Decrypting The Morris Code

D. W. Jia*
Stanford University
450 Serra Mall, Stanford, CA 94305, USA

I. GENISIS

In 1988, Robert Tappan Morris, a graduate student in computer science at Cornell University, constructed the Morris Worm. The Morris Worm was the first major attack on the budding inter-web of connected computers at universities, research institutions, and government agencies called the Internet (Fig. 2).

A significant flaw in what was supposed to be merely a research project to reveal the security loopholes of UNIX systems inadvertently turned Morris's program into an attack that caused significant damages to computers and information infrastructures across the nation, affecting over six thousand major UNIX machines, with some estimates of total cost as high as $10,000,000 (US v. Morris).

Yet, because of its overarching impact on the greater computing community, the Morris Worm was instrumental in shedding light on the importance of computer and network security.

FIG. 1: An Image of Robbert T. Morris, creator of the Morris Worm and current professor of computer science at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. Image recreated by the author. Original image from the MIT CSAIL website.

In this paper, we will first investigate the technical flaws in the computer systems that the Morris Worm was able to exploit, then examine the inherent bugs in the design of the Worm that produced its devastating effects, and finally discuss the cost and damages it incurred. We will then assess this event as a pivotal factor in legitimizing the field of network security.

II. AN EXPERIMENT GONE WRONG

Just before midnight on November 2, 1988, the twenty-three year old Morris planted a computer program he had been painstakingly constructing for weeks on a computer connected to the Internet at the Massachusetts Institute of Technology (MIT). This program, a computer worm, was intended to replicate itself and spread silently from computer to computer. All it needed was one seed.

For Morris, planting the seed at MIT was an excellent opportunity to disguise its origin. From here, as we will see, his Worm quickly spread autonomously throughout the network.

The Worm exploited three main security loopholes in the UNIX system to infest neighboring computers on the network (Fig. 3). Its most preferred method was to crack users passwords and attempt to login directly to remote computers. If this fails, the Worm would take advantage of small bugs in the "fingerd" and "sendmail" commands of the UNIX system to gain access to its targets. To better understand these design flaws, we will discuss each of the exploits in detail.

III. DECODING THE WORM

First, the Worm looked to find username and encrypted password pairs. On the UNIX machines that the Worm infected, files containing usernames and encrypted passwords of all users on the network were publicly viewable.

According to Schmidt and Darby, "[t]his mean[t] that the Worm could compare various encryptions of possible passwords against the encrypted passwords in this file without triggering security warnings." Many of these files were also stored in the same location on different systems, making it even easier for the Worm to detect them (Schmidt and Darby).

