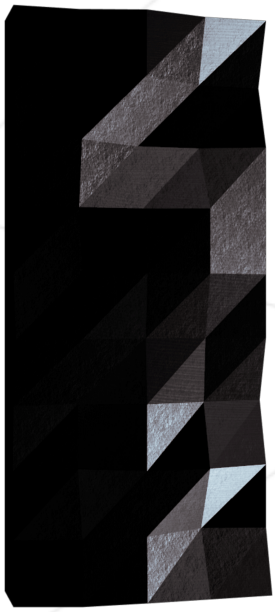

Famine

Project UNIX

Summary: This project is about coding your first virus.

Version: 1

Contents

Chapter I

Foreword

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

Chapter II

Introduction

A computer virus is a non-malicious self-duplicating automaton at the root, but later supplemented with additional malevolent code (therefore qualified as malicious software) designed to spread and infect other computers, sneaking into legitimate software named "hosts". It can disturb more or less severely the affected computer. It may spread by any means of exchange of digital date such as computer networks, CD-ROM, USB flash drives etc.

Its name comes from an analogy with the biological virus since it has similarities in its way of propagating using the reproductive faculties of the host cell. The term "computer virus" was coined by computer scientist and molecular biology specialist Leonard Adleman.

Chapter III

Objectives

Now that you have a better understanding of the self-duplicating programs, we will see a possible use for them with the creation of a basic virus that will have to leave a trace in some files contained in a specific folder..

Since you learned how to add some executable data in a given binary, this little virus should be a piece of cake.

Chapter IV

Mandatory part

Famine is a binary you need to conceive that will modify one or multiple binaries applying some additional functions to it/them, without modifying the original behavior of said binary(ies). So this time, we will limit ourselves to add a "signature" to this binary and nothing else. Famine will have to apply this "signature" to all the binaries present in a specific temporary folder. The "signature" is symbolized with a line containing your specific logins and it could look like that:

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

Since this program is meant to be a virus, you will understand that a very low-profile is required. Therefore, **NO OUTPUT WHATSOEVER WILL BE DONE** when famine is executed when you will return it, none in a log file or in any random exit, and even in case of a crash.

So to summarize:

- The executable must be named `Famine`.
- Your project must be written in C or assembly language nothing else.
- You program will bot display anything neither on the standard output nor on the error output.
- It is **MANDATORY** to work in a VM.
- The target operating system is free. However you will have to prepare an appropriate VM for your p2p.
- Your program will have to act on the /tmp/test and /tmp/test2 folders or equivalent according to your target operating system, and ONLY in those folders. The control of the spreading of your program is your responsibility.
- **WARNING !** Only one infection on said binary is possible.
- Infections will be at first on the binaries of the operating system that have a 64 bits architecture.

Chapter V

Usage example

Here are some possible usage examples:

Prepare the field:

```
# 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
```

We created sample.c for our 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
#
```

We copy the binaries (tests +ls) for our 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
#
```

We run Famine and look at the resuts:

```
# ./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
#
```

Chapter VI

Bonus part

We will look at your bonus part if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management needs to be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

Find below a few ideas of interesting bonuses:

- Being able to infect the 32 bits architecture binaries.
- Add functionalities to your virus (launch only under some specific conditions only for example, or even launch on the boot of the operating system...).
- Being able to infect all the files from the root of your exploitation system in a recursive manner.

You must optimize this part by executing the infected binaries.

- Allow an infection on non-binaries files.
- Use packing type methods directly on the virus whose purpose will be to make the binary as light as possible.

Chapter VII

Submission and peer-evaluation

- Submit your work on your GiT repository as usual. Only the work on your repository will be graded.
- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- You have to be in a VM with a Linux kernel > 3.14. Note that grading was designed on a Debian 7.0 stable 64 bits.
- You are allowed to use what you need within the limit of the libraries doing the work for you, which would be equivalent to cheating..
- You can ask your questions on the forum, on slack...