Reverse Engineering: Its Critical Role in Computer-Based Forensic Investigations

CS4991 Capstone Report, 2024

Michael Kirk
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
mlk4zyg@virginia.edu

ABSTRACT
During my internship at a defense contracting firm, we recovered a file from a corrupted system that had no metadata signifying what it was or how it worked. I utilized a suite of forensic tools to reverse engineer the file. I analyzed the binary of the file in a virtual, secured environment and identified it as an executable file. I then used Ghidra to reverse engineer the program and determine how I could interact with it. After reverse engineering multiple subroutines without much success, I analyzed the structure of the program subdirectory. Ultimately, I determined that the file was a corrupt Docker container and I rebuilt it to be functional for further analysis. Reverse engineering was critical to the forensic investigation conducted. Future work should include constructing a program that automates the process of rebuilding a Docker container.

1. INTRODUCTION

It is May of 2017 and suddenly, across 150 countries, hundreds of thousands of computer systems are left nonfunctional. Some of these systems belong to major companies like FedEx and Nissan, while others are critical government infrastructure, like the UK's National Health Service systems. If it were not for reverse engineering, this disaster may have continued until it affected most computer systems in the world.

Reverse engineering (RE), an area of digital forensics, is the process by which a cybersecurity professional analyzes a system or software to document and understand its functionality and inner-workings without accessing its source-code. RE is an important field in cybersecurity because it helps identify security flaws in vulnerable systems, can determine a way to combat malware, and can aid in the forensic investigation of systems. RE is used to analyze potentially malicious software without any prior information on what it is or how it works.

The project I worked on this summer is known as the Antioch Project. It is a digital forensics challenge focused on RE that is meant to reflect real-world scenarios. I was given a potentially malicious file called "antioch" and with no other information, was told I needed to determine the purpose of the file and document any information I uncovered.

2. RELATED WORKS

There are large bodies of work currently published on RE and its role in combatting cyber threats in the past. These stories and publishing give good insight into the importance of RE and how it fits into cyber-security.

One of the most infamous examples was an attack called WannaCry. WannaCry was a

international cybersecurity crisis in May of 2017, in which 200,000 computer systems across 150 countries were hit by the WannaCry ransomware virus. Marcus Hutchins, a cybersecurity researcher, was able to RE the program and find a "kill switch" that halted the attack (Cloudflare, n.d.)

In 2020 the SolarWinds hack was uncovered and was found to be one of the most sophisticated and widespread cyberattacks ever conducted. Code was injected into software updates of a program used by large government agencies and companies. It was through reverse engineering this code that the hack and its true extent and purpose were understood (Temple-Raston, 2021).

3. METHODS AND PROCEDURES

A file was given as a download, titled "antioch." The goal was to discover a "flag" somewhere in the challenge, which in a real-world scenario would likely represent some critical information. There was no file extension and no other information given with the file. It was placed in a hex-editing program to view the raw data. The data was not obfuscated, so the file was quickly determined to be a "tarball," a standard type of archive utilized on Linux-based systems. The tarball was then extracted into its own directory.

The extracted contents of the tarball included 31 folders with alphanumeric names, seemingly hash values, and three miscellaneous files. Upon inspection, 30 of the folders had identical layouts, each containing two files with key-value pairs ("json" and "VERSION"). One of the miscellaneous files in the root directory contained a key-value pair titled "antioch," with the value of one of the folder names. This also happened to be one folder not identical to the other 30. It contained a file called "antioch," which after inspection in a hex-editor, was determined to be an executable file.

A secure virtual environment was created for the execution of the antioch file. Text is printed to the terminal referencing "AntiochOS" and version-data, seeming to resemble some form of operating system (OS) that is interacted with through terminal. "Type help for help" is also printed; however, typing "help" results in the string output "Available commands: help – print this help." and no further output. Typical Linux commands, such as "ls," "cd," etc., seem to have no effect on the program.

To determine the functions of AntiochOS, the executable file is put into Ghidra, a disassembler, which allows viewing of the program's assembly code. The program is also placed into a debugger, which enables "live" analysis of the program while it is running. After determining the lines that process the user-input using the live debugger, the disassembler was used to statically analyze the code of the program. The user-input was compared to a sequence of memory addresses, which were determined to be the string-representations of acceptable user-inputs to the program. The commands "consult" and "approach" were found.

