichier à afficher, l'appel de la fonction `system()` exécute la commande `cat` **ET** effectue la seconde commande.

Illustration 1 - Le problème

- Du fait que le programme s'exécute avec des privilèges root, l'appel de la fonction `system()` permet d'exécuter des commandes avec des privilèges root.
- Si un utilisateur spécifie un nom de fichier standard, le programme fonctionnera comme souhaité.
- Cependant, si un attaquant passe une chaîne de caractère de type `;rm -rf /` en plus du nom de fichier à afficher, l'appel de la fonction `system()` exécute la commande `cat` **ET** effectue la seconde commande.

Conséquence

Une destruction complète du système de fichier aura donc lieu !

Démonstration

```
./badcode1 "bonjour.txt ; ls -al /"

Hello, World!total 120
drwxr-xr-x 26 root root 4096 dec. 29 09:56 .
drwxr-xr-x 26 root root 4096 dec. 29 09:56 ..
drwxr-xr-x  2 root root 4096 dec. 29 09:55 bin
drwxr-xr-x  4 root root 4096 dec. 14 15:47 boot
drwxr-xr-x  2 root root 4096 dec. 29 09:56 .config
drwxr-xr-x 15 root root 3400 janv. 1 11:23 dev
drwxr-xr-x 164 root root 12288 dec. 31 18:30 etc
drwxr-xr-x  3 root root 4096 aout 25 2013 home
lrwxrwxrwx  1 root root   32 aout 18  2013 initrd.img -> /boot/initrd.img-3.2.0-4-686-pae
drwxr-xr-x 18 root root 4096 oct. 19 20:52 lib
drwx------  2 root root 16384 aout 18 2013 lost+found
drwxr-xr-x  5 root root 4096 janv. 1 11:56 media
drwxr-xr-x  5 root root 4096 sept. 6 2013 mnt
drwxr-xr-x  2 root root 4096 mars 29 2014 nas
drwxr-xr-x  3 root root 4096 aout 29 2013 opt
dr-xr-xr-x 170 root root  0 janv. 1 11:22 proc
drwx------  2 root root 4096 janv. 1 11:23 .pulse
-rw-------  1 root root  256 aout 18  2013 .pulse-cookie
drwx------ 24 root root 4096 dec. 21 16:11 root
drwxr-xr-x  2 root root 4096 sept. 1 2013 .rpmdb
drwxr-xr-x 26 root root 1020 janv. 1 11:28 run
drwxr-xr-x  2 root root 12288 oct. 19 20:54 sbin
drwxr-xr-x  2 root root 4096 juin 10 2012 selinux
drwxr-xr-x  2 root root 4096 aout 18 2013 srv
drwxr-xr-x 13 root root   0 janv. 1 11:22 sys
drwxrwxrwt 11 root root 4096 janv. 1 13:05 tmp
drwxr-xr-x 10 root root 4096 aout 18 2013 usr
drwxr-xr-x 13 root root 4096 aout 19 2013 var
lrwxrwxrwx  1 root root  28 aout 18  2013 vmlinuz -> boot/vmlinuz-3.2.0-4-686-pae
Execution de /bin/cat bonjour.txt ; ls -al /
```

Conclusion

Mon programme a besoin de droits d'Administrateur

Suis-je vraiment sûr de ce que j'indique ?

Conclusion

Mon programme a besoin de droits d'Administrateur

Suis-je vraiment sûr de ce que j'indique ?

Mon code est inviolable

Mais encore ...

Conclusion

Mon programme a besoin de droits d'Administrateur

Suis-je vraiment sûr de ce que j'indique ?

Mon code est inviolable

Mais encore ...

L'ingénierie inverse est illégale

En France ...