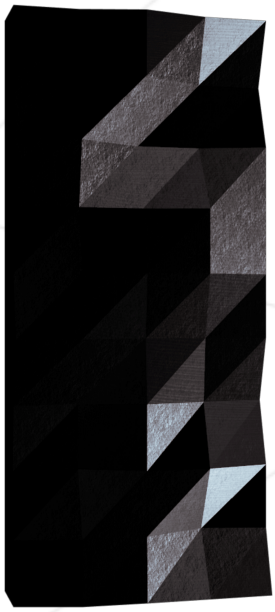

Project UNIX

Famine

42 Staff pedago@staff.42.fr

Résumé: Ce projet consiste à coder votre premier virus.

Table des matières

Chapitre I

Préambule

$$
\begin{aligned}
&\forall M\ \forall V\ \ (M,V) \in \mathcal{V} \Leftrightarrow [V \subset \mathbb{I}^*] \text{ et } [M \in \mathcal{M}] \text{ et} \\
&\quad [\forall v \in V\ \ [\forall H_M\ [\forall t\ \forall j \in \mathbb{N} \\
&\qquad [\ 1.\ P_M(t) = j \text{ et} \\
&\qquad\ \ 2.\ \$_M(t) = \$_M(0) \text{ et} \\
&\qquad\ \ 3.\ (\square_M(t,j), \ldots, \square_M(t, j + |v| - 1)) = v] \\
&\quad \Rightarrow \qquad [\exists v' \in V [\exists t', t'', j' \in \mathbb{N} \text{ et } t' > t \\
&\qquad [\ 1.\ [[(j' + |v'|) \leq j] \text{ ou } [(j + |v|) \leq j']] \\
&\qquad\ \ 2.\ (\square_M(t', j'), \ldots, \square_M(t', j' + |v'| - 1)) = v' \text{ et} \\
&\qquad\ \ 3.\ [\exists t'' \text{ tel que } [t < t'' < t'] \text{ et} \\
&\qquad\quad [P_M(t'') \in \{j', \ldots, j' + |v'| - 1\}] \\
&\quad]]]]]]]]
\end{aligned}
$$

Chapitre II

Introduction

Un virus informatique est un automate autoréplicatif à la base non malveillant, mais aujourd'hui souvent additionné de code malveillant (donc classifié comme logiciel malveillant), conçu pour se propager à d'autres ordinateurs en s'insérant dans des logiciels l'egitimes, appelés « hôtes ». Il peut perturber plus ou moins gravement le fonctionnement de lórdinateur infecté. Il peut se répandre par tout moyen d'échange de données numériques comme les réseaux informatiques et les cédéroms, les clefs USB, etc.

Son appellation provient d'une analogie avec le virus biologique puisqu'il présente des similitudes dans sa manière de se propager en utilisant les facultés de reproduction de la cellule hôte. On attribue le terme de « virus informatique » à l'informaticien et spécialiste en biologie moléculaire Leonard Adleman.

Chapitre III

Objectifs

Maintenant que vous avez une bonne conception des programmes de type auto-répliquant, nous allons voir un usage possible avec la création d'un virus basique qui aura pour but de simplement laisser une trace dans certains fichier contenu dans un dossier spécifique.

Puisque vous avez appris à ajouter des données exécutables dans un binaire donné, ce petit virus ne devrait pas vous poser de problèmes.

Chapitre IV

Partie obligatoire

Famine est un binaire de votre conception qui devra modifier un ou plusieurs autre(s) binaire(s) pour y appliquer des fonctionnalités supplémentaires, sans altérer le comportement initial du-dit binaire. Pour le coup, on va se limiter à ajouter une "signature" à ce binaire et rien d'autre.
Famine devra appliquer cette "signature" à tous les binaires présents dans un dossier temporaire spécifique. La "signature" est symbolisé par une phrase contenant vos logins respectifs qui pourrait ressembler à ceci :

```
Famine version 1.0 (c)oded by <first-login>-<second-login>
```

Comme votre programme est censé être un virus, vous comprendrez qu'un minimum de discrétion est de mise. De ce fait, **AUCUN OUTPUT NE SERA FAIT** lors de l'exécution de Famine lorsque vous le rendrez, que ce soit dans un fichier de log ou sur une sortie quelconque et ce même en cas de plantage.

Ainsi pour résumer :

- L'exécutable devra se nommer `Famine`.
- Cet exécutable est codé en assembleur ou en C et rien d'autre.
- Votre programme ne va rien afficher sur la sortie standard ni d'erreur.
- Vous devrez **OBLIGATOIREMENT** travailler dans une VM.
- Le système d'exploitation cible est libre. Toutefois, vous devrez amenager lors de votre correction une VM appropriée.
- Votre programme va devoir agir sur les dossiers /tmp/test et /tmp/test2 ou équivalent en fonction de votre système d'exploitation cible, et UNIQUEMENT dans ces dossiers. Le contrôle de la propagation de votre programme est de votre responsabilité
- **ATTENTION !** Une seule infection sur ledit binaire est possible.
- Les infections se feront dans un premier temps sur des binaires du type de votre système d'exploitation ayant pour architecture 64 bits.

Chapitre V

Exemple d'utilisation

Voici un exemple d'utilisation possible :

On prepare le terrain :

```
# ls -al ~/famine
total 736
drwxr-xr-x 3 root root   4096 May 24 08:03 .
drwxr-xr-x 5 root root   4096 May 24 07:32 ..
-rwxr-xr-x 1 root root 744284 May 24 08:03 Famine
```

On crée sample.c pour nos tests :

```
# nl sample.c
1 #include <stdio.h>
2 int
3 main(void) {
4       printf("Hello, World!\n");
5       return 0;
6 }
# gcc -m64 ~/Virus/sample/sample.c
#
```

On copie nos binaires (tests + ls) pour nos tests :

```
# cp ~/Virus/sample/sample /tmp/test2/.
# ls -al /tmp/test
total 16
drwxr-xr-x  2 root root 4096 May 24 08:07 .
drwxrwxrwt 13 root root 4096 May 24 08:08 ..
-rwxr-xr-x  1 root root 6712 May 24 08:11 sample
# /tmp/test/sample
Hello, World!
# file /tmp/test/sample
/tmp/test/sample: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
    lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=938[...]10b, not stripped
# strings /tmp/test/sample | grep "wandre"
# cp /bin/ls /tmp/test2/
# ls -al /tmp/test2
total 132
drwxr-xr-x  2 root root   4096 May 24 08:11 .
drwxrwxrwt 14 root root   4096 May 24 08:11 ..
-rwxr-xr-x  1 root root 126480 May 24 08:12 ls
# file /tmp/test2/ls
/ tmp/test2/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
    lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=67e[...]281, stripped
#
```

On lance `Famine` et on voit le resultat :

```
# ./Famine
# strings /tmp/test/sample | grep "wandre"
Famine version 1.0 (c)oded oct-2015 by wandre
# /tmp/test/sample
Hello, World!
# strings /tmp/test2/ls | grep "wandre"
Famine version 1.0 (c)oded oct-2015 by wandre
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x  2 root root   4096 May 24 08:11 .
drwxrwxrwt 14 root root   4096 May 24 08:17 ..
-rwxr-xr-x  1 root root 126480 May 24 08:12 ls
# gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample
# ls -al /tmp/test
total 16
drwxr-xr-x  2 root root 4096 May 24 08:07 .
drwxrwxrwt 13 root root 4096 May 24 08:08 ..
-rwxr-xr-x  1 root root 6712 May 24 08:12 sample
# /tmp/test/sample
Hello, World!
# file /tmp/test/sample
/tmp/test/sample: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
    lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=93866
    e4ce7a2fe18506bd6c6218e413156a8d10b, not stripped
# strings /tmp/test/sample | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x  2 root root   4096 May 24 08:11 .
drwxrwxrwt 14 root root   4096 May 24 08:17 ..
-rwxr-xr-x  1 root root 126480 May 24 08:12 ls
# strings /tmp/test/sample | grep "wandre"
Famine version 1.0 (c)oded oct-2015 by wandre
#
```

Chapitre VI

Partie bonus

Les bonus ne seront comptabilisés que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc ... Concrètement, cela signifie que si votre partie obligatoire n'est pas validée, vos bonus seront intégralement IGNORÉS.

Des idées de bonus :

- Pouvoir infecter les binaires d'architecture 32 bits.
- Ajout de fonctionalités à notre virus (lancement uniquement sous certaines conditions par exemple, ou encore lancement au boot du système d'exploitation...)
- Pouvoir infécter tous les fichiers en partant de la racine de votre système d'exploitation de manière récursive.

Vous devez optimiser cette partie en executant les binaires infectés..

- Permettre une infection sur des fichiers non binaires.
- Utilisation de méthodes de type packing sur le virus directement dont le but sera de rendre le binaire le plus léger possible.

Chapitre VII

Rendu et peer-évaluation

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.

- Vous devez gérer les erreurs de façon raisonnées. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc).

- Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.

- Vous devez être sous une VM durant votre correction. Pour info, le barême a été fait avec une Debian 7.0 stable 64 bits.

- Vous être libre d'utiliser ce dont vous avez besoin, dans la limite des bibliothèques faisant le travail pour vous, ce qui correspondrait à un cas de triche.

- Vous pouvez poser vos questions sur le forum, Slack...