Executing "consult" results in a string of 15 repetitions of the character "V" for multiple lines. "approach" prompted the user with a sequence of questions that referenced a scene from the movie "Monte Python and the Holy Grail." After inspecting the program, it was found that the answer to the first question always had a corresponding correct combination with the third question. For example, "{John, (anything), Blue}" and {Alice, (anything), Green}" may be answer combinations, but "{John, (anything), Green}" would not be. Using a scene from

the movie, the first answer combination was determined, and an integer was printed. Several other inputs were found for the first question after examining the folders that were extracted from the initial tarball. Each of the 30 other folders had a key-value in their json file with a name that could be used for the first input, each a name of a character from the movie. The problem that arose was trying to determine the corresponding answer to the third question. At this point, the purpose of the integer output was still unknown.

Multiple approaches were used to try to determine the corresponding answer pairs. It was found in the assembly that the integer would be printed from a mathematical function only after the first and third answers were hashed using an MD5 function, and compared to the folder name where the answer was found. Finally, an elegant solution was found when the program was patched to override the code that processed user-input. Instead of checking the condition of whether the hash was the expected value, with the developed patch the program would skip the comparison and continue to the code that printed the integer value. Using this approach, a series of integers 1-30 were determined for each of the 30 names input to the program.

After consideration, it was determined that the remainder of the functionality of the program would likely be discovered from the "consult" function and its true purpose, as well as what the 30 folders were meant to do. Upon extensive inspection of the folders and substantial research, it was finally determined that the tarball was a Docker container. Docker is a technology that allows for the containerization of applications: essentially packaging an application with all its dependencies to give it minimal footprint and make deployments of the application consistent and efficient. Each of the folders were "layers" to the Docker container, which stored information about updates to the base program. Layers are placed onto a container in a specific order, which was the purpose of the integer numbers.

4. RESULTS

A sample operational Docker container was analyzed to determine the proper structure of a container and how layers are applied. After modifying the manifest file in the root directory and inputting information on the layers and in what order to apply them, the container was successfully loaded into the Docker software and run. Upon execution of the "consult" function, ascii-art of the program's "flag" was now printed, signifying completion of the challenge. At this point, the container was fully rebuilt and functioned as intended.

5. CONCLUSION

The Antioch project was a challenging RE project that allowed me to explore numerous methodologies in digital forensics. I had previously never used a disassembler and had only used a debugger when writing my own code to fix issues. My knowledge with hex editors was more substantial, since I had completed an RE project on archive files the year before; however, I still found the Antioch project expanded my knowledge further.

The project was very challenging, and there were times where I spent entire 8-hour days staring at the same few lines of assembly without making progress. I feel it was through this process, though, that I learned one of the lessons my mentors emphasized the most in RE: patience, practice, and perseverance. There is always an answer to how the code works: computer programs are deterministic in that way. It is a matter of having enough patience and perseverance to attain the skills to find the answer.

6. FUTURE WORK

Docker as a service is growing in popularity. With this increase in popularity, debugging, manipulating, and reverse engineering Docker containers are skills that are increasing in usefulness. Highlighted by this project was the need for a tool that effectively recognizes and rebuilds Docker containers. While it was a tedious process to do manually, there is a high feasibility that, with the correct set of files, a program could be created that automatically analyzes and reconstructs containers that have been corrupted.

7. ACKNOWLEDGMENTS

I would like to thank my mentors in my internship, who have spent the last six years guiding me in my professional career. They have guided me personally and professionally and have worked hard to support me in all aspects of my life, including on the Antioch project. I would also like to thank Dr. Rosanne Vrugtman, my technical writing instructor, who was immensely helpful in the writing of this technical paper.

REFERENCES

Cloudflare. (n.d.). *What was the WannaCry ransomware attack? | cloudflare*. https://www.cloudflare.com/learning/security/ransomware/wannacry-ransomware/

Temple-Raston, D. (2021, April 16). *A "worst nightmare" cyberattack: The untold story of the solarwinds hack*. NPR. https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack