Prospectus

Language Agnostic Deep Learning-Based System for Vulnerability Detection
(Technical Topic)

Actor Network Theory and the Growing Disparity Between Modern Security Practices and User Integration
(STS topic)

By

Ethan Gumabay

17 October 2019

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Signed: _____

Approved: _____ Date: _____

Benjamin Laugelli, Department of Engineering and Society

Approved: _____ Date: _____

Yuan Tian, Department of Computer Science

Introduction

In 1988, an exploit later dubbed the Morris Worm shocked the internet, rendering it largely unusable for the better part of a week. The Morris worm was an exploit that propagated through the internet infecting 10% of all connected computers by taking advantage of a software vulnerability commonly known as a buffer overflow (Harald, 2018). The vulnerability itself is relatively easy to patch but extremely challenging to detect (Tevis, 2004). This project proposes an adaptation of a particular model currently used for vulnerability detection that would allow for broader analysis. The current model is extremely effective at detecting vulnerabilities, but only if those vulnerabilities are contained in a C or C++ program. The aim of this project is to produce a model that will detect viruses in a broader range of programming languages in order to widen the protection scheme.

Simply building a model with a broader detection range, however, is not sufficient to entirely solve the problem at hand. A fine-tuned model also addresses the non-human actors in the network. The oblivious human actors who were used as surrogates for transporting the worm are an entirely separate entity from the software itself. For this reason, it is also important to consider the implications of building such a system for the consumers. A better model also fails to address the unbridled decision making of the worm itself, which is ultimately the actor that allowed the Morris Worm to have the widespread effect it did (FBI, 2018). By addressing all actors in the network, future exploits can be effectively prevented. The key to understanding how to protect against an exploit is to understand which vulnerabilities it takes advantage of; the network of actors that allow an exploit to spread is no different in that its weaknesses in their entirety must be identified and addressed. Failing to address the other sociotechnical factors involved would render the model useless, defeating the purpose of its development.

The problem of protecting modern systems from rapidly developing malware has long plagued developers. The problem has persisted because the solution is not merely technical; many of the protections fall into the hands of the users. For this reason, the solution to the problem calls for addressing all actors in the network, both human and non-human (Callon, 1987). On the technical side, the development of a more rigid model that is able to detect vulnerabilities on a broader scale would be extremely useful to developers. This is because it would allow them to automate vulnerability detection during development and be proactive to detect potential flaws in the system instead of the current reactive approach they are forced to take. On the non-technical side, it is important to consider every actor in the network when addressing exploit prevention. Many security features are available to consumers but are never used. Evaluating the network holistically allows us to determine exactly where the human actors are susceptible, thereby allowing us to prevent exploits from a non-technical standpoint. Ultimately, a purely technical or social fix would be largely useless without the presence of the other.

Technical Problem

Every software ever produced has had some sort of vulnerability, many of which have led to damaging exploits. Developers of Firefox, a popular web browser, have claimed they find about 100 new vulnerabilities every year in their own software (Holt, 2019). Known software vulnerabilities have fairly good protections and are listed in the National Vulnerability Database (NVD). The NVD, is clearly limited in that it only contains protection information about known vulnerabilities; this means that any new vulnerability needs to be logged so the NVD can be updated. In 2016, Li and Zou published a paper outlining a method of detecting potential

vulnerabilities via code similarity analysis. The system was about 94% accurate for known vulnerabilities (Li et. al 2016). Two years later, Li and Zou published a paper also aiming to detect software vulnerabilities, this time via a deep learning-based system that would detect never-before-seen vulnerabilities. The system was shown several examples of software containing vulnerabilities. Once this model has been trained, it can be shown a never-before-seen piece of software and deduce with about 85% accuracy whether it contains a vulnerability while only incurring about a 4% false positive rate (Li et. al, 2019). In theory, this model could be packaged and used to do an automated scan for particular software vulnerabilities when downloading files (the primary source of exploits) (Computer Hope, 2019). This particular model could potentially reduce the amount of vulnerabilities on a typical users' computer by a significant margin.

The primary limitation of the VulDeePecker model is that it is not language agnostic. Notably, it only functions if the given code is written in C or C++. The problem, then, is that not all downloadable software is written in C or C++, thereby rendering the model useless in many cases. One solution would be to translate every downloaded file into C or C++ and then check, but there are complications with that, namely that translating code from one language to another is extremely difficult and sometimes even impossible. The technology I plan to develop, then, is an adaptation to the model that will allow it to be language agnostic.

A language agnostic model seems daunting on its surface due to the extremely large number of coding languages, however the foundation for the model relies on the compilation process of most coding languages. Because computers only have the ability to understand machine code (1's and 0's), there must be some method of converting human written code into something a computer can understand and execute. This process is called the compilation

process. Human written code is passed to a device known as a compiler, which transforms the code into a parse tree, then a type of code known as assembly, then objective code, and finally machine code. Objective code and machine code are not human-readable; however, assembly code is. The foundation of the model I propose is that most code must be translated to assembly at one point before it can be turned into an executable, or machine code. Therefore, it is theoretically possible to retrain the model to recognize vulnerabilities within assembly code instead of C/C++, and the model should be able to detect vulnerabilities in other languages that also compile to the same style assembly.

I will begin by converting all of the C/C++ code used by Li and Zou as well as a variety of Python and Java code snippets to assembly code. Next, I will employ a standard method to convert this code into x86 assembly (every UNIX machine has a standard set of commands to accomplish this). Once the code has been converted to assembly, I will train a new recurrent neural network (RNN) from scratch to classify these dissembled code snippets as either malicious or benign. The model will then be inherently language agnostic because the RNN was trained on uniform x86 assembly for all C/C++ files.

STS Problem

In 1988, a malicious computer program later named the Morris Worm was released which ultimately infected about 10% of all computers connected to the internet at the time (Radware, 2018). The network which fostered such a large-scale attack was incredibly successful and contained a variety of both human and non-human actors. Upon reflection, the actors in the network seem easy to identify; the FBI identified both Robert Morris (the author of the worm) and the unwitting victims infected by the worm (FBI, 2018). These actors, however,

are not sufficient in understanding how the network managed to be as successful as it did. In fact, all exploits contain the same set of human actors, but the Morris Worm managed to infect 10% of the computers connected to the internet, drastically more than the average exploit manages to reach (FBI, 2018). One crucial actor in the network that allowed the Morris Worm to be so successful that the FBI failed to consider was the worm itself. The worm would reach out to each computer it encountered and simply ask if it already had a copy of the worm running; if the answer was no, the worm would be installed, but if the answer was yes, the worm would still install a copy of itself 1 out of 7 times (Harald, 2018). This decision is what made the worm so challenging to mitigate, because even operating systems that could falsify the existence of the worm could still be infected (FBI, 2018). Ultimately the network was so successful because the associations and interconnections between the unwitting victims and the worm itself were not only incredibly strong, but also nonconsensual. A failure to understand the network as a whole leaves readers vulnerable to similar exploits, while an understanding of the network in its entirety would inherently protect readers from the same exploits.

My analysis of the Morris Worm draws on the science, technology, and society concept of actor-network theory. Actor Network Theory suggests that both human and non-human actors are recruited by a network builder in order to accomplish a particular goal. The Morris Worm, like any network, needed a very specific group of actors and factors in order to be as successful as it was (Callon, 1987). Actor Network Theory will allow me to analyze the human and non-human actors that fostered such a successful exploit. Additionally, using Actor Network Theory I will be able to identify the network builder that was actually responsible for the overall success of the network. In this case, the network builder (the worm itself) recruited unwitting people to be a part of the network without their consent. I will therefore be able to investigate the Morris

Worm with respect to its malicious network builder that ultimately led to the success of the network.

Conclusion

In this paper, the technical and sociotechnical solutions address modern security practices by evaluating the networks in their entirety as they relate to large-scale exploits. I propose the design of a language agnostic deep learning-based system that would allow for vulnerability detection on a larger scale. Such a design would address both the technical and non-technical aspects of the problem. From a technical standpoint, the model would build upon working state-of-the-art technology and adapt it to widen the scope. From a non-technical standpoint, such a design would delegate a large amount of the security currently handled by potentially uneducated users to the hands of very well-versed developers. By placing security in the hands of developers, unwitting users are protected by the software they use. In this sense, the unwitting actors which made the Morris Worm network so successful are inherently protected.

By exploring the Morris Worm via the Actor Network Theory framework, I will be able to identify and explain the network builder at the core of the worm's success. A better understanding of the network builder and the manner in which it recruited its human and non-human actors will protect readers from future exploits that try to take advantage of unwitting victims. Evaluating the Morris Worm through the scope of Actor Network Theory partially bridges the disparity between modern security practices and user integration by exposing the malicious network builder responsible for the extreme success of the network.

Ultimately, I strive to develop a better protection from modern security exploits by designing a model that will address both the technical and non-technical actors in the network.

By addressing these factors holistically, cyber-attack networks such as that of the Morris Worm will be far less successful if not complete failures.

Word count: 1,918 words

References

Callon, M. (1987) Society in the making: The study of technology as a tool for sociological analysis. New directions in the sociology and history of technology (pp. 83-103). MIT Press, Cambridge, Mass.

Clark, L. Probing question: What are computer viruses and where do they come from? *Penn State University*, news.psu.edu/story/141201/2008/07/17/research/probing-question-what-are-computer-viruses-and-where-do-they-come.

Holt, T. What are software vulnerabilities, and why are there so many of them? *The Conversation*, 6 Oct. 2019, theconversation.com/what-are-software-vulnerabilities-and-why-are-there-so-many-of-them-77930.

How does a computer get infected with a virus or spyware? *Computer Hope*, 2 Aug. 2019, www.computerhope.com/issues/ch001045.htm.

Li, Z. et al. VulDeePecker: A deep learning-based system for vulnerability detection. (2018) *Proceedings 2018 Network and distributed system security symposium*, 2018, doi:10.14722/ndss.2018.23158.

Li, Z. et al. VulPecker (2016). *Proceedings of the 32nd Annual Conference on Computer Security Applications - ACSAC '16*, 2016, doi:10.1145/2991079.2991102.

The Morris Worm. *FBI*, FBI, 2 Nov. 2018, www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218.

Radware. DDoS attack definitions - DDoSPedia. *Radware*, security.radware.com/ddos-knowledge-center/ddospedia/morris-worm/.

Sack, H., The Story of the Morris Worm – *First malware hits the internet*. *SciHi Blog*, 3 Nov. 2018, scihi.org/internet-morris-worm/.

Tevis, Jay Evan J., J.A.H. Methods for the prevention, detection and removal of software security vulnerabilities *Proceedings of the 42nd Annual Southeast Regional Conference on - ACM-SE 42*, 2004, doi:10.1145/986537.986583.