Malicious Software Analysis

Prepared by:

Dhaipule Sujith Sai (2101068)
Darigi Srimedha (2101065)

Indian Institute of Information Technology Guwahati
Computer Security (CS361)
9 March 2024

1 Introduction

The aim of this assignment is to learn how a malicious software is built, how it changes the working of programs, and how they propagate. The limitations of the virus created are confined to the Linux environment.

2 Description

The program, named V (`v.cpp`), is written in C++ and simulates the behavior of a virus and a worm. Initially, it searches for files with the `.foo` extension in the `~/Documents` folder and appends its contents to all the `.foo` files found. Additionally, it scans for `.foo` files in any connected USB drives using mount points obtained from `/media/$USER/`. Furthermore, it's designed to auto-run itself whenever the infected USB drive is connected to a new device, effectively behaving like a worm.

3 Security Implications

The `v.cpp` program introduces significant security implications due to its actions:

1. **File Integrity and Confidentiality Concerns:** The program's action of modifying files in the `~/Documents` folder poses risks to data integrity and confidentiality, potentially leading to unintended information disclosure.
2. **Propagation and Spread:** By infecting `.foo` files on both local machines and connected USB drives, the program demonstrates characteristics similar to traditional viruses, potentially causing widespread infection and data loss.
3. **Unauthorized Replication and Distribution:** The auto-run feature enables the program to replicate itself on new devices without user intervention, increasing the risk of unauthorized distribution and proliferation.
4. **Potential for Exploitation:** Malicious actors could exploit `v.cpp` as a template for creating more sophisticated malware, leveraging its file system traversal and propagation capabilities.
5. **Disruption of System Functionality:** The infection and modification of files, especially in critical directories like `~/Documents`, can disrupt system functionality, leading to data corruption and application failures.

4 Pseudo Code

Algorithm 1 Function `readFileLines(filePath)`

```
1:  procedure readFileLines(filePath)
2:      Input: filePath (string)
3:      Output: lines (vector¡string¿)
4:      Create an empty vector lines
5:      Open the file specified by filePath
6:      if the file cannot be opened then
7:          Print an error message
8:          return the empty vector
9:      end if
10:     repeat
11:         Read a line from the file
12:         Add the line to the lines vector
13:     until end of file
14:     Close the file
15:     return the lines vector
16: end procedure
```

Algorithm 2 Function `main()`

procedure MAIN
    Execute system command "ls /media/$USER ¿ output.txt" to get a list of mounted drives
    Define an empty string command
    Call `readFileLines("output.txt")` and store the result in a vector *mounts*
    for all mount in *mounts* **do**
        Construct a command to list contents of the mount and execute it by appending the output to output.txt
        Print the constructed command
    end for
    for all mount in *mounts* **do**
        Prepend "/media/$USER/" to the mount to get the full path
    end for
    Append " /\/" (home directory) to *mounts*
    Execute system command "find / -type d -name
"Documents
" -print -quit ¿ output.txt" to find the "Documents" folder
    Call `readFileLines("output.txt")` and store the result in a vector *mountDocuments*
    Append *mounts* to *mountDocuments*
    Remove the last element (home directory) from *mountDocuments*
    for all mountDocument in *mountDocuments* **do**
        Construct a command to find files with extension ".foo" within the *mountDocument* directory
        Execute the constructed command to find ".foo" files
        for all ".foo" file found **do**
            Append the contents of "v.cpp" to it
        end for
        Print the name of the infected *mountDocument*
    end for
    for all mount in *mounts* **do**
        Construct a command to copy "v.cpp" to the mount and execute it
        Create a udev rule to execute "autorun.sh" on USB drive connect
        Create an "autorun.sh" script to compile and execute "v.cpp"
        Create an "autorun.desktop" file for autorun on USB drive connect
        Set permissions for "autorun.sh" and "autorun.desktop"
        if the current working directory is under "/media/" **then**
            Prompt the user for a directory name
            Create the directory under " /Documents/"
            Copy all files to the created directory
        end if
    end for
    Remove temporary files ("autorun.sh", "autorun.desktop", "v.cpp", "output.txt")
    Exit the program with status 0
end procedure

5 Detection and Prevention

Antivirus programs can detect and prevent the `v.cpp` program by utilizing the following methods:

1. **Signature-Based Detection:** By comparing file signatures against a database of known malware, antivirus software can identify and quarantine `v.cpp`.
2. **Heuristic Analysis:** Antivirus programs analyze the behavior and attributes of files to detect suspicious patterns indicative of malware, potentially flagging `v.cpp` for further investigation.
3. **Behavioral Monitoring:** Real-time monitoring of system behavior allows antivirus software to identify and block actions characteristic of malware, preventing `v.cpp` from executing its malicious behavior.

6 Conclusion

In conclusion, the `v.cpp` program serves as an educational tool to explore the workings of malicious software. Its behavior highlights the importance of cybersecurity awareness and proactive measures to mitigate risks associated with malware propagation and exploitation. Understanding such threats is crucial for safeguarding digital assets and privacy in both personal and organizational contexts.