Assignment 1
CS-456
Fall 2015

Due Date: 10/24/2015 at 11:59 pm (No extensions)
This Project May be Done in Groups of 4
Only one group member should submit

Goals:

1. To gain hands-on experience with authoring malware.
2. To observe the stages of a computer worm lifecycle.
3. To appreciate the challenges of writing self-propagating malware.
4. To implement a self-propagating SSH worm.

WARNING!!! PLEASE READ BEFORE CONTINUING!!!

This assignment involves implementation of a malicious program which performs port scanning, an act considered highly offensive, and launches password cracking attacks against other systems, which is *illegal!* For this reason, this assignment *must be implemented and tested only using only the ThoTh Lab and nothing else! Failure to follow these instructions can get you into serious legal trouble. The instructor, the Computer Science department, the School, and the University take no responsibility for any personal damages resulting from the failure to comply.* By proceeding with this assignment you agree to follow all instructions.

Overview

In class we learned about computer worms; self-replicating malware programs which break into vulnerable systems and use the systems they compromise to launch attacks against other vulnerable systems. Different worms spread using different tactics. Some spread through email, some through the web, while others take advantage of file sharing networks as well as other means. In this assignment you are going to author a worm which uses the Secure Shell (SSH) service to spread, download malicious payload, and, finally, steal data from compromised systems.

You are going to implement three distinct types of worms described below. All of the worms that follow make use of SSH and Python skills you acquired during our in-class exercise.

1. **Worm 1: The Replicator:** A basic worm which scans its local network to detect systems running SSH service, attempts to break into one of those systems using a dictionary attack (i.e. password guessing attack), copies itself onto the compromised system, executes itself on the compromised system, and repeats the same process from the compromised system. This particular worm carries no malicious payload.

2. **Worm 2: The Extorter:** This worm is similar to the replicator worm, but in addition to spreading it also downloads an encryption program and uses it to encrypt user's files in the `Documents` directory and leaves an extortion message on the user's desktop.

3. **Worm 3: The Password File Thief:** This worm is similar to the replicator, except from every infected system it copies the `/etc/passwd` file to the attacker's server (that is, the VM from which the attack was originally initiated).

Sections that follow describe the specific requirements of each type of worm and give technical hints useful in completing this assignment.

Requirements

This section lays out the specific requirements for each of the three types of worms. The replicator, extorter, and password file thief worms shall be implemented in *separate* files named `replicator_worm.py`, `extorter_worm.py`, and `passwordthief_worm.py`, respectively. The detailed requirements for each type of worm are as follows:

- **The Replicator Worm:**

  1. This worm shall be executed using command line `python replicator_worm.py ARG1 ARG2 ...ARGN` from the origin system (the use of arguments is optional).
  2. When executed, the worm shall scan its local area network for presence of other systems running SSH service.
  3. The worm shall carry a dictionary of possible user names and passwords. This list shall include both correct and incorrect passwords for all VMs on the node.
  4. The worm shall attempt to login into discovered host systems using SSH user names and passwords in its dictionary until it successfully logs into **one of the hosts** or until it has tried all user names and passwords against all hosts without success. If the worm is unable to guess the credentials for any of the discovered systems, it terminates. Otherwise, the worm checks if the remote system has already been infected. If so, then it skips this system and moves on to attacking other systems. If not so, the worm copies itself onto the compromised system, executes itself on the newly compromised system, and terminates on the current system.
  5. Once executed on the remote system, the worm shall check if the system is is already infected and if so, terminates. Otherwise, the worm attempts to spread to other systems using the above-stated process. This check prevents two copies of the same worm from executing on the same system at the same time.
  6. When attempting to guess system credentials, the worm shall only use the user names and passwords in its dictionary.
  7. The worm ***shall not*** launch attacks from the same system more than once.

- **The Extorter Worm:**

  1. This worm shall be executed using command line `python extorter_worm.py ARG1 ARG2 ...ARGN` from the origin system (the use of arguments is optional).
  2. This worm shall encompass all features and conform to all requirements of the *replicator worm*, except the above requirement.

