Justin Strelka

Analyze\_morris\_worm

7/21/20

**Analyzing Morris Worm**

**Table of Contents:**

Abstract…………………………………1

Motivation…………………………….1

Introduction………………………….2

Method/Measurement………….2

Results…………………………………..4

Conclusion…………………………….8

Bibliography………………………….9

Reflection……………………………..9

**Abstract:**

This paper begins with a brief explanation of why it is important to understand how various types of malware escalate in their deployment. Following is a short description of the malware to be analyzed and how to handle the malware files safely. We will discuss our findings on how the malware deploys itself and what parts of our system is targeted. Lastly, we will conclude with an overview of our findings and re-affirm what we have learned.

**Motivation:**

This paper is intended for students that are interested in understanding how to analyze malware and the tools available. Upon reading this paper you will be able to discuss how various pieces of malware escalate within a system. Understanding how malware escalates within a system is highly beneficial for hardening our systems and detecting the presence of malware when vulnerabilities have been exploited. You will be able to conduct malware analysis of your own and distribute your knowledge to an organization or the globe. You will understand how to handle a live piece of malware on your system safely without deploying the malware on your system.

**Introduction:**

Understanding how malware will infiltrate a live system and escalate its deployment within a system can better help us defend our production systems. Malware packages can be encountered in many places across the web and even the most secure systems can become compromised. Once a system has been compromised by malware the damages can be catastrophic to an organizations future. Early detection is key to minimizing the cost of clean-up and damages to a system. There is a variety of software available for analyzing known and new pieces of malware. We analyze malware to understand how it attacks a system for better formulated plans to restore or harden production systems.

**Method/Measurement:**

Before we begin analyzing live pieces of malware, we must first understand what type of malware we will be analyzing and how to handle it safely on our system. For the purposes of this paper we will be analyzing a piece of malware called the “Morris-Worm”. On November 2, 1988 a student at Cornell university by the name of Robert Morris Jr. had released a piece of malware to the global network which later came to be known as the “Morris-Worm”. The worm was an experimental program that was self-replicating and took advantage of a vulnerability within the UNIX sendmail service. [1] There was a bug in the worm that caused it to replicate and spread much faster than Morris had originally anticipated causing many more systems to become infected.

We will be using the Morris-Worm along with Kali Linux VM to conduct our analysis outlined in this paper. The Morris-Worm can be obtained at the following GitHub repository: <https://github.com/arialdomartini/morris-worm>. There are no live binaries in this repository, however we must still handle the source code with care. We do not want to run the source code in an IDE because the IDE will generate a live binary file and deploy it on our VM. If the code is deployed on our VM, it has the potential to infect and compromise the CIA of our system. Instead we can use the tools built into an IDE to view and analyze the code without running it, similar to debugging a program without deploying the debugger.

To protect our systems from being infected by the downloaded malware we will be using a VirtualBox VM to store and analyze the Morris-Worm. For the purposes of this paper we will be using our pre-established kali\_attack VM from our previous labs hosted by VirtualBox on a Windows 10 system. To analyze the Morris-Worm safely on our kali\_attack VM we will be using an IDE called CLion. If you are nervous about opening the source code in an IDE a simple text editor would be acceptable as well. However, you may not get all the same text coloring and other tools for analyzing the source code that a true IDE provides. CLion can be obtained at the following link: <https://www.jetbrains.com/clion/>. Provide a valid student email for a free student license upon install.

To begin we will first boot up our kali\_attack VM using the VirtualBox hypervisor. Once your kali\_attack VM has booted make sure the network adapter is set to NAT. We will need access to the global network for obtaining the Morris-Worm source code. Open a web browser and navigate to the GitHub repository in the provided link. Now open a terminal and create a directory you will use to store the malware in. I will be naming my directory “handle\_with\_care” as a pre-warning that the file contains sensitive code. We can now copy the link from the Morris-Worm GitHub repo and clone a copy of the repository into the handle\_with\_care directory as shown in **Figure 1.**

**A close up of a screen

Description automatically generated**

Figure : Cloning Morris-Worm repository onto local kali\_attack VM for analysis.

