FL : Techniques de hacking (Jon Erickson. Ed. Pearson)

Novembre 2009

Ce document est constitué des notes prises durant la lecture de « Techniques de Hacking », par Jon Erickson, aux éditions Pearson.

DISCLAIMER : d'abord c'est pour moi, alors venez pas râler si paf.

Chapitre 1 : Programmation

Gcc :

gcc ma_source.c
fichier de sortie par défaut : a.out

gcc -g ma_source.c
Ajoute des informations de débogage exploitable par gdb

objdump :

objdump -D a.out | grep -A20 [ma fonction]
ex : [ma fonction] = main()
option -M intel = syntaxe Intel (defaut : AT&T)

gdb :

gdb -q ./mon_fichier
(gdb) break [ma fonction]
(gdb) run
(gdb) info register

```
eax            0xbf9ecfc4       -1080111164
ecx            0xbf9ecf40       -1080111296
edx            0xbf9ecf60       -1080111264
ebx            0xb7fb1ff4       -1208279052
esp            0xbf9ecf24       0xbf9ecf24
ebp            0xbf9ecf28       0xbf9ecf28
esi            0x8048430        134513712
edi            0x8048320        134513440
eip            0x80483e2        0x80483e2 <main+14>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
```

gs 0x33 51

EAX : Accumulateur
ECX : Calculateur
EDX : Données
EBX : Base

ESP : Pointeur de pile (stack)
EBP : Pointeur de base
EIP : pointeur d'instruction
ESI : index de source
EDI : index de destination

Les pointeurs stockent les adresses 32 bits. Le plus important est ESI car il indique l'instruction en cours de lecture par le cpu.

Fichier de configuration : .gdbinit

(gdb) list : affiche la source du programme en cours (si compilé avec « gcc -g »)

(gdb) disassemble [ma_fonction]

Exemple : (gdb) disassemble main
Dump of assembler code for function main:
0x080483d4 <main+0>: lea ecx,[esp+0x4]
0x080483d8 <main+4>: and esp,0xfffffff0
0x080483db <main+7>: push DWORD PTR [ecx-0x4]
0x080483de <main+10>: push ebp
0x080483df <main+11>: mov ebp,esp
0x080483e1 <main+13>: push ecx
0x080483e2 <main+14>: sub esp,0x14
0x080483e5 <main+17>: mov DWORD PTR [ebp-0x8],0x0
0x080483ec <main+24>: jmp 0x80483fe <main+42>
0x080483ee <main+26>: mov DWORD PTR [esp],0x80484e0
0x080483f5 <main+33>: call 0x804830c <puts@plt>
0x080483fa <main+38>: add DWORD PTR [ebp-0x8],0x1
0x080483fe <main+42>: cmp DWORD PTR [ebp-0x8],0x9
0x08048402 <main+46>: jle 0x80483ee <main+26>
0x08048404 <main+48>: mov eax,0x0
0x08048409 <main+53>: add esp,0x14
0x0804840c <main+56>: pop ecx
0x0804840d <main+57>: pop ebp
0x0804840e <main+58>: lea esp,[ecx-0x4]
0x08048411 <main+61>: ret
End of assembler dump.

(gdb) info register eip (abbr : « i r eip »: permet d'identifier l'adresse en cours pour l'instruction. A comparer avec celles de la liste retournée par « disassemble ».

Pour examiner la mémoire : « x » + format (« o » = octal, « x » = hexa, « u » = décimal non signé, « t » = binaire).

Exemple :
(gdb) x/x 0x80483e5
0x80483e5 <main+17>: 0x00f845c7

```
(gdb) x/x $eip
0x80483e5 <main+17>:    0x00f845c7
(gdb) x/x $eip
0x80483e5 <main+17>:    0x00f845c7
(gdb) x/x $eip
0x80483e5 <main+17>:    0x00f845c7
(gdb) x/t $eip
0x80483e5 <main+17>:    00000000111110000100010111000111
```

Tiens, c'est rigolo, on dirait qu'il y une variable pour le pointeur... Est-ce le cas des autres ? Mystère.

