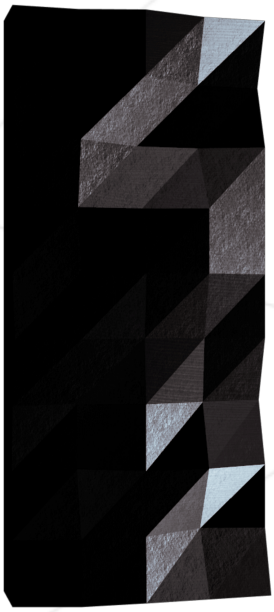

Project UNIX

Death

42 Staff pedago@staff.42.fr

Summary: Ce projet consiste à coder votre dernier virus dit "métamorphe".

Contents

Chapter I

Préambule

Chapter II

Introduction

Si vous êtes inscrit à ce projet, tout d'abord, de la part de tout le staff, Félicitations, vous avez accompli de "grandes choses", mais il vous reste maintenant l'épreuve ultime de votre formation de Virologue.

Après l'oligomorphie avec `Pestilence` et la polymorphie avec `War`, vous allez entreprendre un voyage initiatique dans la voie de la métamorphie. La caractéristique principale d'un virus métamorphique est sa capacité à modifier sa structure interne ainsi que les instructions qui le composent.

Le métamorphisme est un mécanisme de défense utilisé par différents programmes malveillants afin d'éviter leur reconnaissance et leur détection par des programmes de contrôle et de protection tel que les logiciels antivirus.

Un virus dit métamorphique ne contient pas de décrypteur ni un corps viral constant, mais est capable de créer de nouvelles générations de lui-même différentes à chaque réplication tout en évitant qu'une génération soit trop ressemblante avec celle qui la précède.

Chapter III

Objectifs

Nous voila enfin dans la partie vraiment fun ! Via les virus que vous avez développés, vous avez pu découvrir des méthodes d'obfuscation avec de la programation polymorphique... Mais, il manque encore des choses assez fun à découvrir. Par ce projet, vous allez devoir penser différement. Votre méthode pour trouver des solutions aux problématiques en général va d'ailleurs en prendre un coup ici.

En effet, en plus de toutes les fonctionalités que vous avez du implementer dans les précédents projets, vous allez par ce projet devoir créer un code dont la structure interne va partiellement évoluer. Pour ce faire le passage vers les langage machines sera obligatoire. Vous devrez donc simplement développer un dernier programme avec plein de propriétées que vous avez déjà vu. Mais en plus de tout cela on va rendre notre programme vraiment plus intéréssant.

Ce projet qu'on soit d'accord n'est pas forcément prévu pour tout le monde au vu de la difficultée de celui-ci. Donc si vous souhaitez vous lancer dans ce projet il faudra vous motivez et surtout ne pas abandonner le résultat vaut le coup :)

Chapter IV

Partie obligatoire

`Death` est un binaire de votre conception qui devra:

- comme `Famine`, infecter des binaires présents dans 2 dossiers spécifiques et y appliquer une signature, sans altérer le fonctionnement du-dit binaire.
- comme `Pestilence`, ne pas déclencher la routine d'infection si un processus ciblé est en cours d'exécution, si le programme est lancé via un debugueur quelconque, et présenter une partie de la routine d'infection de manière obfusquée.
- comme `War`, embarquer un FINGERPRINT évoluant au fil des exécutions des routines d'infection.

La signature en question devra d'ailleurs ressembler à quelque chose de ce style :

```
D34TH version 1.0 (c)oded by <first-login> - <second-login> - [FINGERPRINT]
```

Et maintenant, vient la partie WTF de ce sujet, et prenez bien le temps de lire plusieurs fois cette partie: Vous devrez faire en sorte que votre programme ne soit `structurellement` jamais le même, une fois que ce dernier a fini de s'exécuter.

Pour se faire, il faudra simplement apprendre à re-développer votre virus pour que la partie infectueuse puisse évoluer. On doit pouvoir infecter 2 fichiers identiques en ayant pour obligation de voir une vraie différence au niveau du code machine. Les programmes inféctées doivent toujours fonctionner sans soucis bien entendu.

Consignes générales :

- L'exécutable devra se nommer `Death`.
- Cet exécutable est codé en assembleur, en C ou en C++ et rien d'autre. On tolère le JAVA pour ce projet, mais c'est tout.
- 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 comme toujours libre de choix. Toutefois, vous devrez aménager 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, pas plus.
- Les infections se feront dans un premier temps sur des binaires du type de votre système d'exploitation ayant pour architecture 64 bits.

Chapter V

Exemple d'utilisation

Voici un exemple d'utilisation possible:

On prepare le terrain:

```
# ls -al ~/Death
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 Death
```

On crée sample.c pour nos tests :

```
# nl sample.c
1 #include <stdio.h>
2 int
3 main(void) {
4       printf("This is hard...\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 `Death` sans le processus "test" et on voit le resultat:

```
# pgrep "test"
# ./Death
# strings /tmp/test/sample | grep "wandre"
virus.custom version 1.3 (c)oded may-2017 by wandre - 42424242
# /tmp/test/sample
This is hard..
# strings /tmp/test2/ls | grep "wandre"
virus.custom version 1.3 (c)oded may-2017 by wandre - 43434343
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root  4096 May 4 12:13 .
drwxrwxrwt 14 root root 4096 May 4 12:11 ..
-rwxr-xr-x 1 root root xxxxxx May 4 12:19 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 4 12:13 .
drwxrwxrwt 13 root root 4096 May 4 12:11 ..
-rwxr-xr-x 1 root root xxxx May 4 12:19 sample
# /tmp/test/sample
This is hard..
# 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]=
     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, not stripped
# strings /tmp/test/sample | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root  4096 May 3 12:03 .
drwxrwxrwt 14 root root 4096 May 3 12:01 ..
-rwxr-xr-x 1 root root xxxxxx May 3 12:20 ls
# strings /tmp/test/sample | grep "wandre"
virus.custom version 1.3 (c)oded may-2017 by wandre - 444444
#
```

On voit clairement une évolution de la "signature". Il faut bien entendu faire en sorte que la modification soit visible clairement. La partie de la signature qui doit évoluer doit être controlée. Si celle-ci est aléatoire la note sera en conséquence.. Une double infection ne doit pas être possible je rappel. Pour cette partie je vous encourage grandement à développer votre logique en assembleur. Il existe des méthodes pour parvenir à ce résultat en C/C++ mais à mes yeux ca risque de compliquer fortement la réalisation de ce projet.

On lance `Death` avec le processus "test" ainsi que l'environement initial et on voit le resultat:

```
# pgrep "test"
57452
# ./Death
# strings /tmp/test/sample | grep "wandre"
# /tmp/test/sample
This is hard..
# strings /tmp/test2/ls | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root  4096 May 4 12:03 .
drwxrwxrwt 14 root root 4096 May 4 12:01 ..
-rwxr-xr-x 1 root root xxxxxx May 4 12:21 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 3 12:03 .
drwxrwxrwt 13 root root 4096 May 3 12:01 ..
-rwxr-xr-x 1 root root xxxx May 3 12:21 sample
# /tmp/test/sample
This is hard..
# 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]=
     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, not stripped
# strings /tmp/test/sample | grep "wandre"
# /tmp/test2/ls -la /tmp/test2/
total 132
drwxr-xr-x 2 root root  4096 May 3 12:03 .
drwxrwxrwt 14 root root 4096 May 3 12:01 ..
-rwxr-xr-x 1 root root xxxxxx May 3 12:23 ls
# strings /tmp/test/sample | grep "wandre"
#
```

On va tenter de lancer `Death` dans un debugeur je vais utiliser gdb. J'ai mis un petit message pour que ce soit plus clair:

```
# gdb -q ./Death
(gdb) run
Starting program: /root/Death
DEBUGGING..
[Inferior 1 (process 2683) exited with code 01]
# strings /tmp/test/sample | grep "wandre"
# /tmp/test/sample
This is hard..
# strings /tmp/test2/ls | grep "wandre"
#
```

On va enfin voir la partie metamorphique pour se faire on va simplement prendre 2 sample identiques que l'on va nommé assez logiquement sample1 puis sample2 et on va faire un diff pour contrer le nombre de différence via un simple diff. Par soucis de clareté je vais faire de manière à e que ce soit le plus lisible.

```
# nl sample.c
1 #include <stdio.h>
2 int
3 main(void) {
4       printf("This is hard..\n");
5       return 0;
6 }
# gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample1
# cp /tmp/test/sample1 /tmp/test/sample2
# /tmp/test/sample1
This is hard..
# /tmp/test/sample2
This is hard..
# objdump -D /tmp/test/sample1 > samp ; objdump -D /tmp/test/sample2 > samp2 ; diff -y --suppress-
    common-lines samp samp2 | grep '^' | wc -l
1
```

Aucune différence ici au dela du nom du programme en lui même. Le diff simplement montre le nombre de ligne différente entre chaque binaire au niveau du code machine. On va lancer le virus et voir le résultat.

```
# ./Death
# /tmp/test/sample1
This is hard..
# /tmp/test/sample2
This is hard..
# strings /tmp/test/sample1 | grep "wandre"
virus.custom version 1.3 (c)oded may-2017 by wandre - 41414141
# strings /tmp/test/sample2 | grep "wandre"
virus.custom version 1.3 (c)oded may-2017 by wandre - 42424242
# objdump -D /tmp/test/sample1 > samp ; objdump -D /tmp/test/sample2 > samp2 ; diff -y --suppress-
    common-lines samp samp2 | grep '^' | wc -l
165
#
```

On ira pas plus en profondeur, mais vous comprenez le principe hein. Bref, vous avez pas mal de travail devant vous.

Chapter 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.
- 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 inféctés..

- 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.
- Ajouter un métamorphisme total à votre programme (ceci validera tout les bonnus au vu de la difficultée).

Chapter 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.

- Votre routine rendant votre programme "polymorphe" doit être controlé. C'est vraiment important.

- Une explication de votre partie métamorphique sera obligatoire.

- Vous devez gérer les erreurs de façon raisonnée. 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, sur jabber, IRC, slack...