Antivirus in the Network Cloud

Xu Da-wei
School of Electronic and Information Engineering
Changchun University
Changchun, China
e-mail: 24003908@qq.com

Yu Cun-jiang
School of Electronic and Information Engineering
Changchun University
Changchun, China
e-mail: 358219140@qq.com

Abstract—**In this paper a new model for malware detection on end hosts based on providing antivirus as an in-cloud network service. This model enables identification of malicious and unwanted software by multiple, heterogeneous detection engines in parallel, a technique we term 'N-version protection'. This approach provides several important benefits including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deployability and management. To explore this idea we construct and deploy a production quality in-cloud antivirus system called CloudAV.CloudAV includes a lightweight, cross-platform host agent and a network service with ten antivirus engines and two behavioral detection engines.**

Keywords-antivirus;network cloud; cloudAV

I. INTRODUCTION

Detecting malicious software is a complex problem. The vast, ever-increasing ecosystem of malicious software and tools presents a daunting challenge for network operators and IT administrators. Antivirus software is one of the most widely used tools for detecting and stopping malicious and unwanted software. However, the elevating sophistication of modern malicious software means that it is increasingly challenging for any single vendor to develop signatures for every new threat [1].

In this paper we suggest a new model for the detection functionality currently performed by host-based antivirus software. To explore and validate this new antivirus model, we propose an in-cloud antivirus architecture that consists of three major components: a lightweight host agent run on end hosts like desktops, laptops, and mobiles devices that identifies new files and sends them into the network for analysis; a network service that receives files from hosts and identifies malicious or unwanted content; and an archival and forensics service that stores information about analyzed files and provides a management interface for operators. We construct, deploy, and evaluate a production quality in-cloud antivirus system called CloudAV. CloudAV includes a lightweight, cross-platform host agent for Windows, Linux, and FreeBSD and a network service consisting of ten antivirus engines and two behavioral detection engines.

II. APPROACH

This paper advocates a new model for the detection functionality currently performed by antivirus software. First, the detection capabilities currently provided by host-based antivirus software can be more efficiently and effectively provided as an in-cloud network service. Second, the identification of malicious and unwanted software should be determined by multiple, heterogeneous detection engines in parallel.

A. Deployment Environment

Before getting into details of the approach, it is important to understand the environment in which such architecture is most effective. First and foremost, we do not see the architecture replacing existing antivirus or intrusion detection solutions. We base our approach on the same threat model as existing host-based antivirus solutions and assume an in-cloud antivirus service would run as an additional layer of protection to augment existing security systems such as those inside an organizational network like an enterprise. Some possible deployment environments include: enterprise networks and government networks and mobile/cellular networks and privacy implications.

B. In-Cloud Detection

The core of the proposed approach is moving the detection of malicious and unwanted files from end hosts and into the network. This idea was originally introduced in and we significantly extend and evaluate the concept in this paper.

There is currently a strong trend toward moving services from end host and monolithic servers into the network cloud [2]. Moving the detection of malicious and unwanted files into the network significantly lowers the complexity of host-based monitoring software. Clients no longer need to continually update their local signature database, reducing administrative cost. Simplifying the host software also decreases the chance that it could contain exploitable vulnerabilities. Finally, a lightweight host agent allows the service to be extended to mobile and resource-limited devices that lack sufficient processing power but remain an enticing target for malware.

C. N-Version Protection

The second core component of the proposed approach is a set of heterogeneous detection engines that are used to provide analysis results on a file, also known as N-version protection. This approach is very similar to N-version programming, a paradigm in which multiple implementations of critical software are written by independent parties to increase the reliability of software by reducing the probability of concurrent failures. Traditionally, N-version programming has been applied to systems requiring high availability such as distributed file systems. N-version programming has also been applied to security realm to detect implementation faults in web services that

may be exploited by an attacker. While N-version programming uses multiple implementations to increase fault tolerance in complex software, the proposed approach uses multiple independent implementations of detection engines to increase coverage against a highly complex and ever-evolving ecosystem of malicious software.