*djia@stanford.edu

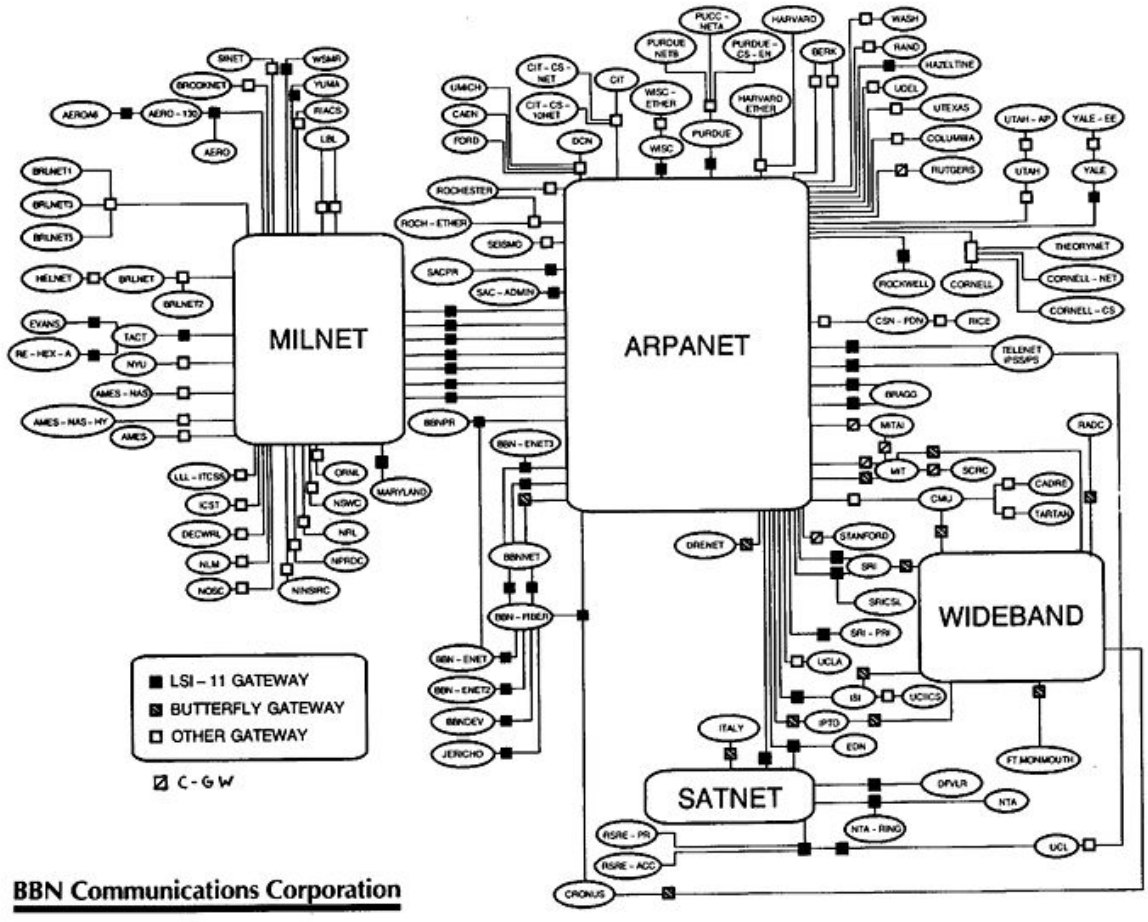

FIG. 2: A graph illustrating the connected nodes of the Internet circa 1986. Each node represents an end connection point. MILNET, ARPANET, WIDEBAND, and SATINET are centralized connection locations. Cite: BBN Communication Corporation.

A. Remote Login

The Worm was programmed with code to try to recover these passwords through brute-forced cross-checking against words in a dictionaries. Using this method, the Worm was sometimes able to recover encrypted passwords for the target addresses. It would then use the recovered username password combination to remote login to the target machine and download a copy of itself onto the new host computer and subsequently infect it.

In addition, during the process of looking through public files, the Worm would also be able to gather a list of addresses of neighboring machines on the network that could be potential targets. The following two infection methods utilize this collected list of addresses to spread the worm without depending on the use of login passwords.

B. "fingerd"

If remote login failed or no passwords were recovered, the Worm then resorts to exploit a crucial flaw in the "fingerd" command. This innocent-sounding command was used to retrieve information from a remote machine. Normally, it took an input of no more than 512-characters, but because of a flaw in its design, instead of ignoring any input that exceeds the 512-character limit, the excess characters would automatically be copied on the memory stack of the remote machine.

This means that a malicious user can execute arbitrary code on the remote machine by carefully constructing an input larger than the 512-character limit. The Worm cleverly took advantage of this exploit by calling the "fingerd" command with its carefully constructed input where the extra characters installed a small program onto the remote computer that downloaded a copy of the

Worm to infest the new machine.

C. "sendmail"

The third and final exploit relied on a major flaw in the "sendmail" command. As its name suggests, the command was standard on many UNIX systems for sending emails. The designers of "sendmail" added a settable "DEBUG" flag that "allowed someone to send mail to a process, rather than a user account" (Schmidt and Darby). This was originally meant to streamline the program testing process. Although this was indeed a common method software engineers used to assist the debugging and testing process, it was never taken out of the production version of the program. As a result, all copies of the program that were distributed included this extraneous feature (Schmidt and Darby).

Because of this, the Morris Worm was able to send a carefully constructed message using the "sendmail" command with the "DEBUG" flag turned on. When this altered message was interpreted at the remote machine, because the "DEBUG" flag was enabled, the receiver would pass the message to a process instead of a user account. This resulted in code being executed at the remote machine to download a copy of the worm onto it, thereby infecting the machine.

Yet, in some cases, all of these methodsremote login, "fingerd," and "sendmail" would prove to be unsuccessful. In this case, the Worm would mark the target as immune and move on to the next potential target on its address list.