We can now open the Morris-Worm source code in an IDE or text editor of our choosing. For the purposes of this paper I will be viewing an analyzing the Morris-Worm source code using the CLion IDE provided by JetBrains which is free for students. Start CLion on the kali\_attack VM and select open or import project. When prompted navigate to the handle\_with\_care directory and select it to be opened. Once opened we will be able to view all the C source code within the Morris-Worm repository. **Again,** **use extreme caution when opening the source code in an IDE such as CLion as it is possible to build and deploy a live binary of the worm on your VM.**

**Results:**

We can see in **Figure 2,** the worm.c and worm.h files contain the main() function within them. This is where the Morris-Worm will start from as all C programs contain a main() function that is the beginning of a program. From the main() function we can see that the malware begins by changing the process name to “sh” which is the same alias that the legitimate program “Bourne shell” to disguise iteself. This change is done in the following line of code as shown in **Figure 2**: strcpy(argv[0], XS("sh"));. After disguising itself as a legitimate process the malware then takes several steps to eliminate the possibility of a core dump that could be traced if the process were to crash. Making the malware difficult to detect as it was disguised in the system as a legitimate process.

A screenshot of a computer screen

Description automatically generated

Figure : Morris-Worm disguising itself in the process table.

After the initialization sequence of the main() function the malware then enters a looping function called mainloop() which is where the program will continue to run. The mainloop() function stores the internal clock time to measure how long the process has been running on the infected host. After the mainloop() function calls a series of functions that attempt to find vulnerabilities to infect other machines. These functions are called the “H routines” and the flow of the program is designed in such a way that the H routines are guaranteed to call the r\_init() function once as shown in **Figures 3 & 4**. The r\_init() function creates a list of up to 500 gateways listed within the netstat tables of the host. Each of the H routines then exhaustively cycle the listed gateways searching for known vulnerabilities. The first H routine to be called hg() attempts to exploit a vulnerability in the Simple Mail Transfer Protocol (SMTP). If the exploit is successful it notifies the mainloop() routine.

A screenshot of a cell phone

Description automatically generated

Figure : Mainloop calling H routines.

A screenshot of a cell phone

Description automatically generatedIf there is a successful break in there is a report generated and sent to the owner of IP address 128.32.137.13. Once a machine has been infected the malware then initializes a password cracking phase. The malware uses the hosts resources to try cracking the passwords on the machine. The first of the three phases uses simple tactics of cracking passwords such as no password and username as password. If all attempts in phase 1 fail phase 2 is deployed which uses a word list to attempt to exhaustively crack passwords. If phase 2 fails the final phase 3 is initiated which uses a UNIX distributed word list which is used as a spelling dictionary as shown in **Figure 5.**

Figure : hg function to populate gateway list by calling r\_init() and attempt to exploit SMTP.

**A screenshot of a cell phone

Description automatically generated**

Figure : Cracksome subroutine for cracking passwords on infected host.

If a password is successfully cracked the account is then used to further escalate the malwares foothold. After the password cracking phase the process will then fork itself to a new process id as shown in **Figure 6**. Upon a successful fork the child process will begin its lifecycle in the mainloop() phase where it will repeat the process of calling the H routines and attempting to infect other machines. After a certain period of life the gateway list initially generated will be wiped and repopulated in an attempt to find new gateways to be infected. The parent process then kills itself off and the malware lives on in the child process. This makes the malware pid hard to track since it changes rapidly.

A screenshot of a cell phone

Description automatically generated

Figure : forking of child process and termination of parent for change in pid.

**Conclusion:**

We can see that the first objective of this piece of malware was to disguise itself on the hosts machine. Allowing the worm to continue running on the host undetected. The Morris-Worm did not try to extract any sensitive information from the infected hosts or corrupt data. The Morris-Worm simply found new hosts to infect by using the netstat tables and would send a response back to the IP address 128.32.137.13. The response would allow the user at the end point (Robert Morris Jr.) to create a map of the internet. However, the worm did cause some issues on the infected hosts. The overall speed of the host machines was greatly affected as the CPU time on them was simply overloaded by trying to continue the spread of the worm. The worm also tried to break into accounts on the hosts which can be viewed as a malicious act. It is not uncommon for modern malware to take a similar approach in trying to disguise itself and remove traces of logs. Understanding how malware will attack our production servers and where it will hide can help us harden our systems against such an attack and provide insight on how to remove a successful exploit of our system.

**References**

[1] <http://groups.csail.mit.edu/mac/classes/6.805/articles/morris-worm.html>

[2] <https://www.mit.edu/people/eichin/virus/program.html>

**Reflection:**

I found the simplicity of the Morris-Worm fascinating and that it was one of the first worms spread through the internet. I feel that Morris did not actually have malicious motives behind creating and releasing the worm. I think he was simply conducting research as an overzealous college student that took his research one step to far in the eyes of the public. The complexity of todays major attacks are a showing of how far we have come as an industry towards security in our systems. Yet it’s ironic that some simple vulnerabilities in our communication protocols can be exploited in a primitive manner such as the Morris-Worm. I feel this is a showing of how difficult of a position being a Cybersecurity specialist really is and how easy it is for a hacker with just enough knowledge to pull the correct card and make the house fall.