III. ARCHITECTURE

In order to move the detection of malicious and unwanted files from end hosts and into the network, several important challenges must be overcome [3]: (1) unlike existing antivirus software, files must transported into the network for analysis; (2) an efficient analysis system must be constructed to handle the analysis of files from many different hosts using many different detection engines in parallel; (3) the performance of the system must be similar or better than existing detection systems such as antivirus software.

To address these problems we envision an architecture that includes three major components. The first is a lightweight host agent run on end systems like desktops, laptops, and mobiles devices that identifies new files and sends them into the network for analysis. The second is a network service that receives files from the host agent, identifies malicious and unwanted content, and instructs hosts whether access to the files is safe. The third component is an archival and forensics service that stores information about what files were analyzed and provides a query and alerting interface for operators. Figure 1 shows the high level architecture of the approach.

A. *Client Software*

Malicious and unwanted files can enter an organization from many sources. For example, mobile devices, an USB drive, email attachments, downloads, and vulnerable network services are all common entry points. Due to the broad range of entry vectors, the proposed architecture uses a lightweight file acquisition agent run on each end system. Just like existing antivirus software, the host agent runs on each end host and inspects each file on the system. Access to each file is trapped and diverted to a handling routine which begins by generating a unique identifier (UID) of the file and comparing that identifier against a cache of previously analyzed files. If a file UID is not present in the cache then the file is sent to the in-cloud network service for analysis. To make the analysis process more efficient, the architecture provides a method for sending a file for analysis as soon as it is written on the end host's file system. Doing so amortizes the transmission and analysis cost over the time elapsed between file creation and system or user-initiated access [4].

(1) Threat Model

The threat model for the host agent is similar to that of existing software protection mechanisms such as antivirus, host-based firewalls, and host-based intrusion detection. As with these host-based systems, if an attacker has already achieved code execution privileges, it may be possible to evade or disable the host agent. As described in Section 2, antivirus software contains much vulnerability that can be directly targeted by malware due to its complexity. By reducing the complexity of the host agent by moving detection into the network, it is possible to reduce the vulnerability footprint of host software that may lead to elevated privileges or code execution [5].

(2) File Unique Identifiers

One of the core components of the host agent is the file unique identifier (UID) generator. The goal of the UID generator is to provide a compact summary of a file. That summary is transmitted over the network to determine if an identical file has already been analyzed by the network service. One of the simplest methods of generating such a UID is a cryptographic hash of a file, such as MD5 or SHA-1. Cryptographic hashes are fast and provide excellent resistance to collision attacks. However, the same collision resistance also means that changing a single byte in a file results in completely different UID.To combat polymorphic threats, a more complex UID generator algorithm could be employed.

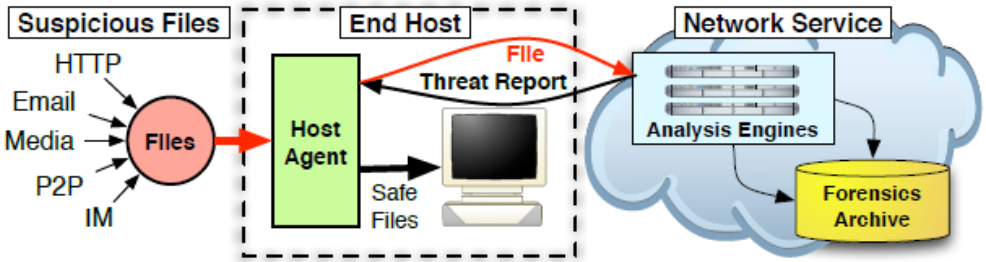

Figure 1: Architectural approach for in-cloud file analysis service

(3) User Interface

We envision three majors modes of operation that affect how users interact with the host agent that range from less to more interactive. These include three modes: transparent mode and warning mode and blocking mode.

B. *Network Service*

The second major component of the architecture is the network service responsible for file analysis. The core task of the network service is to determine whether a file is malicious or unwanted. Unlike existing systems, each file is analyzed by a collection of detection engines. That is, each file is analyzed by multiple detection engines in parallel and a final determination of whether a file is malicious or unwanted is made by aggregating these individual results into a threat report [6].

(1) Detection Engines

A cluster of servers can quickly analyze files using multiple detection techniques. 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. In addition, the use of engines from different vendors using different detection techniques means that the overall result does not rely too heavily on a single vendor or detection technology.

(2) Result Aggregation

The results from the different detection engines must be combined to determine whether a file is safe to open, access, or execute. The result of the aggregation process is a threat report that is sent to the host agent and can be cached on the server. A threat report can contain a variety of metadata and analysis results about a file. The specific contents of the report depend on the deployment scenario. Some possible report sections include: (1) an operation directive; a set of instructions indicating the action to be performed by the host agent, such as how the file should be accessed, opened, executed, or quarantined; (2) family/variant labels; a list of malware family/variant classification labels assigned to the file by the different detection engines; and (3) 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, or other state changes.

C. *Archival and Forensics Service*

The third and final component of the architecture is a service that provides information on file usage across participating hosts which can assist in post-infection forensic analysis. While some forensics tracking systems provide fine-grained details tracing back to the exact vulnerable processes and system objects involved in an infection, they are often accompanied by high storage requirements and performance degradation. Instead, we opt for a lightweight solution consisting of file access information sent by the host agent and stored securely by the network service, in addition to the behavioral profiles of malicious software generated by the behavioral detection engines. Depending on the privacy policy of organization, a tunable amount of forensics information can be logged and sent to the archival service. For example, a more security conscious organization could specify that information about every executable launch be recorded and sent to the archival service. Another policy might specify that only accesses to unsafe files be archived without any personally identifiable information.

Archiving forensic and file usage information provides a rich information source for both security professionals and administrators. From a security perspective, tracking the system events leading up to an infection can assist in determining its cause, assessing the risk involved with the compromise, and aiding in any necessary disinfection and cleanup. In addition, threat reports from behavior slanginess provide a valuable source of forensic data as the exact operations performed by a piece of malicious software can be analyzed in detail. From a general administration perspective, knowledge of what applications and files are frequently in use can aid the placement of file caches, application servers, and even be used to determine the optimal number of licenses needed for expensive applications.

Consider the outbreak of a zero-day exploit. An enterprise might receive a notice of a new malware attack and wonder how many of their systems were infected. In the past, this might require performing an inventory of all systems, determining which were running vulnerable software, and then manually inspecting each system. Using the forensics archival interface in the proposed architecture, an operator could search for the UID of the malicious file over the past few months and instantly find out where, when, and who opened the file and what malicious actions the file performed. The impacted machines could then immediately be quarantined. The forensics archive also enables retrospective detection. The complete archive of files that are transmitted to the network service may be re-scanned by available engines whenever a signature update occurs. Retrospective detection allows previously undetected malware that has infected a host to be identified and quarantined.

IV. CLOUDAV IMPLEMENTATION

To explore and validate the proposed in-cloud antivirus architecture, we constructed a production quality implementation called CloudAV. In this section we describe how CloudAV implements each of the three main components of the architecture.

A. *Host Agent*

We implement the host agent for a variety of platforms including Windows 2000/XP/Vista, Linux 2.4/2.6, and FreeBSD 6.0+. The implementation of the host agent is designed to acquire executable files for analysis by the in-cloud network service, as executables are a common source of malicious content.

While the exact APIs are platform dependent (CreateProcess on Win32, execve syscall on Linux 2.4, LSM hooks on Linux 2.6, etc), the host agent hooks and interposes on system events. This interposition is implemented via the MadCodeHook package on the Win32 platform and via the Dazuko framework for the other platforms. Process creation events are interposed upon by the host agent to acquire and process candidate executables before they are allowed to continue.

In addition, filesystem events are captured to identify new files entering a host and preemptively transfer them to the network service before execution to eliminate any user-perceived latencies. As motivating factors of our work include the complexity and security risks involved in running host-based antivirus, the host agent was designed to be simple and lightweight, both in code size and resource requirements. The Win32 agent is approximately 1500 lines of code of which 60% is managed code, further reducing the vulnerability profile of the agent. The agent for the other platforms is written in python and is under 300 lines of code.

While the host agent is primarily targeted at end hosts, our architecture is also effective in other deployment scenarios such as mail servers. To demonstrate this, we also implemented a milter (mail filter) frontend for use with mail transfer agents (MTAs) such as Send mail and Postfix to scan all attachments on incoming emails. Using the pymilter API, the milter frontend weighs in at approximately 100 lines of code.

B. Network Service

The network service acts as a dispatch manager between the host agent and the backend analysis engines. Incoming candidate files are received, analyzed, and a threat report is returned to the host agent dictating the appropriate action to take. Communication between the host agent and the network service uses a HTTP wire protocol protected by mutually authenticated SSL/TLS. Between the components within the network service itself, communication is performed via a publish/subscribe bus to allow modularization and effective scalability.

The network service allows for various priorities to be assigned to analysis requests to aid latency-sensitive applications and penalize misbehaving hosts. This also enables the system to penalize or temporarily suspend misbehaving hosts than may try to submit many analysis requests or otherwise flood the system.

Each backend engine runs in a Xen virtualized container, which offers significant advantages in terms of isolation and scalability. If one of the antivirus engines in the backend is targeted and successfully exploited by a malicious candidate file, the virtualized container can simply be disposed of and immediately reverted to a clean snapshot. As for scalability, virtualized containers allows the network service to spin up multiple instances of a particular engine when demand for its services increase.

Our current implementation employs 12 engines: 10 traditional antivirus engines (Avast, AVG, BitDefender, ClamAV, F-Prot, F-Secure, Kaspersky, McAfee, Symantec, and Trend Micro) and 2 behavioral engines (Norman Sandbox and CWSandbox). 9 of the backend engines run in a Windows XP environment using Xen's HVM capabilities while the other 3 run in a Gentoo Linux environment using Xen domU paravirtualization. Implementing each particular engine for the backend is a simple task and extending the backend with additional engines in the future is equally as simple. For reference, the amount of code required for each engine is 42 lines of python code on average with a median of 26 lines of code.

C. Management Interface

The third component is a management interface which provides access to the forensics archive, policy enforcement, alerting, and report generation. These interfaces are exposed to network administrators via a web based management interface. The web interface is implemented using Cherrypy, a python web development framework. The centralized management and network-based architecture allows for administrators to enforce network wide policies and define alerts when those policies are violated. Alerts are defined through a flexible specification language consisting of attributes describing an access request from the host agent and boolean predicates similar to an SQL WHERE clause. The specification language allows for notification for triggered alerts (via email, syslog, SNMP) and enforcement of administrator defined policies. While these unwanted applications may not be explicitly malicious, they may have a negative effect on host or network performance or be against acceptable use policies. We observed several classes of these potentially unwanted applications in our production deployment including P2P applications (uTorrent, Limewire, etc) and multi-player gaming (World of Warcraft, online poker, etc). Other policies can be defined to reinforce prudent security practices, such as blocking the user from executing attachments from an email application.

V. CONCLUSION

To address the ever-growing sophistication and threat of modern malicious software, we have proposed a new model for antivirus deployment by providing antivirus functionality as a network service using N-version protection. This novel paradigm provides significant advantages over traditional host-based antivirus including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deploy ability and management. In the future, we plan to investigate the application of N-version protection to intrusion detection, phasing, and other realms of security that may benefit from heterogeneity. We also plan to open our backend analysis infrastructure to security researchers to aid in the detection and classification of collected malware samples.

REFERENCES

[1] Adobe Systems Incorporated.Apsb07-18:Adobe reader and acrobat vulnerability.http://www.adobe.com/support/security/bulletins/apsb07-18.html, 2007.

[2] Algirdas Avizienis. The n-version approach to fault-tolerant software. IEEE Transactions on Software Engineering, 1985.

[3] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In 9th International Symposium On Recent Advances In Intrusion Detection. Springer-Verlag, 2006.

[4] Josh Ballard. An Eye on the Storm: Inside the Storm Epidemic.41st Meeting of the North Americian Network Operators Group,October 2007.

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

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