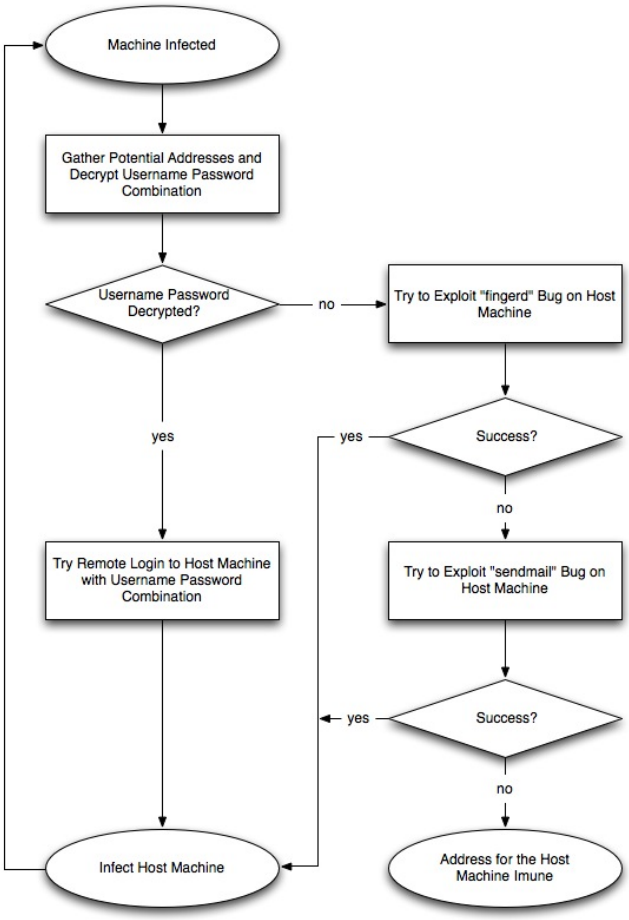

FIG. 3: A flow diagram showing the high level logic that the Morris Worm used to infect host machines and spread itself onto other machines. This diagram is created by the author.

IV. WHO'S FAULT WAS IT ANYWAYS?

After seeing the Worm turn these seemingly trivial system bugs into major security loopholes, it may be convenient to label software engineering errors as the main culprits of the network failure. However, just as much responsibility for this massive network failure could be attributed to the unprecedented nature of the attack.

Unlike the Internet ecosystem today, where software engineers are largely aware of the network security threats, "[t]here had never been a simultaneous large-scale security event prior to [the Morris Worm]...[i]t was the first significant denial-of-service issue that came to peoples attention" (Marsan). Before 1988, computer network security was simply "not a major concern of Internet community, at least, not to the degree it was after November 2."

Software developers who were more used to developing for disconnected individual machines were not as aware of the security implication of interconnected computers. They were far more focused on other features such as ease of use and functionality (Schmidt and Darby). The small bugs in the UNIX operating systems that the Worm had cleverly exploited were disregarded and brushed off as being less essential to the overall system.

Nevertheless, many of these security loopholes were indeed simple programming errors that were overlooked in the testing process or forgotten altogether. Inserting a line of code to check for the input character size could have easily averted the attack that utilized the buffer overflow bug in "fingerd". It also would not be unreasonable to blame the error in "sendmail" on inadequate testing or simply a memory lapse.

The Worm demonstrated that these simple flaws, though seemingly small, could still lead to significant security risks. The Worm also forced "close re[-]inspection of operating systems" code, which led systems designers to discover "a number of other bugs that the worm did not exploit" (Schmidt and Darby). In later systems, many of these bugs, including the ones in "fingerd" and "sendmail" have been patched, and password files made no only publicly viewable by all users on a network (Schmidt and Darby).

V. WHAT GOES AROUND COMES AROUND

Having discussed the inherent flaws of the system and oversights that have led to major security loopholes, one might ask: to what responsibilities should the original designer of the Worm, Robert Morris, have to answer? It was, after all, his Worm that caused this upheaval in the computing community. But in fact, Morris never intended for the Worm to do any harm. He was purely intellectually motivated. Morris original vision was to merely spread "a tiny program...and have it secretly take up residence in the memory of each computer it entered...always hiding in the background to escape detection."

Its creator never meant for the Worm to slow down computers (Markoff), and in fact created, albeit unsuccessfully, specific mechanisms to deter any compromises of normal performance on infected machines. By spreading it silently, he simply wanted to demonstrate the network vulnerabilities that he had discovered. Yet, it turned out that flaws in Morris own code had caused the unintended catatonic effect that the Worm had on the systems that it plagued.

There were several key flaws in the design of the Worm. Morris knew that many instances of the Worm running on the same machine would clog up the computers memory and eventually cause it to crash. With this in mind, he designed the Worm so that each time it infected a machine, it would check for the presence of other worms by sending out a signal to which other Worms could listen. When a preexisting Worm is found, a global variable called "pleasequit" is set so that the new Worm would know to quit its process. However, throughout the entire lifespan of a Worm, only one such signal is ever sent out. This means that if multiple instances of the Worm infected the same clean machine at once, their signals looking for existing Worms would all be sent out at once. But because these Worms infected the machine simultaneously, there would be no preexisting Worms on the system to listen to their signals. Thus, no Worm would receive the signal. Each Worm then, believing that it was the only one running on the current machine, would proceed to infect it. Because a Worm only sends out this seeking signal once in their lifespan, all of the Worms that have infected this machine, unaware of each other, would become invulnerable. Also, in many cases a machine would be heavily loaded with Worms, causing a significant lag in the signal transmission process. The result is that even when a signal was received, the abnormally high latency would cause the Worm to label it as a false signal.

Finally, in an over-ambitious effort to eliminate the possibility of programmers manually setting the "pleasequit" global variable on the network to prevent the Worms infection, Morris designed the Worm so that a certain percentage of the infecting swarm would never listen for the existence of other Worms and would continue to infect regardless of the state of "pleasequit". In perhaps the most significant blunder of the Worms design, Morris set this ratio of immortal Worms to be one out of seven when in fact, given the conditions, "this choice was off by a factor of about 10,000" (Schmidt and Darby).

In addition to the already inherent design flaws of the Worms exit conditions, this extremely overcompensated ratio allowed multiple copies of the Worm to readily take over a machine, drain its system resources, and eventually cause it to crash. In this way, the young Morris, in an innocent effort to showcase his research and intellectual interests, caused over 6,000 computers to crash in a matter of hours.

VI. NOT THE HACKER WE DESERVE, BUT THE HACKER WE NEED

Although causing little long-term damages, the Morris Worm infected roughly ten percent of the estimated 60,000 computers connected to the Internet almost overnight, affecting Sun 3 systems and Digital VAX computers running BSD UNIX (US v. Morris). The original intent of the Worm was to prove the security flaws in the network. It was meant to do nothing besides infect computers connected the Internet. But Morris major design flaws as we have discussed had caused it to self-replicate in already infected machines and significantly slow down, and in many cases, crashed the computers.

The creator of the Worm, who is now a professor of computer science at MIT (Fig. 1), was later convicted of violating the Computer Fraud and Abuse Act. At the time, Morris became the first person to be convicted under the act. He was later sentenced to three years probation, four hundred hours of community service, and fined $10,000. According to US v. Morris, while the Worm caused no physical damages to the machines, the United States General Accounting Office estimated that $100,000 to $10,000,000 in damages were lost due to lost Internet access and reconfiguration costs of machines that were infected (U.S. v. Morris). This course decision set the precedent that tampering with public computer networks, even for research, should be done with caution.

In spite of the uproar the Worm caused in the national computing community, it did little to compromise information security. At the time that the Worm was released, the Internet was still in its infancy, so the approximately 6,000 affected computers were almost entirely composed of those from research institutions and government agencies. Although these computers held important documents, the worm did not alter or destroy any files, save or transmit any passwords that it cracked, nor did it cause any physical damage (Schmidt and Darby).

The most significant breach of information security would perhaps be limiting access and availability of information. At sites where computers were being infected, system administrators pulled computers off of the network in an effort to stop the Worm from spreading. Some

were forced to shut down and restart the machines to manually kill off the Worm. Ironically, many of these actions proved to be counterproductive because disconnecting from the network prevented the reception of several key messages that were sent out on the network detailing methods on the Worms removal. At least one of these messages was sent out by Robert Morris and one of his colleagues when they discovered the unintended adverse effects of the Worm.

In retrospect, perhaps the most prominent consequence of "The Great Worm" of 1988 was the profound effect it had on legitimizing the field of network security and its central role in "set[ing] the stage for network security to become a valid area of research and development" (Marsan). Prior to the Morris Worm, the Internet "was considered a friendly place, a clubhouse" (Marsan) where information could be passed and everyone was trusted. "[C]omputer security was not a major concern of [the] Internet community" (Schmidt and Darby). The Worm showed the greater computing community that some members could have malicious intent in mind (Marsan), and that engineers must defend against these intents.

