CS4471 Computer and Network Security

Fall 2014

Project 4: A Simple Virus

1 Overview

In this project, you will implement a simple virus and try to hide its existence. The virus will be implanted by luring a user to execute your infected binary. The `open` and `close` system calls will be modified to try to make the virus invisible to any program that uses `open` to operate against an infected binary.

2 Requirements

The project is comprised of three parts. Each is described below.

2.1 The Virus

The virus spreads from an infected binary. It replicates by "prepending" itself to an uninfected host, as depicted below. When an executable is created from a file with this content, the virus executes and the host is carried along as a payload to be executed by the virus binary at the appropriate time.

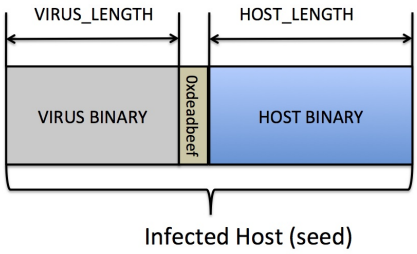

Figure 1: Binary File for Infected Host

The actions of the virus are to:

(*i*) Copy the host program (the last `HOST_LENGTH` bytes of the binary from which the executing process was created) to a file named `/tmp/host.`*`ruid`*, where *`ruid`* is the real (numeric) uid of the executing process. If a file with this name already exists, the virus should attempt to delete the file and exit.[1] (The virus will not spread.) If this file does not exist, the host program is executed and the temporary file is deleted.

(*ii*) Determine whether a file exists that: has the name given by `argv[1]`, is owned and can be written by

[1] This provides a mechanism to prevent the spread of the virus.

the real uid of the executing process, and does **not** have the owner, group or world execute bit set.[2] If so, the virus determines whether or not the file is already infected. If this file exists (with the specified ownership and permissions) and is not infected, the first `VIRUS_LENGTH` bytes of the binary file from which the executing process was created are prepended to the file named by `argv[1]`. If not, the virus exits.

(*iii*) The virus portion of an infected host is separated from the host portion by the four byte value `0xdeadbeef`. The first byte is `0xde`. The virus writes this value between the virus and the host binary. Hence, the size of the virus need not be known at the time of its compilation. The virus code can be modified, recompiled, and prepended (along with the sentinel value) to the host to create a seed binary.

2.2 The Host

You will write a host program that encourages a user to enter the name of a file as its first parameter. The host program should be named `host`. When executed, it should not alter any files nor intrude on the privacy of the user executing the program. The `host` must be able to run as a standalone program, that is, when it is not infected.

2.3 The Seed

The seed binary is created by prepending the virus to the host with the sentinel string `0xdeadbeef` between the virus and the host. (See figure 1.) The seed can be created as follows. In this example, `host` is the host binary, `virus` is the virus binary, and `seed` is the infected host binary.

```
$cp virus seed
$printf '\xde\xad\xbe\xef' >> seed
$cat host >> seed
```

One can put the value `0xdeadbeef` at offset `OFFSET` in file `seed` using the command `printf '\xde\xad\xbe\xef'| dd of=seed bs=1 seek=OFFSET count=4 conv=notrunc`. The size of the seed binary should be the sum of the sizes of the virus and the host plus four (the length of the sentinel value). This can be verified with the following commands.

```
$ls -l virus
$ls -l host
$ls -l seed
```

The `ls -l` command will give the file size in bytes.

One can read the contents of a file in hexadecimal with the command `od`. For example, the command `od -j SKIP -tx1 seed -N LENGTH` will print in hexadecimal the `LENGTH` bytes that follow the initial `SKIP` bytes of the file `seed`.

[2]This requires an infected file to explicitly be made executable before it can be run with the virus attached. This is another measure to prevent inadvertent spread of the virus. Note that fstat (and friends) can be used to determine which permission bits are set.

2.4 The `Open` and `Close` System Calls

You will wrapper the `open` and `close` system calls so that only the uninfected portion of a binary file appears to operations between the open and the corresponding close.

2.4.1 Writing a Wrapper

A system call can be wrapped using the `syscall` system call. The first parameter to this call is a number that identifies the system call to be invoked. Macros defining the system call numbers are in `/usr/include/asm/unistd.h`. The remaining parameters to `syscall` are the parameters for the system call being invoked.

The following code wrappers the `rename` system call to print a message prior to invoking `rename`.

```
#include <asm/unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#include <sys/types.h>

int rename(const char *oldpath, const char *newpath) {

  printf("Renaming <%s> to <%s>\n",oldpath,newpath);
  return(syscall(__NR_rename,oldpath,newpath));
}
```

A program that calls `rename` will by default be linked against the standard C library. In order to use the wrappered version, either the modified version of `rename` must be inserted into the standard C library, or a program must be linked against the object code for the modified system call. You will use the second approach, since you cannot modify the standard C library.

As an example, suppose the code above is placed into a file named `rename.c`. Further suppose the following code is placed into a file named `useRename.c`.

```
int main(int argc, char *argv[])
{
  rename(argv[1],argv[2]);
}
```

Then the following commands will cause the modified version of `rename` to be used.

```
gcc -c rename.c
gcc -o useRename useRename.c rename.o
```

2.4.2 Modification of Open and Close

Your modified version of `open` should determine whether the named file is infected. If so, the virus code should be removed to a file named `/tmp/.pid.fd.ruid`, where `pid` is the process id returned from

`getpid()`, `fd` is the descriptor associated with the file, and `ruid` is the real uid of the executing process. After removal of the virus code, the file from which the binary was created should only contain the host "payload". Any subsequent operations on this file should not see the virus. Your modified `close()` will reinfect the file associated with the input descriptor. [3]

Write a test program that takes the name of a file as an input argument, calls `open` on the file, calls `fstat` on the descriptor returned from open, prints out the file size and first eight bytes of the file in hex, and then calls `close`. Name the test program `tstWrappers.c`.

3 Collaboration Rules and Academic Integrity

Coding for this project is to be performed in pairs. Of course, there are no restrictions on interactions among group members. However, each group must work independently. Individual work will be allowed, but no allowance will be made in the due date or other submission requirements if this option is freely chosen.

Note that writing a virus that performs actions outside the scope of this project will be considered a misuse of University computing resources and is subject to disciplinary action. Each group member is responsible to ensure their virus cannot spread outside their home directory during the development process.

This code need not follow the secure coding guidelines.

4 Submissions

You must prepare a `Makefile` and all necessary source files so that I can simply do a `make` and build the `host`, `virus`, and `seed` binaries, the system call wrappers `open.o` and `close.o` and the `tstWrappers` program. The executables and object code should be located in the top-level directory. (The grader may write his scripts on this assumption.) Link the `tstWrappers` program against your modified versions of open and close. The Makefile should support a `clean` directive that removes all object code and executables.

Submit a file named `bug.tgz` to Canvas. To that end, create a directory called `bug.d` in which your `Makefile` and all required source files will reside. Make `bug.d` your working directory and tar the contents of the directory with "`gtar zcvf bug.tgz *`". Use `submit` to submit the file. The submission is due Thursday, Dec. 11 at 11:59pm. Please remember that no slip days can be used for this homework. beyond Friday Dec. 12 at midnight.

[3] You may need to take care with the version (wrappered or unwrappered) of `open` if it is called in your close routine. For simplicity, you may assume any infected file is opened for both read and write.