Rethinking Antivirus: Executable Analysis in the Network Cloud

Jon Oberheide, Evan Cooke, Farnam Jahanian
Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor, MI 48109
{*jonojono, emcooke, farnam*}*@umich.edu*

ABSTRACT

Antivirus software installed on each end host in an organization has become the de-facto security mechanism used to defend against unwanted executables. We argue that the executable analysis currently provided by host-based antivirus software can be more efficiently and effectively provided as an *in-cloud network service*. Instead of running complex analysis software on every end host, we suggest that each end host run a lightweight process to acquire executables entering a system, send them into the network for analysis, and then run or quarantine them based on a threat report returned by the network service. An executable analysis service run inside an enterprise network or by a service provider could integrate antivirus software, behavioral simulation, and other analysis engines from multiple vendors providing better detection of malware and simplify client software enabling deployment on a broader range of devices. To explore this idea we construct a prototype composed of a Windows based host agent and an in-cloud analysis service and evaluate it using a diverse dataset of 5066 unique malicious executables. By correlating information between multiple detection engines, our system provides over 98% detection coverage of the malicious executables using eight antivirus engines and two behavioral engines compared to a 54% to 86% detection rate using the latest commercial antivirus products.

1 INTRODUCTION

Detecting malicious software has become an increasingly challenging problem. While a single antivirus engine may be able to detect many types of malware, 0-day threats and other obfuscated attacks [12] can frequently evade a single engine. We argue that the executable analysis service currently provided by host-based antivirus software can be more efficiently and effectively provided as an *in-cloud network service*. Instead of running complex analysis software on every end host, we suggest that each end host run a lightweight process to acquire executables entering a system, send them to a network service for analysis, and then run or quarantine them based on a threat report returned by the network service.

Centralizing the analysis of suspicious software as a network service has several advantages over existing host-based antivirus software.

1. **More is better**: A network service can employ a cluster of servers to quickly analyze executables using multiple techniques. For example, instead of using a single antivirus engine, a network service can run a wide range of antivirus engines in parallel while also performing behavioral analysis [3, 8]. Additional detection engines can easily be integrated into a network service, allowing for considerable extensibility. Such comprehensive analysis can significantly increase the detection coverage of malicious software.

2. **Keep it simple stupid**: By significantly lowering the complexity of host-based monitoring software, a number of advantages are realized. Clients no longer need to continually update their local signature database, reducing administrative cost. Simplifying the client software also decreases the chance that it could contain exploitable vulnerabilities [5, 13]. Finally, lightweight client software allows the service to be extended to mobile and resource-limited devices that lack sufficient processing power but remain an enticing target for malware.

3. **Sharing is caring**: Correlating information between engines is enabled when multiple engines are run within a network service and can be beneficial to detection coverage. For example, if a behavioral system finds that the behavior of an unknown executable has similar behavior to an executable previously classified as malicious by antivirus engines, the unknown executable can be quarantined.

This paper investigates how to design a network service for executable analysis. We construct a prototype composed of a Windows-based host agent and an in-cloud network service. Using a recent dataset of 5066 unique malicious executables [7], we demonstrate how the system detects 98% of the samples using eight antivirus engines and two behavioral engines in parallel, compared to a 54% to 86% detection rate using the latest commercial antivirus products. We also show that the average time to analyze a legitimate or malicious exe-

cutable, including network latencies, is less than a second.

2 APPROACH

The use of traditional signature-based approaches for the problem of detecting malicious software in the network and on the host has become increasingly difficult. The elevating sophistication of modern malicious software poses a significant challenge for any single vendor to develop signatures for every new piece of malicious software. Indeed, a recent Microsoft survey found more than 45,000 new variants of backdoors, trojans, and bots during the second half of 2006 [6].

In response, a new generation of systems has been developed to monitor the behavior of suspicious applications to identify malicious binaries. Sandbox services such as Norman [8] and CWSandbox [3] enable the analysis of suspicious binaries and produce a detailed report of a binary's actions in a simulated environment. In addition, virtual machines have been used to analyze malware and track malicious behavior [1, 10, 14]. While behavioral and VM-based detection techniques improve our ability to detect malicious software, they can be resource intensive and are not designed to protect every end host.

In this paper, we explore how to improve the detection of malicious software by providing executable analysis as an in-cloud network service. We envision a vendor-agnostic service that is deployed in a high-speed, low-latency network such as inside an enterprise or by a service provider. Such a centralized service would integrate detection engines from many vendors and include an end host component for the automated acquisition of binaries. The network service is similar to other in-cloud protection mechanisms such as email [2, 4] and HTTP [11] filtering, except that we use a dedicated end host agent which enables our system to protect against executables that arrive using different protocols and devices.

