*The Morris Worm*

*Bryan Johnsen, Frank Coloccia, Purvika Gaikar, Karan Patel*

# **I.** **Introduction**

The Morris Worm is one of the most infamous and recognizable Malware in history. Developed by Robert Tappan Morris, it was a computer worm specifically designed to infect the ARPANET system - an early rendition of the world wide web that was made for improving communications between military personnel[1]. Morris would unleash his worm on November 2nd, 1989 to the dismay of the system administrators and even Morris himself, the Worm would nearly grind the entire ARPANET network to a mere halt - Emails would take days to send, removing the worms completely off of affected equipment took nearly two days, and caused around $53,000 dollars of damage [2]. Morris didn’t intend for serious damage to incur, rather he sought to point out critical weaknesses within itself. According to an official transcript of the case of US v. Morris - Court of Appeals, 2nd Circuit 1991, it was quoted “To demonstrate the inadequacies of current security measures on computer networks by exploiting the security defects”[3]. What Morris had inadvertently accomplished was not only the first recorded case of “Cyber Crimes” but highlighted the drastic impact hackers can have on computer systems, the often dire nature of malware, and the effects it has on computer networks and beyond. What this lab by Dr. Kevin Du seeks to accomplish is to demonstrate the nature of the Morris Worm attack, how worms in general typically work, and the significance of the most notorious lines of code ever written. Worms are dangerous when left unchecked - can grind any internet service, server, or connection to a timely halt; a total system crash if untreated.

# **II. Research Method**

## **Analysis of Worms**

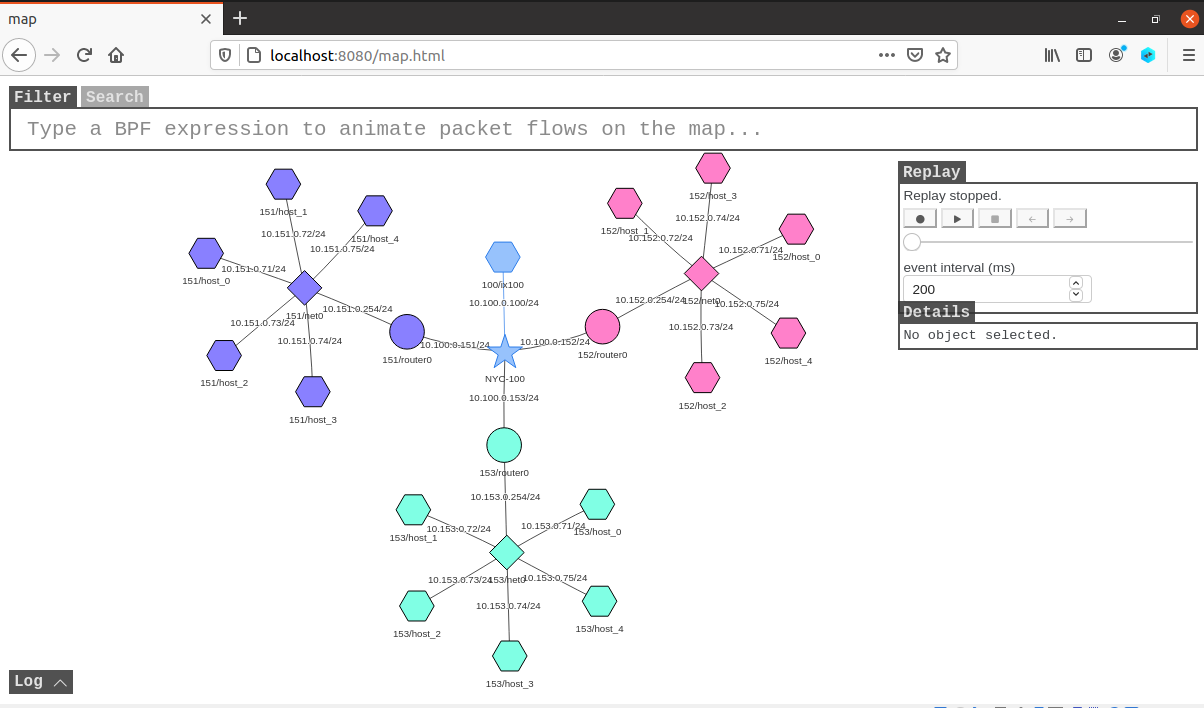
Research for this experiment was simple - most of us are somewhat familiar with the Morris Worm, but for those who are not, read up on Robert Morris and his history. He was a Graduate Student at Cornell University and is a gifted computer scientist. To understand this further - we must read into worms themselves and understand how they operate. Worms are a subset of Trojan horse malware - they self-replicate from one computer to another without having to rely on human activation (this being the key different from it to Viruses, who require physical initiation) [4]. Spreading throughout the LAN connection, they actively continue to propagate and consume computer resources in order to function properly. Worms are typically spread through various means like phishing, file sharing, instant messengers, external devices like infected USBs, and even social networks can spread this malware - MySpace had to deal with the Samy Worm written by Samy Kamkar. While the worm wasn’t as dangerous - only inserting a string “but most of all, samy is my hero” into peoples profiles, it spread to over a million profiles in 20 hours [5]. Once we understood the Malware and its history - particularly with Robert Morris, we could proceed to researching how to conduct the Seed Lab.