3. This worm shall download the encryption program from the http://ecs.fullerton.edu/~mgofman/openssl URL.
4. After downloading the `openssl` program the worm shall create a `tar` archive of the `/home/cpsc/Documents` directory and encrypt it using the `openssl` program. After the `Documents` directory has been encrypted, the worm shall delete the `/home/cpsc/Documents` directory and leave a note telling the user that his files have been encrypted and that he/she needs to purchase the decryption key from the attacker in order to get the files back.
5. All files shall be encrypted using password **cs456worm** (which openssl program accepts as one of the arguments; please see the next section for details).
6. This worm shall leave the files on the attacker's system unharmed.

- **The Password File Thief Worm:**

1. This worm shall be executed using command line `python passwordthief_worm.py ARG1 ARG2 ...ARGN` from the origin system (the use of arguments is optional).
2. This worm shall encompass all features and conform to all requirements of the *replicator worm*, except the requirement above.
3. When the is executed on a victim system, it shall copy the `/etc/passwd` file, the file containing information about system user names and passwords, back to the attacker's system (that is, the system from which the attack was originally initiated).
4. When the `/etc/passwd` file is copied to the attacker's system it shall be named as `passwd_<IP of the victim system>`. For example, `passwd_192.168.1.101`.
5. This worm ***shall not*** touch the password file on the attacker's system (that is, the system from which the attack was originally initiated).

Technical Guidelines

The list bellow gives hints and links to resources which will help you complete this assignment. All files containing sample code snippets can be found in file `samples.tar` accompanying this pdf file.

- The skeleton for the *replicator worm* has been provided in file `worm.py`. You are not required to use it.
- All SSH communications necessary for connecting to victim systems, copying files between systems, and executing the worm remotely can be implemented using the Python `paramiko` library and the associated techniques which we practiced during an in-class exercise. A copy of the code we used during exercise can be found on TITANIUM. If you need to brush up on your python skills, please check out the following good tutorial:http://www.tutorialspoint.com/python/python_quick_guide.htm. In addition, `samples.tar` contains the following sample files:
  - `getip.py`: contains code for finding the IP address of the local system.
  - `hostscan.py`: contains code for scanning for hosts on the local network.
  - `download.py`: contains code for downloading files from the web.

  - `runprog.py` contains code for running `openssl` program after it has been downloaded. The extorter worm will make use of this functionality. The code can be modified in order to run any external program.
  - `maketar.py`: contain code for creating a tar file of a directory.
  - `delfile.py` contains code for deleting a directory.

- ***This assignment MUST be completed using Python.***
- You may work in groups of 4.
- ***Your assignment must run on the ThoTh lab systems.*** Please consult the instructor if you do not have an account.
- Please hand in your source code electronically through **TITANIUM**.
- In addition to submitting your code through TITANIUM, **also leave a copy of your worm on an attacker VM on your node. On the desktop of the host system leave a README.txt file telling the name of the attacker VM and the path on the VM where the worm can be found.**
- You must make sure that your code runs correctly.
- Write a README file (text file, do not submit a .doc file) which contains
  - Names and email addresses of all team members.
  - Your node number, password (if you changed it), and the name of the VM containing a copy of the worm from which the initial attack can be launched.
  - How to execute your program.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Testing Guidelines

Your worm shall be tested for its ability to spread between systems. For example, assume your node contains 3 VMs, `A`, `B`, and `C`. You need to make sure that your worm is able to spread from `A` to `B` and then from `B` to `C` without human intervention asides from launching the worm on system `A`.

The nodes contain only two VMs. In class we will go through the steps necessary for creating the third VM and how to copy files to the nodes and VMs.

BONUS:

Implement the same worm using C++. Hint, you may want to use the `libssh` library to help facilitate communications. A tutorial for using `libssh` API can be found at http://api.libssh.org/master/libssh_tutorial.html.

Grading guideline:

- Program executes when run 10'
- Correct implementation of the *replicator worm* conforming to the requirements: 25'
- Correct implementation of the *extorter worm* conforming to the requirements: 30'
- Correct implementation of the *password file thief worm* conforming to the requirements: 30'
- README file: 5'
- Bonus: 15'
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.