

## Sécurité des Réseaux

2009

Corrigé n° 3

### Solution 1: L'attaque de Kevin Mitnick

Cet exercice est fondé sur les traces originales de l'attaque de Kevin Mitnick contre Tsutomu Shimomura. L'attaque nécessite plusieurs étapes. Premièrement, l'attaquant doit choisir sa victime (un serveur). Ensuite, il doit trouver une machine à laquelle le serveur accorde sa confiance. Le but est alors de se faire passer pour cette machine. Pour cela, la machine autorisée est paralysée (afin qu'elle ne puisse pas réagir) et les numéros de séquence du serveur sont analysés. Une connexion utilisant des paquets forgés par l'attaquant est alors initiée avec le serveur, avec des numéros de séquence devinés. Si la connexion est établie, l'attaquant peut alors envoyer des paquets de données à sa cible. Regardons plus en détail notre exemple.

Dans une première phase, il est important que Kevin paralyse Alice pour que celle-ci ne puisse pas répondre à Bob. Pour cela, il pratique des connexions en masse (*Syn flooding*) à partir d'une machine complice, Mallory. Quand une connexion est demandée avec le flag SYN activé, le récepteur renvoie un SYN-ACK et attend le ACK de la part de l'émetteur. Tant que l'émetteur n'a pas renvoyé son ACK, la connexion n'est pas établie. Le nombre de connexion en attente d'être établie est cependant limité par le système d'exploitation. Si cette limite est atteinte, les futures connexions TCP sont tout simplement ignorées jusqu'à ce que des connexions en attente soient établies. Mallory envoie ainsi plusieurs requêtes SYN sur le port 513 d'Alice (port du login). Ensuite, Kevin doit déterminer les numéros de séquence utilisés par Bob. Pour cela, il va se connecter sur le port 514 de Bob et analyser les paquets qui transitent. Ce processus est répété plusieurs fois et à chaque fois on observe le numéro de séquence utilisé par Bob de façon à déterminer l'incrément utilisé, en l'occurrence 128000. Kevin peut maintenant se faire passer pour Alice :

étape	nom	action	nom	trace
1	Kevin	--SYN-->	Bob	14:18:36.24 alice.513 > bob.514: S 1382727010
2	Alice	<--SYN-ACK--	Bob	n'est pas observée par Kevin
3	Kevin	--ACK-->	Bob	14:18:36.75 alice.513 > bob.514: . ack 2024384001

Table 1: Three-way handshake entre Kevin et Bob lors d'une attaque de spoofing aveugle

1. Kevin simule l'adresse IP de la personne de confiance (Alice) et envoie sa connexion sur le port 514 de Bob.
2. Bob répond à Alice avec son numéro de séquence, cette réponse étant invisible pour Kevin. Comme Alice est débordée par les tentatives de connexions de Mallory, elle ignore le paquet au lieu de renvoyer un RST comme elle devrait le faire pour réinitialiser la connexion.

3. Kevin acquitte le paquet de Bob avec le numéro de séquence correct, soit 2024384001, car  $2024256000 + 128000 + 1 = 2024384001$ . Il peut maintenant envoyer ses données (en l'occurrence, il s'agissait de rajouter des lignes dans un fichier configuration du service `rsh`, permettant ainsi à Kevin Mitnick d'avoir accès à la machine cible).

Après l'envoi des données, Kevin, sous l'identité d'Alice, referme correctement la connexion avec Bob. Puis, sous l'identité de Mallory, il met un terme à l'attente d'Alice en envoyant un `RST` pour toutes les demandes de connexion qu'il avait effectuées.

La plupart des implémentations modernes de piles TCP/IP résistent à cette attaque. Tout d'abord, un tampon circulaire peut être utilisé pour stocker les connexions en attente : dès que le tampon est plein, les connexions les plus anciennes sont éliminées. Ensuite, les numéros de séquences initiaux sont générés pseudo-aléatoirement, de façon à ce qu'il soit difficile de les deviner.

### **Solution 2: Attaque par dénis de service**

1. Il suffit à Christophe Pirate de forger des requêtes (sous forme de paquets UDP) de transfert de zone à `dns.machinchose.com` possédant l'adresse IP de `pauvre.victime.com` comme adresse source. Le serveur DNS va ainsi envoyer les réponses directement à `pauvre.victime.com`.

2. Christophe Pirate parviendra donc à générer du trafic inutile utilisant une bande passante égale à  $256 \times \frac{745}{27} \approx 7$  Mbps chez `pauvre.victime.com`, ce qui est environ 27,6 fois plus que celle dont il dispose lui-même.

