Lab 2 - Malware: Viruses, Worms, and UTMs

Created: Fall 2022 (Amir Herzberg), Updated: Spring 2024 (Tim Curry & Yuan Hong)

Monday sections submit by 2/25

Wednesday sections submit by 2/27

See instructions for all labs in Lab 0.

Note: in this lab we use the Rubber Ducky gadget; at the end of this lab, pack it and return to the TA.

Overview

Malicious software, aka Malware, is the most serious threat for most users and organizations. Malware is any software that is designed to benefit its "owner", by running without authorization on a victim's machine (a phone, computer, server, or other device). Malware can be distributed for different goals, such as Denial-of-Service attacks, exposure of sensitive information, unauthorized use of computing resources (e.g., for mining bitcoins), and many other nefarious goals; the functionality of the malware designed to meet these nefarious goals is called the *payload*.

Most malware also has functionality for distributing the malware, which we refer to as *infection* or *propagation.*
Malware is often categorized by its method of infection/propagation; the categories include:

- **Virus:** malware which searches the storage to identify other programs, and then changes them so they will contain a copy of the virus and execute it, in addition to their "real" function. Viruses often infect a specific type of program, such as binary executables, macro files (e.g., of Office), or programs written in a specific language. In this lab you'll write a very simple virus that will infect Python source-code programs.
- **Worm:** malware that searches a network to identify vulnerable machines, and then uses the vulnerability to copy itself to these machines (and run also from there).
- **USB-Transmitted Malware (UTM):** malware which is injected by connecting a rogue device to the USB interface of computer. The rogue device can appear to be a USB memory stick, a USB cable, or a USB charger. After being injected by the USB interface, the malware may further propagate as a worm and/or virus. In fact, many USB memory sticks on the market may be modified to become such rogue devices, using software on a computer to which the USB is connected. In this way, USB-port malware can also propagate on other USB sticks.
- **Trojan (horse):** malware, which is distributed disguised as a desirable application, relying on users to innocently install them. In a sense, this is the simplest type of malware, as it depends on the user installing it. However, this simple strategy is surprisingly effective. In fact, many smartphone apps, as well as different programs and utilities for computers, are Trojans!

You can find more information about different kinds of malware online, e.g., in this link. In this lab, we will learn about malware by writing simple versions of a virus, a worm, and a UTM.

Question 1 (40 points): My First (?) Virus:

In this question, you will be writing a simple **virus**. A virus is a program that can "attach itself" to other programs, so that when the other program is run, the virus is also run.

Your simple virus will be a Python script that attaches itself to (other) **source-code Python scripts (.py files) in the same folder**. Typically, this is done by searching for other .py files on the same folder, opening each such file, and attaching the virus code to the script (if it does not already contain the virus code). As a note, practical viruses often attach to machine-code programs, and many attach to different macro/script files (e.g. of MS Office).

Note: your virus should not re-attach itself to a program which already contains the virus!

This question has three parts: A, B, and C. During each part we'll build a piece of our virus codebase until we're ready to put all the pieces together.

Part A:

We will first write a Python script named **Q1A.py**, that reads the files in the current (working) directory, and outputs a file containing the names of all .py files, each on a separate line.

Your program should work on both Linux and Windows; you may want to read about, and possibly use, the Python's os module. On your VM, place the completed Q1A.py program in ~/Lab2/Solutions directory.

Part B:

Next, write another script named **Q1B.py**, that receives as a parameter (on the command line) the name of a .py file in the current directory, e.g., *x.py*.

Q1B.py should check two things:

1) Is the given Python file a script (i.e., is it intended to be run directly)?
    a. For our purposes, a Python file is considered a script if it contains an `if __name__ == "__main__":` statement
2) Is it uninfected (i.e., does it not yet contain the virus code)?

If both of the above checks are true, then the Q1B script should re-write the input script (e.g., *x.py*), so that the new "x.py" will contain a Python script with the same functionality as of the original x.py, except that the new script will also perform the following simple spyware payload functionality:

Whenever the new "x.py" script is run, it appends, to the end of a file called Q1B.out, a line containing the entire command line used to invoke it, i.e., the file/script name ("x.py") followed by the arguments (parameters) with which the script ("x.py") was run, if any. If Q1B.out does not exist when the new "x.py" is run, then "x.py" should create Q1B.out.

Part C:

Finally, we are ready to write the full virus. You should create another Python script, **Q1C.py**, to house the full virus code. Your Q1C.py must *infect* every .py script in the current directory. By "infection", we mean alter the original .py script (e.g., x.py) so when the infected "x.py" would be run, it would retain its original functionality, but also have two additional functionalities:

1) The first additional functionality, the *payload*, is the spyware functionality of Q1B. Whenever the modified script "x.py" is run, it will append the entire command line used to invoke it to the end of a file called Q1C.out.
2) The second additional functionality is an *infection* functionality. Namely, the modified script will also have the same functionality as Q1C.py, modifying all .py scripts in the directory in which it runs, by adding the same spyware functionality and infection functionality. Q1C (and the modified scripts) should not modify scripts which have already been "infected" by our virus.

Save all three scripts (Q1A.py, Q1B.py and Q1C.py) in the Lab2/Solutions directory, and include them as part of your solution in your lab report that will be uploaded to HuskyCT.
Note: your programs should be documented, e.g., explain how you avoid re-infection of an already infected script.

Show your running code to the TA for review and approval!

Submit in submission site (*submit.edu,* IP: 172.16.48.8): you'll obtain an "approval code" from your TA upon showing them the completed and correct work for this question.

Include in your report uploaded to HuskyCT: all of your code, as text (not screen shot), and a screen recording showing its operation, including testing all relevant aspects. Also include the approval code you obtained from your TA.

Question 2 (30 points): My First (?) Worm

In this question, you will be writing a simple **worm**, **Q2worm.py**. A worm is malware that searches for machines that can be accessed over a network connection and has a specific vulnerability. Once such machine is found, the worm exploits the vulnerability to copy itself to the vulnerable machine, where it executes its payload – and deploys its infection functionality to infect other machines. (Of course, it may also propagate as a virus as well.)

Your simple worm will be a Python program, that uses the SSH and Telnet protocols to find vulnerable machines and infect them (some machines may support SSH, some Telnet, some both). Specifically, your worm will look for machines which have open SSH/Telnet ports and use a user/password pair from the list of "exposed" username-password pairs which you are given, in the file *Q2pwd* (in the *Lab2* directory). Search for machines in the subnet 172.16.48.0/24, i.e., IP addresses in the form 172.16.48.x where x is between 0 and 255.

Once your worm finds such a machine (and vulnerable account), you should copy the value in the file *Q2secret* from the home directory of the account, to your VM, in file *Q2secrets* in directory *Lab2/Solutions.* If you find several such machines, accounts, and *Q2secret* files, put all of the "secrets" on separate lines in your *Q2secrets* file. You should also copy *Q2worm.py* to the home directory of the vulnerable VM, and to the *Lab2/Solutions* directory of your VM.

Some Python modules that may be helpful:

- `socket` – for checking open ports
- `paramiko` – for establishing SSH connections
- `telnetlib` – for establishing Telnet sessions

Note: similar attacks (and worms) have been found in the wild; often, the worms identify vulnerable machines which were left with default passwords, e.g., home routers and IoT devices. Note that a "real worm" should follow the "break-in" step which your worm is doing, by running malware from the victim machine, often running the worm itself, to make the infection more effective and robust to discovery of the "first" machine where the worm began attacking. Worms need to prevent multiple infections of the same machine; lack of attention to this detail can cause excessive overhead – leading to load on the network and discovery of the worm, as happened in the (in)famous and fascinating case of the Morris worm.

Show your running code to the TA for review and approval!

Submit in submission site (*submit.edu,* IP: 172.16.48.8): you'll obtain an "approval code" from your TA upon showing them the completed and correct work for this question.

Include in your report uploaded to HuskyCT: all of your code, as text (not screen shot), and a screen recording showing its operation, including testing all relevant aspects. Also include the approval code.

Question 3 (10 points): A Step Toward My First (?) USB-Transmitted Malware (UTM)

In the following few questions, you will be writing a simple **UTM.** If you google it or learned a bit of Complexity Theory, you may think of writing a Universal Turing Machine, but no; we are talking about a very different type of UTM – a USB-Transmitted Malware.

To make it easier, we will use the **Rubber Ducky,** a USB device that emulates a keyboard. See below the background information on the Rubber Ducky, which you will need to write the scripts for all these questions.

Background: Rubber Ducky

A device we'll be using in the rest of this lab is the Hak5 USB Rubber Ducky. The USB Rubber Ducky is a kit that is used for system administration and penetration testing. It is a USB device that looks like a normal flash drive. The ducky registers itself as a USB keyboard once plugged into a computer. It then starts firing keystrokes to execute some commands on the targeted machine. See how to program the Rubber Duck here; there are also instructions elsewhere, e.g., the following may be helpful: https://blog.hartleybrody.com/Rubber Ducky-guide/.

The goal in this and the following question is to enact changes on your laptop (which will be the victim). Before starting, I encourage you to make sure your laptop is connected to the cse3140 network through the wired switch or the VPN.

We will be creating different *inject.bin* files which will serve as instructions to the Ducky dongle. The keystrokes are programed using a scripting language called **Ducky Script.** The script can be written in any text editor, such as notepad, textedit, Vim, Emacs, Nano or other. When the script is ready, a **Duck encoder** tool is used to convert these Ducky script lines into an *inject.bin* file that will be ready to be launched on the targeted machine. The *inject.bin* file is stored on the microSD and the USB striker dongle is used to execute the commands on the targeted machine.

To aid you in your development, and, to allow you to do some of the work remotely (without the physical Ducky), you should use a *Ducky Emulator.* You can use the emulator developed by UConn students (in Python, from here) or a previously-developed emulator (in C#, only for Windows, from here), as you prefer; see details on both (and some tips on Ducky scripts) in the Rubber Ducky Emulator Documentation file.

> Note: the USB Rubber Ducky does not receive interaction back from the computer. So, you need to carefully note which keystroke combination and delays will successfully accomplish the required task. This will require some trial and error on your part.

In the following questions, you will do the following process:

Step 0: Verify that the Chrome browser is installed on your laptop. If not, you may need to do this part of the project from a personal laptop/computer (with Chrome).

Step 1: Open Notepad (or any text editor you like) and enter your Ducky script commands. Save the plain text (and submit it as part of your report). Again, see how to program the Rubber Duck here.

Step 2: Compile the Ducky Script payload into the file *inject.bin*. There are two options for this:

1) Using the Javascript Ducky Encoder "***jsencoder.html***", which is a single HTML file which you can open and run in a browser locally, and which will encode Ducky script into inject.bin files. If using the (disconnected) lab laptop, you can find ***jsencoder.html in*** your *Lab2* directory, and can download it from there into the laptop.

2) Using Hak5's (online) Payload Studio: https://payloadstudio.hak5.org/community/ .

Step 3: Put the MicroSD card in the MicroSD-to-USB adapter, connect it to the laptop, and copy the inject.bin to the MicroSD.

Step 4: Move the MicroSD to the Rubber Ducky, connect it to the PC, and see if it does the required function. If not, fix your script and repeat till it runs correctly!

Step 5: Submit your work (scripts and screen shots).

Goal for Question 3:

Write a (simple) Rubber Ducky script that opens Notepad, writes a Windows script (batch) file that echoes your name(s), saves the file and runs it (to echo your names). If you are not familiar with Windows scripts, learn a bit about this simple technology first, for example, see the GeekforGeeks `Basics of Batch Scripting' article.

Show your running Rubber Ducky (with script) to the TA for review and approval!

Submit in submission site (*submit.edu,* IP: 172.16.48.8): you'll obtain an "approval code" from your TA upon showing them the completed and correct work for this question.

Include in your report uploaded to HuskyCT: all of your code, as text (not screen shot), and a screen recording showing its operation, including testing all relevant aspects, in the emulator. Also include the approval code.

Note: this UTM requires use of the physical Rubber Ducky, and therefore, cannot propagate further (as is) to other devices; of course, the malware we copy could be a virus/worm, but cannot propagate further using USB. However, it is actually possible to re-program many regular USB memory sticks to have the Ducky functionality, allowing the creation of an *infecting UTM*. But doing this is quite challenging and surely beyond this lab.

Question 4 (10 points):

This will be the same as question 3, but this time your Rubber Ducky script should write **and run** a Python "hello, world!" script.

Show your running Rubber Ducky to the TA for review and approval!

Submit in submission site (*submit.edu,* IP: 172.16.48.8): you'll obtain an "approval code" from your TA upon showing them the completed and correct work for this question.

Include in your report uploaded to HuskyCT: all of your code, as text (not screen shot), and a screen recording showing its operation, including testing all relevant aspects, in the emulator. Also include the approval code.

Question 5 (10 points):

This will be the same as question 4, but this time your Python script that is to be uploaded, saved, and run will be the simple Python **virus** of question 1 (Q1C.py).

Show your running Rubber Ducky to the TA for review and approval!

Submit in submission site (*submit.edu,* IP: 172.16.48.8): you'll obtain an "approval code" from your TA upon showing them the completed and correct work for this question.

Include in your report uploaded to HuskyCT: all of your code, as text (not screen shot), and a screen recording showing its operation, including testing all relevant aspects, in the emulator. Also include the approval code.

If you want to learn more about malware and malware analysis, here is one great resource: https://www.sans.org/blog/-must-have-free-resources-for-malware-analysis/