En ajoutant un nombre devant le format, on peut voir le contenu des adresses au-delà :
exemple :

```
(gdb) x/x $eip
0x80483e5 <main+17>:    0x00f845c7
(gdb) x/2x $eip
0x80483e5 <main+17>:    0x00f845c7      0xeb000000
(gdb) x/12x $eip
0x80483e5 <main+17>:    0x00f845c7      0xeb000000      0x2404c710      0x080484e0
0x80483f5 <main+33>:    0xffff12e8      0xf84583ff      0xf87d8301      0xb8ea7e09
0x8048405 <main+49>:    0x00000000      0x5914c483      0xfc618d5d      0x909090c3
```

Par défaut, chaque pavé contient 4 octets mais cela peut être changé :

```
(gdb) x/x $eip
0x80483e5 <main+17>:    0x00f845c7
(gdb) x/2x $eip
0x80483e5 <main+17>:    0x00f845c7      0xeb000000
(gdb) x/12x $eip
0x80483e5 <main+17>:    0x00f845c7      0xeb000000      0x2404c710      0x080484e0
0x80483f5 <main+33>:    0xffff12e8      0xf84583ff      0xf87d8301      0xb8ea7e09
0x8048405 <main+49>:    0x00000000      0x5914c483      0xfc618d5d      0x909090c3
```

b : un octet
h : demi-mot, soit 2 octets
w : mot, 4 octets
g : double mot (DWORD) dont 8 octects.

Little-endian (« petit-boutiste ») : octet le moins significatif en premier.
Big-endian : l'inverse.

Affiche mémoire sous forme d'instructions assembler :

```
(gdb) x/i $eip
0x80483e5 <main+17>:    mov    DWORD PTR [ebp-0x8],0x0
(gdb) x/3i $eip
0x80483e5 <main+17>:    mov    DWORD PTR [ebp-0x8],0x0
0x80483ec <main+24>:    jmp    0x80483fe <main+42>
0x80483ee <main+26>:    mov    DWORD PTR [esp],0x80484e0
```

Instruction suivante :

```
(gdb) nexti
```

x/2ii

Utilitaire : pcalc (perl ?)

Fonction C : sizeof()

Pointeur noté : *mon_pointeur

(gdb)x/s [adresse ou variable] --> affiche en ascii
exemple :
(gdb) x/xw pointer
0xbfdb0ad8: 0x6c6c6548
(gdb) x/s pointer
0xbfdb0ad8: "Hello, world!\n"

« & » = adresse-de. Retourne adresse du pointeur.

(gdb) x/xw &pointer
0xbfdb0aec: 0xbfdb0ad8
(gdb) print &pointer
$1 = (char **) 0xbfdb0aec
(gdb) print pointer
$2 = 0xbfdb0ad8 "Hello, world!\n"

On peut suivre un programme en cours d'exécution (--pid=) ainsi que des processus enfants (--pid= et « set follow-fork-mode child ».

Les chaines de format :

- Printf :
  %d : décimal
  %u : decimal unsigned
  %x : hexadecimal
  %s : chaine de caractères
  %n : nombre d'octets écrits jusque-là

- atoi (ASCII to integer)

- Variable statique : initialisée une seule fois. Conserve sa valeur dans le contexte d'une fonction… C'est très intéressant. Cf l'exemple du bouquin. C'est assez clair.

En fait, c'est toujours la même adresse mémoire mais sa valeur d'affectation change. J'adore quand je saisie ce genre de phrase. Je suis certain de tout comprendre dans un mois ou deux.

Note : les adresses mémoires des variables locales sont assez élevées, celle des variables statiques assez basses. C'est super.

Segmentation de la mémoire :

5 segments :
- Texte (text) :
Aussi appelé « segment de code ». Contient l'instruction en langage machine. Read only et partagé. Taille figée.
- Données(datas) : contient variables globales et statiques initialisées. Ecriture autorisée. Taille fixe.
- Bss : variables non initialisées. Ecriture autorisée. Taille fixe.
- Tas (heap) : manipulable . Taille variable dynamiquement.
- Pile (stack) : taille variable. Stockage temporaire pour variables locales et contexte d'appel des fonctions. C'est ce qui est montré par la commande 'bt' de gdb. Permet de mémoriser les variables passées. FI-LO. Image : collier de perles. Push = ajouter. Pop : retirer.

p80

Accès aux fichiers

open(), read(), write(), close()

Chapitre 2 : exploitation

Blague de la boite de chocolat. A replacer absolument !

Erreur classique : décalage de un (*off by one*) : exemple poteaux de clôture. « si clôture sur 100 mètres, poteau tous les 10 mètres, combien de poteaux ? » (11). Aussi appelée « erreur des piquets de clôture (*fencepost*).

Autre erreur : conversion unicode.

Introduction de code dans la mémoire = exécution d'un code arbitraire.

Débordement maitrisé permet d'exécuter un code particulier, comme d'obtenir un shell root. Whéé é !

Gdb> p adresse2 – adresse1 = distance entre les deux adresses.

Password = cryptage sens sens unique, chaine + « salt » (valeur arbitraire faisant varier la génération du mdp. La valeur du « salt » est toujours placée au début de la chaine codée.

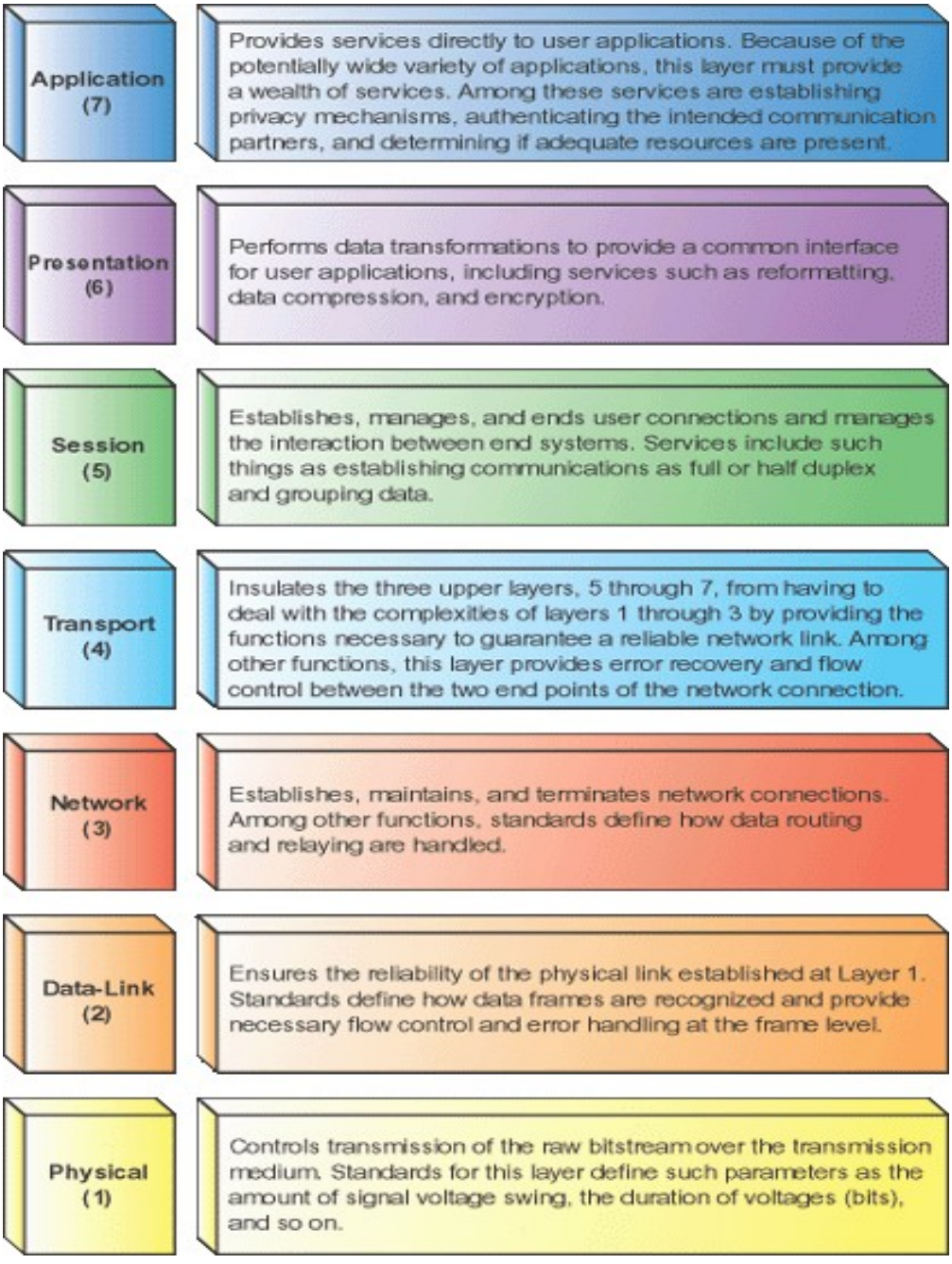
Exemple :
```
 perl -e 'print crypt("password","AA   "). "\n"'
AA6tQYSfGxd/A

perl -e 'print crypt("password","BB"). "\n"'
```