### **Solution 3: Rappel sur les pointeurs en C**

Les caractères 'A', 'B', 'C' et 'D' sont dans un premier temps stockés respectivement dans les quatre premières cases du tableau `buffer`. On stocke ensuite dans la variable `ptr` l'adresse de `buffer + 2`, c'est-à-dire l'adresse de `buffer` à laquelle on ajoute la taille de l'espace mémoire occupé par deux cases du tableau. Dans l'espace mémoire pointé par `ptr`, on met le caractère 'Z'. La troisième case du tableau devient donc 'Z'. Finalement, le programme affiche : " A B Z D ".

#### Solution 4: Modification d'adresse dans la pile

1. Le programme en C donné dans cet exercice affiche " 0 " et non pas " 1 " comme on pourrait le penser en ne regardant que la fonction `main`. L'instruction `x = 1` a été sautée car la fonction `function` modifie son adresse de retour.

c
b
a
ret
sfp
grade[3]
grade[2]
grade[1]
grade[0]
class[3]
class[2]
class[1]
class[0]

L'instruction `class[9] += 7` est la base de cet exploit<sup>1</sup>.

`grade` et `class` sont deux buffers de 4 entiers qui se trouvent au sommet de la pile (voir ci-dessus). L'expression `class[9]` désigne le dixième entier du buffer `class` (le premier se trouvant à `class[0]`). Comme le buffer a une longueur de 4 éléments, le dixième élément se trouve au-delà du buffer `class`, au-delà du buffer `grade`, au-delà du `sfp` à l'endroit où est stockée l'adresse de retour `ret`. En modifiant la valeur de `class[9]`, on modifie donc directement l'adresse de retour de la procédure. En l'incrémentant de sept unités, on obtient que les instructions se trouvant dans les sept octets suivant l'adresse de retour originale seront sautées. Ce sont des instructions qui devaient s'exécuter directement après l'appel de la fonction `function`, c'est-à-dire l'instruction `x = 1`;

---

<sup>1</sup>Le mot " exploit " désigne en sécurité informatique l'outil ou la méthode permettant d'exploiter une faille.

Comment a-t-on pu déterminer qu'il fallait incrémenter l'adresse de retour de 7 pour sauter l'opération  $x = 1$  ? On peut par exemple utiliser le débogueur GDB (*Gnu DeBugger*) pour désassembler le programme (commande *disassemble*). Le résultat dépend du processeur, du système d'exploitation et du compilateur utilisés :

```

gdb code-adresse
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) disassemble main
Dump of assembler code for function main:
0x080483d5 <main+0>:  lea    0x4(%esp),%ecx
0x080483d9 <main+4>:  and    $0xffffffff0,%esp
0x080483dc <main+7>:  pushl  -0x4(%ecx)
0x080483df <main+10>: push   %ebp
0x080483e0 <main+11>: mov    %esp,%ebp
0x080483e2 <main+13>: push  %ecx
0x080483e3 <main+14>: sub   $0x24,%esp
0x080483e6 <main+17>: movl  $0x0,-0x8(%ebp)           // x = 0;
0x080483ed <main+24>: movl  $0x3,0x8(%esp)           // push c
0x080483f5 <main+32>: movl  $0x2,0x4(%esp)           // push b
0x080483fd <main+40>: movl  $0x1,(%esp)              // push a
0x08048404 <main+47>: call  0x80483c4 <function>     // appel de fonction
0x08048409 <main+52>: movl  $0x1,-0x8(%ebp)           // x = 1; <-- 7 bytes
0x08048410 <main+59>: mov   -0x8(%ebp),%eax
0x08048413 <main+62>: mov   %eax,0x4(%esp)           // push x
0x08048417 <main+66>: movl  $0x80484f0,(%esp)        // push "%d\n"
0x0804841e <main+73>: call  0x80482f8 <printf@plt>    // appel de printf
0x08048423 <main+78>: add   $0x24,%esp
0x08048426 <main+81>: pop   %ecx
0x08048427 <main+82>: pop   %ebp
0x08048428 <main+83>: lea  -0x4(%ecx),%esp
0x0804842b <main+86>: ret
End of assembler dump.

```

On voit que l'instruction  $x = 1$ ; se trouve à l'adresse `main+52` et que la prochaine se trouve à `main+59`. On peut donc la sauter en incrémentant l'adresse de retour de 7.