Siren: Catching Evasive Malware (Short Paper)

Kevin Borders, Xin Zhao, Atul Prakash
Department of EECS
University of Michigan, Ann Arbor, MI 48109
{kborders, zhaoxin, aprakash}@umich.edu

Abstract

With the growing popularity of anomaly detection systems, which is due partly to the rise in zero-day attacks, a new class of threats have evolved where the attacker mimics legitimate activity to blend in and avoid detection. We propose a new system called Siren that injects crafted human input alongside legitimate user activity to thwart these mimicry attacks. The crafted input is specially designed to trigger a known sequence of network requests, which Siren compares to the actual traffic. It then flags unexpected messages as malicious. Using this method, we were able to detect ten spyware programs that we tested, many of which attempt to blend in with user activity. This paper presents the design, implementation, and evaluation of the Siren activity injection system, as well as a discussion of its potential limitations.

1. Introduction

Malicious programs that send out information over the internet are a major threat on the internet today. In a recent survey, businesses identified spyware as their number one security issue [21]. In addition to spyware, many organizations are concerned about Trojan or backdoor programs that can hide on a computer and give remote access to an attacker, potentially as part of a botnet. Stopping these different types of malicious software (malware) presents a significant challenge to the security community.

One of the biggest problems for security researchers trying to combat malicious software is the plethora of new threats that emerge every day [24]. Most anti-spyware and anti-virus solutions rely primarily on signature detection to identify attacks. Signature analysis systems are inherently *reactive* instead of *proactive*. When a new attack occurs, there will always be a delay before a signature is released. Furthermore, stealthy attacks by skilled hackers that target only a few machines may never be identified; a signature analysis system cannot tell you that you have a problem until it has been detected elsewhere through other means.

To address the shortcomings of signature detection solutions, researchers have developed a number of anomaly detection systems that focus on abnormal behavior rather than specific attacks [2, 13, 16, 20]. These systems are much more effective at uncovering novel threats, but can be evaded by a clever adversary by blending with normal traffic [18], mimicking normal system behavior [19, 26, 28], or emulating normal user activity patterns [2]. In some cases [18, 19], such attacks can even be automated. Efforts have been made to deal with this problem, mainly by increasing the complexity of the IDS [10], but they have only lead to more sophisticated mimicry attacks [19]. We interpret the term "mimicry attack" here in a broad sense, where the attacker attempts to evade an anomaly detection system by emulating normal behavior. The focus of this paper is on detecting malware that mimics human input in particular, but the techniques presented here could be generalized to detect malware mimicking other types of system activity.

In this paper, we present Siren, an activity injection system that collaborates with an intrusion detection system to catch malicious software. Siren operates by injecting human input with the help of virtual machine technology. The input is designed to generate a known sequence of network requests that Siren sends to the IDS. Then, if the actual network traffic differs from what is expected, the IDS raises an alarm. If a malicious program attempts to mimic or blend in with Siren activity, it will be detected by the IDS. A diagram of this process can be seen in Figure 1. An exception here is traffic to sites on a white list. This helps mitigate false positives from trusted programs such as Google Toolbar.

If Siren is able to generate activity that cannot be differentiated from normal usage by an attacking program, and that program continues to emulate activity over time, then the probability of the malware avoiding detection will decay exponentially over time.

If an attacker can distinguish between real and injected input, however, then he or she would be able to evade detection by Siren. One way of doing this is to identify real user activity through an out-of-band channel. An example would be calling a user on the phone and instructing him or her to enter a

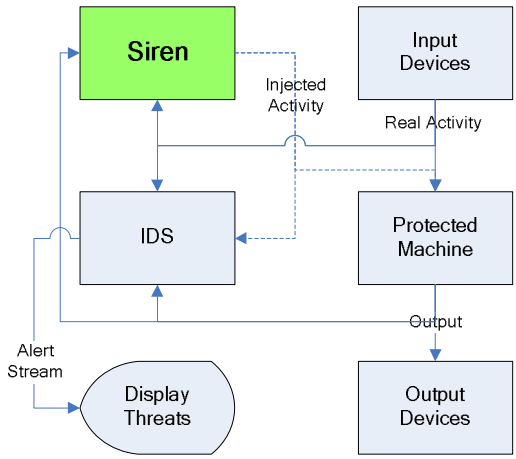

Figure 1. Siren input injection system

predetermined sequence of input that triggers a malware program. In general, these attacks are very difficult to prevent and Siren is unable to stop them. Gathering information out-of-band, however, is also difficult because it is often time-consuming and unreliable.

Automatically identifying real human input in software presents a problem similar to the reverse Turing test, with a few differences. In reverse Turing tests, or CAPTCHAs [4, 7, 23], a computer tries to distinguish between a machine and a human by presenting a challenge problem to the person that he or she can easily solve, but computer cannot. With Siren, an attacking program could freely send traffic if it were able to give a reverse Turing test to the user and the user passes the test. Directly interacting with the user, however, will raise suspicion. The attacking program will have to differentiate between human and computer activity through passive monitoring. For this reason, we describe the task of the attacking program as a *passive reverse Turing test*. To completely escape detection, malicious software would need to perform such a test with a very high level of accuracy.

One of the main challenges identified in this paper is designing an input injection method that can fool malware and prevent it from being able to execute a passive reverse Turing test. This is currently an open problem. For our evaluation, we currently inject a predetermined input sequence and evaluate results by hand. Later in section 4, we present a number of potential approaches for automatically generating human input, as well as weaknesses of each method.

Siren is not the first system that relies on deceiving the enemy. The notion of deception and information warfare can be found in historic literature dating back to 800 B.C. in Sun Tzu's "The Art of War" [27]. An overview of the history of deception and a high-level cognitive model for deception in security is described in [8]. In computer security, much of the use of deception has been limited to attacks. Some examples include stepping stones [29] and social engineering techniques [22]. Specific examples of using deception to defend computer systems include the early work by Cheswick on tracking a hacker [3] and the more recent work on Honeypots and Honeytokens [14, 25]. Siren provides a deeper form of deception at a different level of abstraction (within a host) that is useful against a class of security threats that also uses deception: mimicry attacks.

Input injection is similar to the practice of injecting faults into systems, which has been used in the past for dependability analysis, e.g., see [6]. The goal of previous fault injection methods is to test a system's capability to tolerate failures by artificially inducing them. In contrast, Siren's goal is to inject input that is as close to normal activity as possible so that it cannot be distinguished by an attacker from a real user.

2. Siren Design

One possible way of detecting malware that tries to blend in with normal user web activity would be to monitor web content coming into the browser and human input such as link clicks and typing in URLs. Then, one could compare the resulting network traffic with what is expected and raise an alarm if it differs. Although this method might be very effective and would not require input injection, it would be extremely difficult to implement. Determining what a web browser is supposed to do given arbitrary keyboard and mouse events would require complex modeling of application behavior akin to running the input on a separate machine. This problem is exacerbated by the user's ability to copy and paste data into forms and upload files. Instead of trying to predict what network traffic should result from arbitrary human input, Siren takes the approach of injecting a known sequence of input so that it has control over form data, file uploads, and other browsing activity.

Siren takes advantage of virtual machine (VM) technology to inject input and achieve isolation from the guest operating system, which may be compromised or infected with malicious software. Virtual machines have very useful security properties and can with run low performance overhead [1]. They have been successfully used for inspecting the state of a guest OS without perturbing it [11], and for

Table 1. Evaluation results for ten different types of spyware

Spyware Name	Communicated without input?	Detected by Web Tap	Detected by Siren
AzeSearch			X
Bargain Buddy	X	X	X
CoolWebSearch			X
eXactSearch Bar			X
Gator	X	X	X
Get Mirar			X
Internet Optimizer			X
ISTBar			X
Target Saver	X	X	X
UCmore			X

vulnerability checking [15]. Although VMs can harm security in some cases [12], most of these problems come from running many virtual machines on one host or from reverting to checkpoints. Siren would typically run with only one main VM for the guest operating system that rarely reverts to checkpoints. Another issue with virtual machines is that they are not widely used at this time and would have to be installed to use Siren. However, recent research shows that it is feasible to place an entire operating system inside of a VM without disturbing the OS, significantly hurting performance, or requiring any user interaction [17].

In the Siren architecture, the guest OS contains all of the user's normal files and applications. This is where the user browses the web, sends e-mail, and composes documents. The guest OS is the one most susceptible to infection by spyware, root kits, worms, and other malware. Siren operates below the guest OS on the virtual machine monitor (VMM) in order to isolate itself from these threats. Here it is able to view all device I/O from the guest OS and inject input without being disrupted or detected by the guest OS.

To detect malware, Siren takes advantage of the fact that most legitimate programs rarely communicate over the network when the user is not around. Many computers, however, run a few trusted processes such as automated update software and event notification programs (weather, sports scores, etc.) that generate network traffic when the user is away. If not filtered out, these programs have the potential to generate false positives.

One approach to filtering network messages from trusted applications is to ignore traffic based on process ID. This approach is similar the one taken by many commercial protection programs such as Norton Personal Firewall and BlackIce Defender. Although intuitive, deciding trust by originating process does not work very well in practice; injecting and executing code in other processes is relatively straightforward. Many spyware programs insert libraries into web browsers in order to track the user's browsing patterns, and send private information back to their host servers from within the browser.

Instead of looking at the originating process, Siren can support a *white list* of trusted destination network addresses. For example, if Google Toolbar, Windows Update, and Weatherbug are installed, then Siren would ignore network messages from that workstation to google.com and windowsupdate.com and weatherbug.com, regardless of which application made the requests. Using a white list of trusted addresses, however, may open a hole in the system. If an attacker can passively listen somewhere on the path from the workstation to a trusted server, then he or she can read data sent to the trusted server by a malware. This attack is fairly hard to execute, but networks that are worried about such threats can ban programs like Google Toolbar and Weatherbug, and also run a local update server.

3. Evaluation

A major security threat that Siren aims to eliminate is spyware. We installed ten different types of spyware on a Windows PC instrumented with Siren to evaluate its effectiveness. We also installed Web Tap [2] on the system for the purpose of comparison. Table 1 summarizes our results for the ten spyware programs. For the first phase of the experiment, we ran Siren without injecting any additional input to see how many of the spyware programs generated network traffic when the user was away. Three out of the ten programs (BargainBuddy, Gator, and TargetSaver) did communicate with their home servers in the absence of human input and were detected immediately by both Siren and Web Tap. Web Tap, however, failed to detect the other seven spyware programs because they made only few web requests to blend in with normal browsing activity.

Using systems such as BINDER [9], Norton Personal Firewall, and BlackIce Defender would have been an alternative approach to detecting the spyware programs. However, they are also unable to identify the seven spyware programs missed by Web Tap because these programs run as plug-ins inside of the browser, which is a trusted process receiving legitimate input. Because Siren uses input injection, it is able to detect spyware embedded in a web browser.

For Siren's evaluation, we manually created a sequence of web activity and replayed it with each of the spyware programs installed. Siren then ran a script to compare the websites visited during each input run. Siren flagged request to sites that were not visited in the original input run as malicious. Using this heuristic, Siren identified all ten of the spyware programs, even those that ran inside of the browser and were not detected by Web Tap.

At the present time, we have not tested Siren against any programs that analyze web browsing patterns to mount a sophisticated mimicry attack because we are unaware of any such programs. However, several spyware programs in our tests only communicate when the user is actively browsing in order to blend in with normal traffic.

4. Realistic Input Generation Challenges

For our evaluation, we injected input by opening a web browser window and manually traversing web pages. Ideally, Siren should be able to automatically generate and inject human input. There are a number of challenges, however, in developing such a system. One major issue is ensuring that the injected input does not cause any negative side-effects. Examples include making a purchase over the internet, sending out an e-mail, or modifying an important document. Another problem is configuration changes on the local machine, such as moving programs and adjusting window sizes, that affect the result of human input. Finally, the injected input should be difficult for an attacker to distinguish from real input.

One possible solution is to manually record a sequence of human input and the resulting network output on a single clean computer. Then, Siren could replay the input on other machines, raising an alarm if the content or destination of requests changes between runs, except as expected by the protocol (If a message contains the time or special state such as a browser cookie). With this solution, one must worry about differences between computers, such as program locations, which will affect the result of the input. One method of dealing with this issue is to fix settings across all computers on the network. Another is to re-record input for each configuration or when a user changes a critical setting on his or her computer.

Another problem with recording and replaying input is dynamic web content. Many web pages change frequently, and this will disrupt the network traffic. One could get around this problem by visiting relatively static sites and re-recording input when they change. Avoiding dynamic sites and re-record input in response to content changes is a significant limitation of this approach, and could be exploited by an attacker to identify real user activity.

With manually recorded input, the administrator can make sure that it does not cause any unwanted state changes. However, an attacker may be able to identify real activity by patiently waiting for the user to perform some state-changing operation. Also, if Siren repeatedly injects the same sequence of input, then an attacker will be able to tell that it is fake.

Another way of generating input is to use an algorithm that imitates human behavior. This algorithm would have to know which raw input events will trigger which high-level actions such as text selection and link traversal. It can then infer the expected output based on the content returned by the server and on application behavior. Generating correct input would require either some video processing, which could be very hard, or virtual machine introspection [11] to obtain the locations of objects on the screen for mouse input.

Automatic input generation may be vulnerable to natural language analysis by a malware program. If malware is able to identify input referring to the current time or recent events in natural language, then it may be able to conclude that the real user is present. However, this would have to be done by the attacking program *without communicating over the network*, which is a hard problem. If the malware program tries to send information to its owner without first making sure that it is safe, then it is likely to be detected by Siren.

All of the input injection methods discussed so far also have issues with disturbing state on the local machine. Avoiding state changes altogether in fake input would help a patient attacker to identify real activity. Another solution is to undo changes right before the user returns by either injecting input to undo them (such as closing a browser window) or reverting to a snapshot that was taken earlier. Recent work shows that snapshot restoration can be done efficiently [5]. Undoing state changes, however, is detectable and allows malware to build a schedule from past usage in order to predict future activity. State changes on a remote server also cannot be undone, and need to be addressed separately.

Another way of avoiding detrimental state changes is to create a clone of the current machine and run it instead of the original. Siren could inject input on the clone that damages the state because it would be thrown away later on. Again, malware can create a schedule and attempt to predict future usage by recording and examining old input. (If the input is old, then it is likely to have been entered before the clone operation and thus by the real user.)

For all of these input generation systems, it is currently an open question how difficult it would be for an attacker to identify real human input. However, the attacking program has the extra burden of needing to be very sure that it is correct. If the probability that the input is actually real when the malware program thinks it is, call it *p*, is much less than 1, then the probability of remaining undetected decreases exponentially over time. The program could still have pretty good odds if it is able to package up very large amounts of data into a single burst. This attack can be easily prevented, however, by employing a rate-limiting anomaly detection system such as Web Tap [2] in parallel with Siren. Now, if the probability of correctness for the malware program is .95, or it is correct 95% of the time about the user being present, then it still would have over a 50% chance of being detected after sending 14 times. To be successful, Siren only needs to fool the attacker some of the time.

5. Siren's Security Analysis

There are a few different methods that an attacker could use to avoid or prolong detection by Siren. If the input generation algorithm is weak, then Siren can take advantage of its shortcomings to learn characteristics of real input or even identify it immediately. For example, if input is only injected into cloned virtual machines, then malware can look at older data to see real a user's input and predict when he or she will be active next. If the generation algorithm does not act sufficiently human, malware may be able to use this to its advantage. If the algorithm never uploads files, for example, then a file upload will let the malware know that it is safe to send data over the network.

As mentioned earlier, one of the preconditions for Siren to eventually detect a piece of malware is that the malware must continually send network messages over time. A potential vulnerability in the Siren system is an attack that requires user-interaction to succeed, delivers the payload immediately, and exits. One example of such a threat is an e-mail worm. If the worm only opens up when the user clicks on an e-mail, sends e-mails to everybody in the address book, and then terminates very quickly, Siren would have a very hard time detecting it. One way of mitigating this vulnerability is to deploy additional deception methods, such as adding bogus addresses to the e-mail client's address book.

Another precondition for Siren to be able to detect a piece of malicious software is that the attacker cannot determine usage through an out-of-band channel. A hacker could call up a user on the phone and tell that person to enter a particular sequence of input triggering some malware. The hacker could also talk to the user over an instant messaging program to guarantee that the user is active on his or her computer. Although these and other social engineering techniques could help an attacker figure out when a user is present, and Siren has no way to stop them, they are not trivial to execute and usually require association of a phone number and name with an IP or e-mail address.

Finally, malicious software could hide data in a covert channel. One example would be modifying TCP flags in outbound traffic. However, these channels are often very low bandwidth and would require the attacker to be passively listening on the path of a connection.

6. Conclusion

Traditional behavioral analysis techniques are able to detect many novel threats, but are vulnerable to mimicry attacks. In this paper, we presented Siren, a detection system that looks at human input to identify malicious network traffic. Siren also injects bogus human input to confuse and disrupt attackers who try to mimic legitimate programs by waiting for user input to send network messages.

In this paper we discussed possible methods for creating realistic human input, and addressed potential shortcomings such as inadvertent state modification. For our evaluation, we manually entered a predetermined sequence of input on a clean machine and on virtual machines with ten different types of spyware installed. Siren was able to detect all ten spyware programs, while a competing anomaly detection system that does in inject input was only able to detect three of the ten. Finally, we discussed methods of attacking Siren itself such as out-of-band information gathering and identifying real activity by waiting for state-changing operations.

7. Acknowledgements

We acknowledge support from the Intel Corporation and the National Science Foundation. We also thank the reviewers for their helpful comments.

8. References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symp. on Operating Systems Principles*, pp. 164-177, 2003.

[2] K. Borders and A. Prakash. Web Tap: Detecting Covert Web Traffic. In *Proc. 11th ACM Conf. on Computer and Communications Security,* pp. 110-120*,* 2004.

[3] W. Cheswick. An Evening with Berferd In Which a Hacker is Lured Endured and Studied. In *Proc. of the Winter Usenix 92 Conference*, 1992.

[4] M. Chew and J.D. Tygar: Image Recognition CAPTCHAs. In *Proc. 7th Information Security Conference*, 2004.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. *Proc. 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation*, 2005.

[6] J. Clark and D. Pradhan. Fault Injection: A Method for Validating Computer-System Dependability. *Computer*, 28(6):47-56, June 1995.

[7] A. Coates, H. Baird, and R. Fateman. Pessimistic Print: A Reverse Turing Test. In *Proc. Sixth Intl. Conf. on Document Analysis and Recognition*, pp. 1154, 2001.

[8] F. Cohen, D. Lambert, C. Preston, N. Berry, C. Stewart, and E. Thomas. A Framework for Deception. http://www.all.net/journal/deception/Framework/Framework.html, July 2001.

[9] W Cui, R. Katz, and W. Tan. BINDER: An Extrusion-Based Break-In Detector for Personal Computers. In *Proc. USENIX Annual Technical Conference,* 2005.

[10] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly Detection Using Call Stack Information. In *Proc. IEEE Symp. on Security and Privacy*, 2003.

[11] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proc. ISOC Symp. on Network and Distributed System Security*, 2003.

[12] T. Garfinkel and M. Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Machine-based Computing Environments. In *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS-X)*, May 2005.

[13] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[14] HoneyPots. http://www.honeypots.org, 2005.

[15] A. Joshi, S. King, G. Dunlap, and P. Chen. Detecting Past and Present Intrusions through Vulnerability-Specific Predicates. In *Proc. Symp. on Operating Systems Principles,* 2005.

[16] G. H. Kim and E. H. Spafford. The Design and Implementation of Tripwire: a File System Integrity Checker. In *Proc. ACM Conf. on Computer and Communications Security*, 1994.

[17] S. King and P. Chen. Subvirt: Implementing malware with virtual machines. In *IEEE Symp. on Security and Privacy*, Oakland, California, May 2006.

[18] O. Kolesnikov, D. Dagon, and W. Lee. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Georgia Tech Technical Report, GIT-CC-04-15, 2004-2005.

[19] C. Kruegel and E. Kirda. Automating Mimicry Attacks Using Static Binary Analysis. In *Proc. 14th USENIX Security Symposium,* pp. 161–176, 2005.

[20] Y. Liao and V. R. Vemuri. Using Text Categorization Techniques for Intrusion Detection. In *Proc. 11th USENIX Security Symposium*, pp. 51-59, 2002.

[21] MessageLabs. MessageLabs Survey Finds Spyware No. 1 Web Security Issue for Australian Businesses. http://www.computerworld.com.au/index.php/id;478111644, 2005.

[22] K. Mitnick and W. Simon. *The Art of Deception: Controlling the Human Element of Security.* Wiley (2002).

[23] M. Naor. Verification of a Human in the Loop, or Identification via the Turing Test. *Unpublished manuscript*, http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human_abs.html, 1996.

[24] B. Schneier. Attack Trends: 2004 and 2005, *Queue*, June2005. http://www.schneier.com/blog/archives/2005/06/attack_trends_2.html.

[25] L. Spitzner. The Other Honeypot, July 2003. http://www.securityfocus.com/infocus/1713.

[26] K. Tan, K.S. Killourhy, and R.A. Maxion. Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits. In *Proc. Fifth Intl. Symp. on Recent Advances in Intrusion Detection (RAID)*, pp. 54-73. Lectures Notes in Computer Science #2516, Springer-Verlag, Oct. 2002.

[27] Sun Tzu. The Art of War (translated by James Clavell). Dell Publishing, New York, 1983.

[28] D. Wagner and P. Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *Proc. 9th ACM Conf. on Computer and Communications Security*, pp. 255-264, 2002.

[29] Y. Zhang and V. Paxson. Detecting Stepping Stones. *In Proc. 9th USENIX Security Symposium*, pp. 171-184, August 2000.