BBUTgMrYjsT0.

Chapitre 3 : Réseau

Modèle OSI :
1- Couche physique (connexion, échange du flux brut)
2- couche liaison de données (transfert de données entre deux points)
3- couche réseau (intermédiaire. Liaison entre couches inférieures et supérieures. Fonction d'adressage et de routage).
4- Couche transport : transfert transparent des données. Fiabilité et efficacité de la transmission.
5- Couche session : établissement et maintient des connexions entre les applications réseaux.
6- couche présentation : présentation des données aux applications. Chiffrement et compression.
7- couche application : en charge des exigences de l'application.



Encapsulation : en-tête + corps.

Socket = « bout » de la connexion. Son type détermine la structure de la couche transport (4).

Streams (« communication téléphonique »). Ex : TCP pour web, mail...
Datagrams (« poste »): ex : UDP. Ex : streaming video, jeu en ligne...

Dans le « C », les sockets sont gérés plus ou moins comme des fichiers.
Prototypes des fonctions dans /usr/include/sys/socket.h

Exemple de fonctions : socket, connect, accept, bind...

Les protocoles et les types de sockets sont définis dans :
/usr/include/bits/socket.h

Utilitaires : tcpdump, dsniff

spoofing :
utilitaire : nemesis spoof suites
SHELL> nemesis : affiche options
SHELL> nemesis [option] help

Scanning :
utilitaire : nmap
Furtif : nmap -sS 192.168.0.3
FIN (paquets FIN à « 1 ») : nmap -sF 192.168.0.3
X-MAS (paquet FIN, PUSH et URG à « 1 ») : nmap -sX 192.168.0.3
NULL (tous les indicateurs TCP à NULL) : nmap -sN 192.168.0.3

Utiliser des leurres :
Attention : les adresses doivent exister sur le réseau.
nmap -D 192.168.0.1,192.168.0.2 192.168.0.3

Scan passif :
On forge des paquets IP avec l'adresse d'un interface inactif. Celui-ci reçoit ou pas une réponse de la cible.
Si réponse : inactif renvoi un paquet RST. Du coup, il incrémente son identifiant IP et cela nous permet de savoir qu'il a répondu, donc de déduire que le port de la cible est ouvert.
Sinon, pas d'incrément donc on déduit le port fermé.

Avec nmap, la commande est :
nmap -sI hote_inactif ip_cible

Défense :
Pour les scans FIN, Null et X-mas, on peut bloquer l'envoi de RST au niveau du noyau. Si si, c'est facile...
Cf bouquin p 282 car c'est intéressant mais compliqué.

Chapitre 5 : Shellcode

Strace

... pour le reste j'ai rien lu : trop compliqué.

Chapitre 6 : Contre-mesures

« Daemon » : origine 1960. Ref. à expérience du physicien James Maxwell : être doté de pouvoir surnaturel pouvant effectuer des tâches difficiles sans effort, transgressant apparemment le second principe de la thermodynamique. Oui, je répète tout cela comme un perroquet.

D'ailleurs, je vais laisser ce chapitre pour y revenir d'ici un an ou deux, mettons...

Cela dit, on y trouve les explications pour créer un daemon, donc c'est intéressant.

Chapitre 7 (on en vois le bout !)

Cf Claude Shannon

Système cryptographique inconditionnellement sûr : ne peut être cassé même avec un temps de calcul infini.

Brute force : john

john [fichier mot de passe]

airsnort : attaque Fluher, Mantin et Shamir.

Aircrak-ng

Chapitre 8 : conclusion

http://www.phrack.com/