Flowers for Automated Malware Analysis

Chengyu Song and Paul Royal
Georgia Institute of Technology
csong84@gatech.edu, paul@gtisc.gatech.edu

ABSTRACT

To handle the large volume of malware samples collected each day, numerous automated malware analysis techniques have been developed. In response, malware authors have made analysis environment detections increasingly popular and commoditized. In turn, security practitioners have created systems that make an analysis environment look like a normal system. Thus far, neither side has claimed a definitive advantage.

In this paper, we demonstrate techniques that, if widely adopted by the criminal underground, would permanently disadvantage automated malware analysis by making it ineffective and unscalable. To do so, we turn the problem of analysis environment detection on its head. That is, instead of trying to design techniques that detect specific analysis environments, we instead propose malware that will fail to execute correctly on any environment other than the one originally infected.

1. INTRODUCTION

Malware analysis is the process of understanding the behavior of malicious programs. As intelligence produced by malware analysis systems and tools [3, 7–9] can endanger the cyber criminal's profit model, attackers have continuously developed techniques to prevent malware from being analyzed. In response, defenders have created new techniques [5,11–13] to address malware analysis resistance. Thus far, neither side has yet claimed a definitive advantage.

In this paper we propose techniques capable of rendering automated malware analysis ineffective and unscalable. Specifically, we introduce concepts that, at the host and network levels, interrelate the successful execution of a malware sample with the unique properties of the original host it infects. Note that these techniques are not intended to prevent a human analyst from employing manual efforts to understand the behavior of a particular piece of malware (e.g., Stuxnet).

2. DEFEATING AUTOMATED MALWARE ANALYSIS

The decades-old antivirus model consists of four phases: artifact collection, analysis, signature generation, and runtime detection. The techniques we describe in this section exploit the observation that collection (e.g., via antivirus client submission or malware exchange) and analysis occur in two different environments. Instead of trying to detect a particular analysis environment, a sample can mitigate the analysis step by failing to correctly execute in any environment that is different from the original one infected. This goal is achieved through two techniques: host identity-based encryption (HIE) and instruction set localization (ISL).

2.1 Host Identity-based Encryption (HIE)

Before deploying a malware instance on a given system, information (based on system hardware and software) that can uniquely identify this system is collected. This information is then used to derive a key (or host ID) that will be used to encrypt certain portions of the malware instance. At runtime, the malware instance will gather the same set of information again and use it to derive a decryption key. Thus, if the instance is put into a different execution environment, decryption will fail and the sample will not exhibit malicious behavior.

Instead of using the host ID to protect the entire binary, the most appropriate use of this technique may involve encryption of only mechanisms critical to the malware. As an example, portions of code or data associated with the sample's domain name generation algorithm (used to contact C&C servers) could be encrypted. If decryption fails (e.g., during the analysis phase), the sample will attempt to resolve or connect to the wrong C&C server. The malware analysis system would in turn treat this information as real.

HIE has two major advantages. First, it uses modern cryptography, which means that knowledge of how a key is derived does not affect the integrity of the protection. Unless the defender can guess the same decryption key, they cannot unlock the sample. Second, any two instances of malware will possess different decryption keys, which means that intelligence gathered from successfully analyzing one malware instance provides no advantage in analyzing the second.

Information used to generate the host ID must be unique, invariant, and require no privilege to obtain. In Windows, the following could be used:

- **The Environment Block**. When a process is created, Windows stores environment information in the process' address space. In our design we use the process owner's username, computer name, and CPU identifier. As the environment block is directly accessible by code that executes inside a given process, this information can be easily obtained.

- **MAC address**. The MAC address of the NIC can be obtained from the `GetAdaptersInfo` API.

- **GPU info**. GPU information can be obtained from the `GetAdapterIdentifier` method of `IDirect3D9Ex` interface. In our design, we use the device description.

- **SID of the user**. Using the token of a process, the `GetTokenInformation` API can be used to obtain the SID of the process' owner. This identifier is unique across a Windows domain.

Once collected, this information can be concatenated and used as an input to a cryptographic hash function (e.g., bcrypt [10]) to generate the host identifier, which will then be used as the encryption/decryption key.