3 ARCHITECTURE

Performing the analysis of executables using a network service is not a simple task. Suspicious executables must be acquired and isolated so that they can be sent to an upstream analysis service. The analysis service must be efficient, and the execution of malicious executables must be prevented.

To solve these problems we envision an architecture that includes two major components. The first host-based component acquires executables and sends them to the network service, and the second network-based component identifies malicious executables and returns a threat report to the end host. Figure 1 shows a high level design of the system architecture. It is important to point out that we do not see our system replacing existing antivirus or intrusion detection solutions. Rather, our malware analysis service runs as an additional layer of protection to augment existing security systems inside an organizational network such as an enterprise.

3.1 Executable Identification and Acquisition

Malicious executables can enter an organization using a range of different techniques. For example, mobile devices (e.g., a USB stick), email attachments, downloads (wanted or unwanted), and vulnerable network services are all common entry points. Due to the broad range of entry vectors our system includes a lightweight executable acquisition system that is run on each end host.

Just like existing antivirus software, the client module runs on each host and inspects each executable. The execution of any binary is trapped and diverted to a handling routine first. The handling routine begins by hashing the binary and checking the hash against a local and remote whitelist and blacklist. If the hash does not match in the whitelist or blacklist, the entire binary is sent securely to the in-cloud service for analysis.

To minimize the time between when a user downloads an executable and receives a response from the network service, the architecture provides a method to send a binary for analysis as soon as it is written or changed on the end host system (e.g., via file-copy, installation, or download). Doing so amortizes the transmission and analysis cost over the time elapsed between binary creation and user-initiated execution.

3.2 In-Cloud Executable Analysis

The second major component of the system is the network service responsible for malware analysis. The job of the analysis service is to receive an executable and return a threat report with information about whether the executable is safe to run.

3.2.1 Executable Analysis

Rather then choosing one single technique to detect malicious executables, we suggest that a network service run multiple engines in parallel. Unlike host-based analysis systems that must meet relatively tight storage and computational constraints, a network service can easily scale to meet the demand of multiple analysis engines.

While the specific backend analysis techniques used are independent of the proposed architecture, two general classes seem particularly well-suited to the proposed approach.

- **Antivirus Engines:** There is a wide range of antivirus products commercially available today, and as we will show, they differ significantly in their detection coverage. By analyzing each candidate executable with multiple antivirus engines, the network service is able to provide better detection of malicious software.

Figure 1: Architectural approach for in-cloud executable analysis.

- **Behavioral Analyzers:** A second major method of testing executables is behavioral analysis. A behavioral system executes suspicious executables in a sandboxed environment [3, 8] or virtual machine [1] and records host state changes and network activity.

While performing analysis in parallel is far more efficient then serial execution, delay can still exist between when an executable is submitted and when the result is returned. The architecture employs a caching mechanism to better deal with associated latencies.

The caching mechanism is simply a whitelist/blacklist database of hashes of previously analyzed binaries. The idea is that executables run in an organization will frequently be similar across different hosts so keeping a cache of previously analyzed binaries can significantly improve system performance. A cache hit in either the whitelist or blacklist would therefore result in an immediate server response indicating whether the client should execute the binary.

3.2.2 Executable Threat Report

The results from the executable analysis engines are synthesized into a threat report returned to the client over a secure and authenticated channel. The report includes three distinct sections.

- **Execution Directive:** a boolean indicating whether the executable should be run. The formula used to set this value is customizable as described above but the simplest mode is to set the execution directive to "do not run" if any detection engine classifies an executable as malware.

- **Family/Variant Labels:** a list of family/variant labels assigned to the malware by the different analysis engines.

- **Behavioral Analysis:** a list of host and network behaviors observed during simulation. This may include information about processes spawned, files and registry keys modified, network activity, and other state changes.

When the client receives the threat report, it will either begin execution of the binary or prevent execution and present the report to the user. This, of course, assumes that the client module has not been compromised. Just as with antivirus software, we assume that integrity of the client is maintained as a trusted module.

4 Implementation and Evaluation

We constructed a prototype implementation of the proposed architecture that includes an executable acquisition system for the Windows platform and an in-cloud network service for analysis. This section describes how we implemented the prototype and explores the detection capabilities and performance of the system.

4.1 System Implementation

4.1.1 Client Module

We implemented a lightweight client module to detect executables written to disk and then send them to the network service for analysis. The notification for file system writes is provided by the `ReadDirectoryChangesW` Win32 API call, a mechanism similar to `inotify` on Linux. Each file written is scanned for a valid Portable Executable (PE) header to verify whether it is an executable. The executable is then hashed using the SHA-1 algorithm and compared to both the local and remote whitelist and blacklist. Local whitelists are seeded with hashes of the default set of executables included with common Microsoft Windows installations to reduce remote whitelist lookups.

The client module also prevents the execution of binaries that have been identified as malicious. By hooking the `CreateProcess` Win32 API call, we interpose on the creation of new processes and halt the execution of unwanted code.

4.1.2 Network Service

The second component of the prototype is a network service responsible for analyzing executables using multiple detection engines and returning a comprehensive threat report on each executable. Incoming executables from

AV Vendor	Version	Signatures	Detection Results
Avast	4.7.1001	000740-0	84.7%
ClamAV	0.90.2	3224	59.7%
F-Prot	6.0.7.0	4.3.3	79.9%
F-Secure	7.01.128	N/A	86.6%
Kaspersky	6.0.2.621	N/A	85.3%
McAfee	5100.0194	5027.0000	54.9%
Symantec	14.0.3.3	N/A	81.9%
Trend Micro	15.00.1433	4.459.00	82.0%

Table 1: The antivirus protection engines, the versions used in the prototype, and the percentage of malware samples detected.

#	Detected	Antivirus Products
1	86.59%	F-Secure
2	92.93%	Trend, Avast
3	94.63%	Trend, F-Secure, Avast
4	95.34%	ClamAV, Symantec, Trend, Avast
5	95.85%	ClamAV, Symantec, Trend, F-Secure, Avast
6	96.15%	F-Prot, ClamAV, Symantec, Trend, F-Secure, Avast
7	96.23%	Mcafee, F-Prot, ClamAV, Symantec, Trend, Kaspersky, Avast
8	96.23%	*all*

Table 2: Detection coverage of the malware samples using between one and eight antivirus products.

end hosts are assigned to a request broker which is responsible for delivering the executable to each analysis engine, collecting the results, and returning the threat report back to the client.

We use eight antivirus and two behavioral engines in our prototype. The eight antivirus engines are listed in Table 1, and the behavioral analyzers include Norman Sandbox Analyzer [8] and the behavioral profiling system described in [1]. The detection engines are run in parallel in virtualized environments.

Our prototype also includes an optimization aimed at reducing the latency perceived by end users when running newly obtained executables. We implemented a network stream sensor that promiscuously monitors a network tap (e.g., a switch span port) to acquire executables. By deploying such a component, executables can be extracted out of network transmissions on the fly and analyzed by the in-cloud network service before even reaching the destination end host. Clearly this approach does not speed up the analysis of all executables as network traffic can be encrypted and the sensor currently handles only a handful of common network protocols.

4.2 Deployment and Results

4.2.1 Detection Coverage

To evaluate the detection capabilities of the prototype we obtained over 5000 malware executables from Arbor Network's Arbor Malware Library (AML) [7]. The AML contains malware which has been captured in the wild with a variety of techniques such as honeypot vulnerability emulation, spam traps, and honeymonkey spidering. The dataset consists of 5066 unique executables spanning an eight month collection period from September 2006 to May 2007.

We first measured the detection rates of each antivirus engine individually across the entire malware dataset. The antivirus signatures were all updated on the same date, May 9th, for consistency, and their respective versions are listed in Table 1. The results of using each engine individually are listed in the last column of Table 1. The single engine detection rates vary from as low as 54% up to 86%.

We then looked at the benefit of using multiple antivirus engines to analyze the same set of malware. Table 2 summarizes the results. Using all eight antivirus engines in our network service, we were able to detect 4875 executables as malicious, resulting in a detection rate over 96%. The dataset used in the paper only includes known malicious executables and so exploration of false positives is an open research question.

While the eight antivirus engines were able to identify 96% of the executables as malicious, 191 executables were not flagged. This last 4% is arguably the most difficult for current detection systems to identify. Fortunately, the prototype also includes behavioral analysis and the capability to share information between detection engines. Behavioral reports on the 191 executables revealed that 92 had similar behavior to known malicious binaries already identified by the antivirus software. Such information could be used to classify those 92 binaries as malicious resulting in a detection rate of over 98%, potentially extending the coverage to polymorphic and 0-day malware that have evaded the antivirus engines. Such sharing of information between detection engines illustrates one interesting advantage of combining heterogeneous detection systems into one service.

4.2.2 Performance

By moving analysis into the network, we incur the network latency of sending the binary to the service and the latency of using a multiple engines to analyze the binary. In this subsection, we show that this latency is small on average and is acceptable for clients connected to the high-speed, low-latency ($<$100ms) local networks common to organizations. We also show that the caching mechanism can actually increase the performance in many cases.

To evaluate analysis time, we collected a set of legitimate executables from a default install of Microsoft Windows XP. This set was about 84 MB and consisted of 472 executables with an average size of 183 KB. We processed them with the prototype network service and the average analysis time per executable for the antivirus engines was 0.05 seconds, with ClamAV being the slowest with an average time of 0.14 seconds. We did not evaluate the analysis times for the behavioral engines as the runtime is a configurable parameter. Thus, for this legitimate dataset the average analysis time and network latency on a local network is well under a second.

We also looked at the analysis times for the 5066 mal-

ware samples. The average size of the samples was 366 KB and the average analysis time per executable for the antivirus engines was 0.48 seconds. Symantec was the slowest taking 0.91 seconds per executable on average. Again, the analysis times and executable sizes indicate perceived execution latencies of a second or two.

Thus far we have assumed that the executable under analysis has never been observed before. Our prototype includes both a local and remote cache of threat reports of previously analyzed executables to significantly speed up execution of previously analyzed binaries. While we do not have data in this paper on the frequency of legitimate executables used in an organization, it would be logical that many employees within an organization use a similar set of applications making caching an important performance enhancement. We were able to obtain data from the mwcollect Alliance [9] on the frequency of malicious executables. Over a two month period on a /18 network they observed 213 distinct executables over 2.5 million times. Of the 213 executables, 164 of them (77%) were seen multiple times i.e., 49 were only observed once. Given the vast number of duplicate encountered, a blacklist cache for these 213 executables would have a cache hit rate of over 99.99%, resulting in a near-instant response from the service.

5 DISCUSSION

This paper has presented a vision for moving from a host-centric antivirus paradigm to providing executable analysis as an in-cloud network service. We constructed a prototype and our initial results show potential for a significant improvement in the detection of real malicious software. However, there are several areas that require further investigation.

First and foremost is the question of disconnected operation. When a mobile user is not connected to the network, has a high-latency, low-bandwidth connection, or is the victim of a denial of service attack, sending executables to the network service may not be feasible. While certain organizations may select a policy requiring that users wait for network connectivity before running new applications, others may desire more flexibility. One solution is to have mobile users also run traditional antivirus software as a fail-over backup. Therefore, in the disconnected state, users will have at least the same detection coverage as today.

Another issue is that malicious software comes in many forms. While our prototype currently operates only on Win32 Portable Executables, we are working on extending it to handle external DLLs, interpreted scripting languages, malicious web content, and other file types. Although our system cannot detect shellcode injected into memory, such shellcode often fetches an executable which can be detected.

Finally, since our system integrates other detection engines, it also shares their limitations. For example, a behavioral profiler might not detect malware that delays exhibiting behavior. However, since the architecture is modular, improved detection engines are easily integrated into the system.

We are currently working to refine our prototype and are collaborating with different enterprises and service providers to collect information on executable usage and further evaluate our approach.

ACKNOWLEDGMENTS

This work was supported in part by the Department of Homeland Security (DHS) under contract number NBCHC060090 and by the National Science Foundation (NSF) under contract number CNS 0627445. We would like to thank Jose Nazario from Arbor Networks and Georg Wicherski from the mwcollect Alliance.

REFERENCES

[1] Michael Bailey, Jon Oberheide, Jon Andersen, Z. Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. To appear in *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)*.

[2] Barracuda Networks. Barracuda spam firewall. `http://www.barracudanetworks.com`, 2007.

[3] Carsten Willems and Thorsten Holz. Cwsandbox. `http://www.cwsandbox.org/`, 2007.

[4] Cloudmark. Cloudmark authority anti-virus. `http://www.cloudmark.com`, 2007.

[5] Abhishek Kumar, Vern Paxson, and Nicholas Weaver. Exploiting underlying structure for detailed reconstruction of an internet-scale event. *Proceedings of the USENIX/ACM Internet Measurement Conference*, October 2005.

[6] Microsoft. Microsoft security intelligence report: July-december 2006. `http://www.microsoft.com/technet/security/default.mspx`, May 2007.

[7] Arbor Networks. Arbor malware library (AML). `http://www.arbornetworks.com`, 2007.

[8] Norman Solutions. Norman sandbox whitepaper. `http://download.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf`, 2003.

[9] Paul Bacher and Markus Kotter and Georg Wicherski. The mwcollect alliance. `http://www.mwcollect.org`, 2007.

[10] Honeynet Project. Know Your Enemy: Learning with VMware. 2003.

[11] Niels Provos. Spybye. `http://www.monkey.org/~provos/spybye`, 2007.

[12] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *The 22th Annual Computer Security Applications Conference (ACSAC 2006)*, Miami Beach, FL, December 2006.

[13] Symantec Corporation. Symantec security advisory (sym06-010). `http://www.symantec.com/avcenter/security/Content/2006.05.25.html`, 2006.

[14] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity and containment in the Potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October 2005.