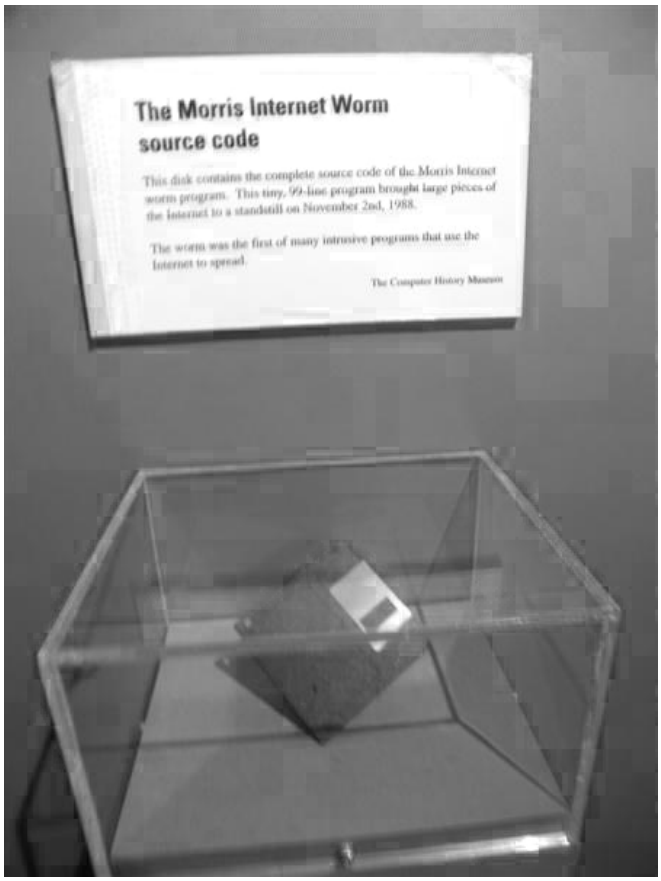

FIG. 4: A floppy disk containing the original 99 lines of source code for the Morris Worm, found in the Boston Museum of Science. Credit: Wikipedia.

Indeed, the Worm "foreshadowed how future distributed denial-of-service attacks would be used to overload systems and knock them off the Internet" (Marsan). Not only did the resulting court case set a precedent for future cases of unauthorized network access, the Morris Worm made it clear that security was as important as usability, efficiency, functionality and any other major tenants of software development.

Looking back with this in mind, one might not be so inclined to hold Mr. Morris accountable for wrongdoing. Yet, regardless of the moral and ethical implications of the infamous Worm, it proved to be the crucial piece that catalyzed the transformation towards a more security-aware computing ecosystem. As the New York Times puts it shortly after the Worm became a national sensation, "it was a programming triumph fit for publication in a journal...it caused no lasting damage...it pointed up far more serious security threats" (Wines).

[1] U.S. v. Morris. United States Court of Appeals, Second Circuit. 7 Mar. 1991. Online.

[2] Markoff, John. "How a Need for Challenge Seduced Computer Expert." The New York Times. The New York Times, 06 Nov. 1988. Web. 18 Apr. 2012. ¡http://www.nytimes.com/1988/11/06/us/how-a-need-for-challenge-seduced-computer-expert.html¿.

[3] Wines, Michael. "A Youth's Passion for Computers, Gone Sour." The New York Times. The New York Times, 11 Nov. 1988. Web. 18 Apr. 2012. ¡http://www.nytimes.com/1988/11/11/us/a-youth-s-passion-for-computers-gone-sour.html¿.

[4] Kotadia, Munir. "16 Candles for First Internet Worm - CNET News." CNET News. CBS Interactive, 3 Nov. 2004. Web. 18 Apr. 2012. ¡http://news.cnet.com/16-candles-for-first-Internet-worm/2100-7349_3-5438291.html¿.

[5] Schmidt, Charles, and Tom Darby. "The What, Why, and How of the 1988 Internet Worm." The What, Why, and How of the 1988 Internet Worm. 1 Aug. 1998. Web. 18 Apr. 2012. ¡http://ethics.csc.ncsu.edu/abuse/wvt/worm/darby/worm.html¿.

[6] Cyberpunk by Katie Hafner and John Markoff, Touchstone Books, New York. 1991

[7] Marsan, Carolyn. "Morris Worm Turns 20: Look What It's Done." Network World. 03 Oct. 2008. Web. 18 Apr. 2012. ¡http://www.networkworld.com/news/2008/103008-morris-worm.html¿.

[8] Page, Bob. "A Report on the Internet Worm." Electrical and Computer Engineering Dept. Ryerson University. 07 Nov. 1988. Web. 18 Apr. 2012. ¡http://www.ee.ryerson.ca/ elf/hack/iworm.html¿.