Virus et exploits, sécurité vs puissance d'expression

Les sources de ce TP sont sur `~sthibault/TP6`

1 (non-)Sécurité des langages

1.1 SQL

Lisez le strip `http://xkcd.com/327/` Que s'est-il passé ?
On peut aussi essayer de faire crasher les radars, googlez "sql injection radar".

1.2 shell

Observez le fichier `test.sh` : comment le détourner pour passer une option `-exec` à `find` ? (ce qui permet donc de faire faire ce que l'on veut au script juste en passant un paramètre !) Comment le corriger pour éviter ce détournement ?

1.3 TeX

Une application web veut produire des documents pdf automatiquement à partir de données fournies par l'utilisateur, en utilisant LaTeX. À quels caractères faudra-t-il faire attention ?

1.4 html

Dans un webforum, que pourrait faire un utilisateur malveillant en mettant du code html dans ses posts ?
Cela s'appelle de manière générale le *cross-site scripting*, qui consiste à injecter du code malveillant dans un site web. Par exemple, allez sur
`http://fr.wikipedia.org/wiki/Wikipédia:Bac_à_sable`
Cliquez sur modifier, ajoutez en tête de page `test <b>test2</b>` et cliquez sur prévisualiser. Observez que votre texte apparaît bien, et surtout, la balise `<b>` est interprétée

par votre navigateur pour mettre le texte en gras ! Pire, lorsque l'on clique sur Enregistrer, ceci est alors stocké sur le site, et donc interprété par tout navigateur ouvrant cette page !

En pratique, wikipedia se protège en pré-interprétant lui-même ce qui est posté, pour ne laisser passer que les balises inoffensives : essayez d'ajouter

`<a href=www.enseirb.fr>test</a>`

Constatez que c'est affiché tel quel. Ouvrez le code source de la page, pour y voir ce que le moteur wikipedia a enregistré à la place.

Imaginez ce que l'on pourrait faire si l'on pouvait injecter une balise qui redirige le navigateur vers un autre site web.

Comment exploiter le fait qu'un navigateur a un bug dans son moteur de rendu d'image ?

Lorsqu'aucune protection n'est mise en place, le code malveillant peut par exemple exécuter du code javascript, ce qui lui permet d'accéder à de nombreuses informations du navigateur, notamment les cookies, souvent utilisés pour identifier les sessions ouvertes. Comment l'attaquant peut-il alors se faire passer pour l'utilisateur attaqué ?

1.5 Attention aux printfs

Observez le programme `printf.c`. Essayez de taper `%p %p`, à quoi correspondent les valeurs affichées ?

Il est ainsi possible de récupérer des informations sur un processus : l'adresse de la libc, l'adresse de la pile, etc. qui peuvent servir pour des exploits (car depuis quelques années, pour plus de protection, Linux les place aléatoirement en mémoire).

Essayez `%n`, puis `%p %n`, que se passe-t-il ? (`man 3 printf` et cherchez " n " dedans).

1.6 C

Voici des bouts de codes apparemment corrects, trouvez le problème.

Ceci est un exemple de code d'un démon pouvant être utilisé soit en tant que root, soit en tant qu'utilisateur normal (auquel cas on lui permet de faire beaucoup plus de choses).

```
#include <unistd.h>
int main(void) {
    if (getuid) {
        /* Pas root, le système vérifie pour nous qu'on ne casse pas
 * tout */
        ...
    } else {
        /* root, faire attention à ce que l'on fait ! */
    }
}
```

Ceci montre deux lignes de code qu'un hacker a essayé de faire glisser dans noyau Linux (marquées par des +) :

```
 int clone(int (*fn)(void *), void *child_stack,
           int options, void *arg,...) {
      ...
+     if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+         retval = -EINVAL;
      retval = -ECHILD;
      ...
      return retval;
 }
```

2 Un virus-macro Word

Lisez le fichier `virus.txt`. Il s'agit d'un virus-macro Word tout ce qu'il y a de plus simple. Il faut savoir que les macros suivantes sont exécutées automatiquement par Word :

AutoOpen Lors de l'ouverture d'un document.

AutoClose Lors de la fermeture d'un document.

AutoExec Au démarrage de Word.