In addition to generation of the host identifier, deployment logistics must be worked out that include the use of intermediate code agents that determine the host ID to which delivered malware instance will be bound. Solutions to this problem are complicated by exploit reliability concerns that mandate shellcode be as small as possible. Thus, gathering information used to produce the host identifier must instead be deferred to an intermediate downloader agent.

If security practitioners capture exploit shellcode or the intermediate downloader, they could use these agents to obtain a malware instance bound to their analysis environment. To solve this problem, we propose using one-time URLs similar to those offered in password reset procedures. More specifically, before the shellcode or downloader is sent to the infected host, the server will assign it a unique path to download the next stage. Although the operational specifics will vary based on the attack vector (e.g., drive-by download versus email attachment), in all cases the one-time URL will also be short-lived.

2.2 Instruction Set Localization

Despite its advantages, host identity-based encryption is not considered sufficiently resistant to forgery. Thus, we also propose a network-based identifier that is derived at the C&C server (and thus intractably difficult to forge). Specifically, we selected the following:

- **Geo-location**. The IP address is the most straightforward candidate for the network identifier, but is not sufficiently stable. For this reason the geo-location of the IP address should be used instead, at the granularity of state or province.

- **Autonomous System Number (ASN)**. In general, geo-location alone comprises a sufficient network identifier. However, as the publication of this information is not mandatory, geo-location databases can contain outdated or incorrect data. For this reason, the ASN should be used as well.

The combination of host and network-based keys are used by instruction set localization (ISL), a second technique that provides a malware instance running on an infected system with its actual malicious behavior.

Before detailing instruction set localization, brief mention of instruction set virtualization (ISV) is merited. ISV (e.g., as used in VMProtect [2] and Code Virtualizer [1]) is an obfuscation technique that protects software by transforming the source code, intermediate representation, or native machine code of a program into bytecode for an arbitrarily chosen instruction set architecture. At runtime, the execution semantics of the original program are fulfilled by a native interpreter bundled with the bytecode.

At a high level, instruction set localization binds (or localizes) a virtualized instruction set to a specific environment. In this scenario, a malware instance deployed on a given system represents only an interpreter of bytecode for a virtualized instruction set. All malicious tasks, which will be requested from and provided by the C&C server, represent bytecode to be interpreted.

The interpreter's request for a task includes the host identifier of the infected system. The C&C server combines the host identifier with a network identifier and uses this information as part of virtualizing the native code representing the malicious task. The bytecode given to the infected host will thus only run on that specific host, as determined by forgery-resistant host and network-based identifiers.

In instruction set localization, each malware instance receives a *task decryption key* derived from the unique ID (the combination of host and network-level identifiers) when deployed. When responding to a task request, the C&C server will encrypt the task using a key derived from the malware instance's unique ID. If there is a mismatch (e.g., in the network identifier used to create the unique ID), the decryption routine will generate invalid or incorrect bytecode that does not reveal the malicious task. To prevent attacks associated with the use of the task decryption key, its derivation should use the unique ID as input to a keyed hash (e.g., HMAC); the private key used in the keyed hash would be kept on the C&C server and known only to the botnet operator.

Like host identity-based encryption, instruction set localization offers several advantages. First, unless the interpreter deployment-time (or infection-time) signature matches the runtime signature, the task cannot be executed correctly due to incorrect bytecode interpretation. Second, the only way to understand the task is to correctly determine its interpretation, such as by brute-forcing the combination of host and network identifiers.

3. DISCUSSION

Operational Security. Both HIE and ISL are implemented using modern cryptography and thus are immune to knowledge of how keys are generated– the only way to break their protections is to derive the correct keys. As security organizations automatically analyze malware in environments separate from those originally infected, derivation of the correct keys requires searching through the entire key space, which is of non-trivial size. Moreover, some configuration information (i.e., that used to derive the network identifier) may be impossible to duplicate.

Another advantage of HIE and ISL is that they are insensitive to analysis techniques. That is, regardless of the employed analysis granularity (e.g., fine-grained dataflow analysis used in [14] or high-level, blackbox network intelligence collection), the resistance offered by HIE and ISL can be broken only if the configuration parameters of the original execution environment are successfully matched.

Potential Countermeasures. One straightforward idea for bypassing the protections provided by HIE and ISL is to analyze samples in the original environments they infected.

While such an approach may work for samples collected by high-interaction honeypots, for a variety of practical reasons the use of this method is not feasible for other sources. Challenges include monitoring system capability limitations (e.g., of low-interaction honeypots), legal and privacy considerations and impact on business operations and continuity (e.g., for client submissions). As samples collected by high-interaction honeypots represent only a small portion of all collected samples, the effectiveness of this approach is limited.

An alternative to analyzing malware on the systems originally infected is the collection and duplication of host and network-level environment information. However, for similar (though perhaps less significant) policy and privacy reasons, the implementation of this idea would face significant hurdles. Moreover, even if the host identifier can be successfully forged, duplication of the correct network identifier would require analysis system deployment on an unprecedented and globally cooperative scale.

Another potential countermeasure is to record and collect the network activity between an infected host and the C&C server, then replay that communication during analysis. Without one additional protection, this approach would bypass ISL and could be combined with attacks or cooperative efforts to forge the host identifier. However, the use of SSL/TLS for C&C communication mitigates the successful use of this response.

Finally, the very manner by which HIE and ISL protect a malware instance could be leveraged by the security community to create instability in a set of host or network identifiers and thus prevent successful or correct execution (i.e., the allergy attack [4]). However, this countermeasure could also make legitimate software systems that use the same information equally unreliable. As such, the success of this response may depend on the willingness of users to accept security over usability.

4. CONCEPT INTEGRATION

In September 2011, Flashback [6] emerged as malware that targeted Mac OS X. By April 2012, the botnet representing Flashback-infected systems had grown to over 600,000 Macintosh systems. The initial Flashback agent connects to its C&C server and downloads one or more additional payloads (e.g., those that can illegally monetize the victim's use of search engines). When requesting a payload, the agent submits the hardware UUID of the infected system. This value is then hashed (via MD5) to create a key that, in combination with the RC4 stream cipher, binds the payload to that system. Like HIE, unless the hardware UUID of the system matches the one used to create the payload, it will not execute successfully.

Flashback's use of an infected system's hardware UUID as a decryption key demonstrates that malware authors have already begun using protections like those described in this paper. Security researchers must therefore prepare for a possible future of malicious software that will only run on the systems it originally infects.

5. CONCLUSION

In this paper we proposed two obfuscation techniques–host identity-based encryption (HIE) and instruction set localization (ISL)–that make the successful execution of a malware sample dependent on the unique properties of the original host it infects. Going forward, researchers must include ways to mitigate these protections or examine alternatives to threat detection and analysis. To highlight the current and future importance of the associated concerns, we briefly discussed the Flashback botnet's use of a similar technique to prevent the automated analysis of its samples.

6. REFERENCES

[1] Code Virtualizer Overview. http://oreans.com/codevirtualizer.php.

[2] VMProtect Software Protection. http://vmpsoft.com.

[3] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM Conference on Computer and Communication Security*, 2009.

[4] S. Chung and A. Mok. Allergy attack against automatic signature generation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection*, 2006.

[5] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and Communications Security*, 2008.

[6] Intego. Mac flashback trojan horse masquerades as flash player installer package, September 2011.

[7] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In *Proceedings of the USENIX Security Symposium*, 2006.

[8] C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th conference on USENIX security symposium*, 2009.

[9] A. Lanzi, M. I. Sharif, and W. Lee. K-tracer: A system for extracting kernel malware behavior. In *Proceedings of the 16th Symposium on Network and Distributed System Security (NDSS'09)*, 2009.

[10] N. Provos and D. Mazieres. A future-adaptable password scheme. In *In Proceedings of the 1999 USENIX Annual Technical Conference, FREENIX Track*, 1999.

[11] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22nd Computer Security Applications Conference (ACSAC)*, 2006.

[12] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Automatic reverse engineering of malware emulators. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[13] A. Vasudevan and R. Yerraballi. Stealth Breakpoints. In *Proceedings of the 21st Computer Security Applications Conference (ACSAC)*, 2005.

[14] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007.