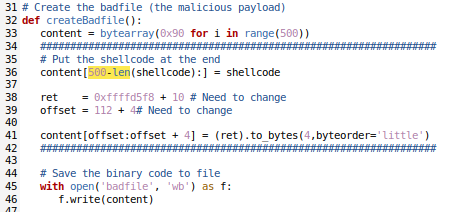
## **The Morris Lab Process & Procedure**

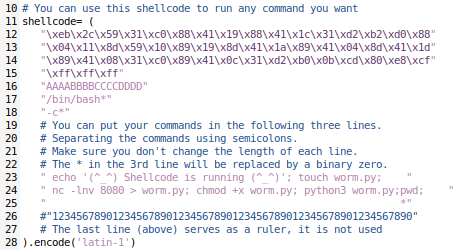
## Conducting the lab would be a serious matter. If there was a mis-step in the set up process or in certain task stages, we could end up completely crashing and destroying the virtual machine and even our home desktop if we were not careful. Carefully we had to scan what was being asked and kept note of 3 things,

1. What to program into the Morris Worm and what is important to the execution of the worm.
2. Crucial lab aspects that cannot be missed or skipped, whether for safety, function, or otherwise.
3. And how to properly understand and use the 3rd party programs / software necessary to safely demonstrate the worm's propagation and progress infecting each container.

For the lab demo itself, we decided to utilize, as you will soon understand, the “nano” internet set up, rather than the “mini” internet set up. Within this experiment lies two different internet models. The next image you’ll see as this report goes on, is the nano-internet. A relatively small collection of containers that we can use to demonstrate how the modified version of the Morris Worm operates. It’s small enough to get a grasp of the worm in action, and does not hinder our lab functionality. The “mini” internet set up is much larger than the nano set up and has many more containers and container-branches to span. This set-up requires powerful work terminals in order to operate. Upon testing on this Mini-Internet, it took our test machines well over 30 to 45 minutes to fully demonstrate the worm's effectiveness. When left unattended, the worm crashed the virtual machine from one of our lab members.

With this in mind, we can proceed with the lab itself. Using a virtual machine of Ubuntu 20.04, we can safely proceed with the set up and emulation. With the provided Internet emulators and the utilization of Docker, we can construct containers to act as infectable nodes for the Internet emulator to monitor and give a visual representation of the Morris Worms infection process. Booting up the lab, we create a lab folder and for the time being we’ll call this folder “wormlab” and this will hold the Labsetup folder and it’s contents. Navigate to the “Labsetup/internet-nano”folder and run dcbuild and dcup to activate nano internet. This will allow 15 containers to act as our nodes and eventual victims. Repeat this step in the Labsetup/map folder. This will boot up the map itself so we can visually see the worm in action. Do this in a new terminal window. Once everything is up and running, type <http://localhost:8080/map.html> and you will get the map of the containers to be infected.  


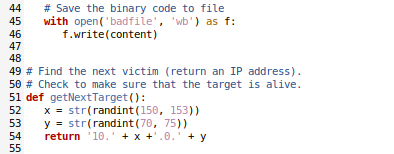
For this particular version of the Morris Worm, we will only target a buffer-overflow vulnerability, the actual Morris Worm targeted several holes in debug mode of the Unix sendmail program and many other vulnerabilities. To execute this we simply need to turn off address randomization - the kernel parameter is global. Once this parameter is off it will affect every container on the network.  
  
We do this by executing the following code:  
 sudo /sbin/sysctl -w kernel.randomize\_va\_space=0   
  
For the next segment, we are to edit the createBadFile() function within our attack code, worm.py. The badfile as the screenshot indicates, is the malicious payload that carries out the propagation. It is the very code that allows worms to be such a menace to the cyber world at large.  
  
While we have minor tasks to show the worm in action, it does not matter here - this section is summarized to a degree to demonstrate our progress towards re-creating a similar Worm that gained worldwide notoriety. What matters here is that our created badfile will allow us to infect and indeed in some cases, crash the containers and virtual machines.  


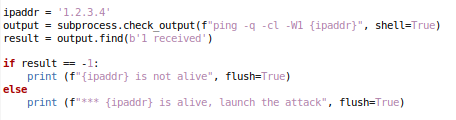
Next we are tasked with constructing the shellcode in worm.py, however the shellcode itself is simply located in the Labsetup/shellcode folder, this is for those who wish to tinker with the shellcode but for the time being, our shellcode looks something like this:

This lab has two approaches for it’s self duplication. To make it brief, one is to put all the code inside the malicious payload, and the other is to divide the attack code into two separate parts, a pilot code and a large sophisticated code. The latter can be further customized with additional programming languages such as C, Python, shell script, etc. and as you can see, our group has chosen the latter (see screenshot above, line 24).

Now we must reconfigure the propagation after the self-duplication task is complete. This is so that the worm can continue to slither its way into new machines. The function getNextTarget() is key here. It hard codes the IP address of its victim in order for the code to continue to function, duplicate, and propagate. This was the real vile part of the Worm itself - it needs to identify the core identity of the computer in order to function. Utilizing a random number generator will accomplish this goal with relative ease. Our groups generator can be seen in the screenshot below:  


What’s important here is that we are able to get getNextTarget() up and running. For brevity sake, our code for getNextTarget() looks like this in order to run our demo for the nano-internet:



But to ensure that if one was to ever initiate an actual worm attack, you need to actually make sure that the target machine is alive or not. The following code will accomplish exactly that:

# 

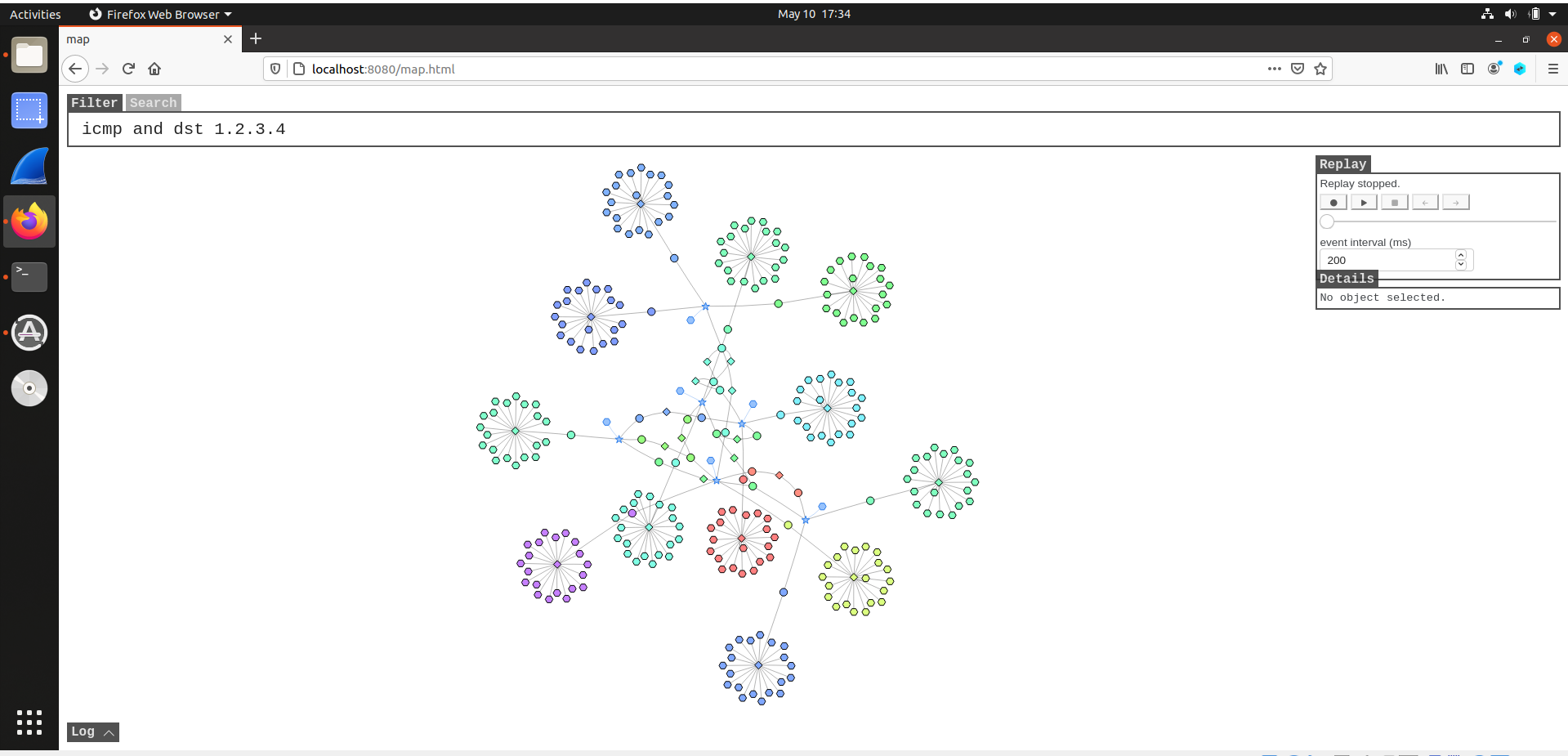
The code will send out a ping to (-c1) to target 1.2.3.4, and wait for (-W1) to reply. This will check whether the output contains the string “1 received”, to which a reply was made, indicating it’s alive. What is important to know in the case of the worm code getting out of control - is that it must be shut down if CPU usage gets near 100%, otherwise the VM will crash. So if you do not wish to potentially lose any work or do not want to risk causing any irreversible damage, utilize the htop command in order to observe resource usage and shut down the nano internet with dcdown or docker-compose down should resource use get out of hand.

Due to slight technological constraint and general improbability of being able to capture worthwhile footage of the mini-internet, we will not be showing a ~40 minute long video of the virtual machine crashing. Our group does not have ample access to video editing software to make the process any more easier thus unfortunately, no demo for it shall be available. However we are successful with running the mini-internet worm attack itself and will show what we have accomplished to make it so.  
  
First and foremost, we must start the mini-internet itself. Like nano-internet, we navigate to our Labsetup/internet-mini folder, and run dcbuild & ./z\_start.sh to build and start the containers. We must use the z\_start.sh command so that docker can initialize 10 containers at a time, the mini internet has over 200 containers. We had to be careful, because any foreign file entering the internet-mini folder would conflict with the z\_start.sh command.

If more files are to be added, then modifying the grep command in z\_start.sh can be used to exclude the additional files. Below is the z\_start.sh command:



To view the logs of each container, utilizing the code docker logs -f <id of container> and using ctrl-c to exit without stopping the container, should be kept in consideration. It is useful and even fascinating to see the events going on with each container during the attack. To get specific activity on a specific container, utilizing the code dockps | grep as153 will display every container in the mini-internet system, dubbed the AS-153 Autonomous system. The mini-internet map looks something like this:



Launching the attack was no laughing matter, this was a massive network of containers. The lab made very clear instructions on a few key details in order to ensure safety.

1. The IP addresses follow a pattern. 10.X.0.Y. X ranges from 151 to 180 and Y ranges from 70 to 100.
2. Do not attack multiple nodes, one node will do. Attacking multiple defeats the purpose of the malware being a worm. It will spread across the “internet” automatically.
3. The parameters icmp and dst 1.2.3.4 in the filter box can assist with visualization of the infected machines.

Rest assured, we have completed the mini internet portion of this lab, however showing it would take too long. For reference, the demo provided by Dr. Kevin Du demonstrates the mini internet infection in 43 seconds (sped up by 5x). With that - we were able to accomplish the goal of bringing down not only the nano & mini internet, but even someone's virtual machine in one instance! With this - we can detail incredible findings.

# **III.** **Results**.

The results of the lab itself was understanding and respecting the code that may be. The Morris Worm was a dangerous piece of software and has significance in Computer Science & Cyber Security history. It was fascinating to see a part of history demonstrated before our eyes. It is not something to take lightly in this regard. At the end of this experiment, we found many interesting details; both in our initial background research as well as the execution of the worm itself.

For starters, the worm executed in a very timely manner. It was quick to initiate and even quicker to spread to other nodes. Obviously this process could be faster but for what we have in terms of testing machines, It was also fairly quick to spread. Given time, any worm could lead to unprecedented levels of damage. Even in our experimental phase, we had a complete shutdown of a virtual machine due to overtaxed resources. The demo video demonstrates the worm in perfect execution. Should the video carry on for too long, it may prove fatal to the stability of the virtual machine and crash it. In total, we had 4 crash incidents occur. Three of them were due to Worm related mishaps with the mini-internet. One instance was the process of trying to boot up the mini internet map itself!

Overall the results were a resounding success. We managed to properly coordinate, build, and inject a worm into a network. Furthermore we gained insightful information on how these sorts of malware are developed and what they look like from a development view point rather than one of security. We worked diligently to get the Worm to function as it should - there were halts and stops in our progress but we managed to get by them and surpass them. What we didn’t account for was the delicate nature of the lab. There have been many discussions on the precarious nature of how the lab is set up and how to properly construct the missing pieces of the worm.py code. But we got resounding success with our endeavors.

# **IV.** **Conclusions**

After we shut down the docker containers and properly cleaned up. We understood the value of dangerous experimentation in a virtual environment. Worms are malware; they are capable of causing significant damage to computers, LANs, and servers responsible for many different things from business practices to data storage. No matter the circumstances, experience will always have a specific edge over any textbook or lecture. To experience exactly what happens during a Worm being unleashed, or to see a virus infect and destroy a computer, or even Ransomware in a controlled environment occur; it’s absolutely riveting to see, understand, and dissect what exactly executes during these sorts of happenings. It is only through experience will we gain a complete understanding of how these sorts of events occur and are dealt with.

These are the sorts of labs we believe will make a most fulfilling learning experience. When you are able to see exactly what you have been taught for so many years, to see it in action is an experience you normally would only get in career positions. But to be able to see it happen, seeing each individual node ping another infected node as the worm spreads throughout the nano-internet network of containers is incredible and invaluable to the student.

This lab experiment was a fulfilling experience for each member involved and was invaluable to learning not only how worms worked, but to the degree that garners a specific kind of understanding. If you were to take a machine apart, then put it back together exactly how it was before, you’d gain a certain technical comprehension of the inner workings of the machine whereas someone who just uses it might not. The same concept can be said of experienced malware programmers and script kiddies. To us individuals in the field of Cyber Security, this sort of experience only bolsters and reinforces present knowledge to newer heights. This is the kind of practice that Cyber Security students must have; create a virus, let it run its course through a controlled environment, understand how it works, how it executes, what it does to the infected system, and then develop a strategy on how to respond or prevent further damage from occurring. If this conclusion was not reaffirming critical concepts already, the concept of “practice makes perfect” should always be considered in Cyber Security.

#### 

##### 

##### **References**

[1] - Hauben, M., & Hauben, R. (1998). Behind the Net: The Untold Story of the ARPANET and Computer Science (Chapter 7). *First Monday*, *3*(8). <https://doi.org/10.5210/fm.v3i8.612>  
  
[2] [3] - 928 F.2d 504 (1991) UNITED STATES of America, Appellee. v/ Robert Tappan MORRIS, Defendant-Appellant, No. 774, docket 90-1336. United States Court of Appeals, Second Circuit transcript https://scholar.google.com/scholar\_case?case=551386241451639668   
  
[4] Nica Latto, Worm vs. Virus: What’s the Difference? AVG Antivirus, December 18th, 2020, updated May 4th 2021, https://www.avg.com/en/signal/computer-worm-vs-virus

[5] - Jeremiah Grossman, Cross-Site Scripting Worms and Viruses: The Impending Threat and the Best Defense, Whitehat Security, April 2006, <https://www.whitehatsec.com>, archive link <https://web.archive.org/web/20110104191201/http://net-security.org/dl/articles/WHXSSThreats.pdf>  
  
Lab Documentation - Dr. Wenliang Du, Seed Labs Morris Worm Attack Lab, 2021 <https://seedsecuritylabs.org/Labs_20.04/Networking/Morris_Worm/> , <https://seedsecuritylabs.org/Labs_20.04/Files/Morris_Worm/Morris_Worm.pdf>