AutoExit À la terminaison de Word.

et sont souvent utilisées pour faire des documents dynamiques, effectuant automatiquement certaines tâches telles que l'impression automatique de factures, etc.

Observez la stratégie du virus : lors de l'ouverture de n'importe quel document, le virus essaie soit de se copier dans `normal.dot` (infection d'une machine lorsqu'elle ouvre un document infecté), soit de se copier dans le document ouvert (infection d'un document par une machine infectée). Notez que le virus essaie de se protéger contre un usager qui voudrait supprimer la macro, en désactivant le menu outil. Quelle est la "charge" du virus, c'est-à-dire son action "néfaste" ? Expliquez pourquoi le virus peut donc se propager facilement.

Relevez les fonctionalités "puissantes" de Word qui sont en fait celles qui ont permis d'écrire ce virus.

A priori si LibreOffice fournit une compatibilité complète sur les macros, il devrait être affecté par le virus aussi. Quel détail tout bête fait que même si ça pourrait arriver sous Windows, ce virus tel quel ne pourrait pas fonctionner sous Unix ? C'est un exemple de détail montrant que la biodiversité permet de mieux résister aux infections virales.

3 À propos du mécanisme d'infection

Ce n'est bien sûr qu'un exemple de virus, d'autres virus ont d'autres stratégies d'infection : des virus de mail tels qu'*ILOVEYOU* (année 2000) incitent l'utilisateur à ouvrir la pièce jointe qui est en fait un programme qui réenvoie le mail à tous les correspondant de l'utilisateur. La méfiance est d'autant moins grande que le mail est reçu depuis quelqu'un que l'on connait a priori (puisque l'on était dans son carnet d'adresses). De plus les logiciels de mail cachent souvent par défaut l'extension des documents, masquant ainsi qu'il s'agit en fait d'un programme...

Comme autre exemple, la fonctionalité d'*autorun* activée par défaut depuis des années sous windows, c'est-à-dire, lorsqu'une clé USB ou un CD est inséré dans une machine, d'exécuter automatiquement le fichier *autorun.inf* s'il existe, permet de tenter de prendre contrôle de machines cibles : il suffit de laisser traîner une clé USB dans les bureaux de l'entreprise visée, et de laisser la curiosité agir...

On voit ainsi sur quelques exemples le problème d'équilibre entre les fonctionalités avancées de traitements automatiques, la volonté de masquer tous les détails de la machine pour ne pas avoir à former l'utilisateur à les comprendre, et la sécurité.

Dans un autre registre, lisez les pages "informations" et "dangers" de `www.hoaxbuster.com` . Il s'agit en fait de nouveau d'un mécanisme d'infection.

Après, il y a des virus qu'on s'amuse même à propager en toute conscience :

```
I am the "ILOVEGNU" signature virus. Just copy me to your signature.
This email was infected under the terms of the GNU General Public License
```

4 Un petit buffer overflow en action

Observez le programme `anodin.c` qui ne fait que lire un entier (ainsi qu'afficher l'adresse du buffer, un vrai programme ne le ferait pas, mais on a vu à une section précédente qu'on pourrait récupérer cette information par ailleurs).

Remarquez que la compilation affiche un avertissement contre l'utilisation de la fonction `gets()`, pourquoi cette fonction n'est pas sûre ?

Examinez `exploit.c`, et surtout `exploit.s`. Essayez de comprendre :)

L'idée est qu'`anodin` pourrait être un serveur, qui demande une commande sur une socket, et qu'`exploit` est le programme d'un client qui se connecte au serveur. Il suffit à `exploit` de savoir où est situé le tableau `buf` et de connaître sa taille (128), pour pouvoir exploiter le débordement de tampon : on écrase l'adresse de retour pour brancher dans le buffer lui-même, qui exécute un shell, qui accepte alors les commandes écrites par `exploit` !

Pour le tester, lancez `./exploit | ./anodin` et copiez-collez l'adresse de `buf` affichée par `anodin`, pour la donner à `exploit` pour qu'il sache où écrire. On se retrouve avec un fichier `ahah` dans le home de l'utilisateur qui a lancé `anodin` !

5 Les compilateurs contre-attaquent

Étudiez
`http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf`