CloudAV: N-Version Antivirus 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 is one of the most widely used tools for detecting and stopping malicious and unwanted files. However, the long term effectiveness of traditional host-based antivirus is questionable. Antivirus software fails to detect many modern threats and its increasing complexity has resulted in vulnerabilities that are being exploited by malware. This paper advocates 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. We evaluate the performance, scalability, and efficacy of the system using data from a real-world deployment lasting more than six months and a database of 7220 malware samples covering a one year period. Using this dataset we find that CloudAV provides 35% better detection coverage against recent threats compared to a single antivirus engine and a 98% detection rate across the full dataset. We show that the average length of time to detect new threats by an antivirus engine is 48 days and that retrospective detection can greatly minimize the impact of this delay. Finally, we relate two case studies demonstrating how the forensics capabilities of CloudAV were used by operators during the deployment.

1 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. Indeed, a recent Microsoft survey found more than 45,000 new variants of backdoors, trojans, and bots during the second half of 2006 [17].

Two important trends call into question the long term effectiveness of products from a single antivirus vendor. First, there is a significant vulnerability window between when a threat first appears and when antivirus vendors generate a signature. Moreover, a substantial percentage of malware is never detected by antivirus software. This means that end systems with the latest antivirus software and signatures can still be vulnerable for long periods of time. The second important trend is that the increasing complexity of antivirus software and services has indirectly resulted in vulnerabilities that can and are being exploited by malware. That is, malware is actually using vulnerabilities in antivirus software itself as a means to infect systems. SANS has listed vulnerabilities in antivirus software as one of the top 20 threats of 2007 [27].

In this paper we suggest a new model for the detection functionality currently performed by host-based antivirus software. This shift is characterized by two key changes.

1. **Antivirus as a network service:** First, the detection capabilities 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 detect new files, send them to a network service for analysis, and then permit access or quarantine them based on a report returned

by the network service.

2. **N-version protection:** Second, the identification of malicious and unwanted software should be determined by multiple, heterogeneous detection engines in parallel. Similar to the idea of N-version programming, we propose the notion of N-version protection and suggest that malware detection systems should leverage the detection capabilities of multiple, heterogeneous detection engines to more effectively determine malicious and unwanted files.

This new model provides several important benefits. (1) *Better detection of malicious software*: antivirus engines have complementary detection capabilities and a combination of many different engines can improve the overall identification of malicious and unwanted software. (2) *Enhanced forensics capabilities*: information about what hosts accessed what files provides an incredibly rich database of information for forensics and intrusion analysis. Such information provides temporal relationships between file access events on the same or different hosts. (3) *Retrospective detection*: when a new threat is identified, historical information can be used to identify exactly which hosts or users open similar or identical files. For example, if a new botnet is detected, an in-cloud antivirus service can use the execution history of hosts on a network to identify which hosts have been infected and notify administrators or even automatically quarantine infected hosts. (4) *Improved deployability and management*: Moving detection off the host and into the network significantly simplifies host software enabling deployment on a wider range of platforms and enabling administrators to centrally control signatures and enforce file access policies.

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. We provide a detailed evaluation of the system using a dataset of 7220 malware samples collected in the wild over a period of a year [20] and a production deployment of our system on a campus network in computer labs spanning multiple departments for a period of over 6 months.

Using the malware dataset, we show how the CloudAV N-version protection approach provides 35% better detection coverage against recent threats compared to a single antivirus engine and 98% detection coverage of the entire dataset compared to 83% with a single engine. In addition, we empirically find that the average length of time to detect new threats by a single engine is 48 days and show how retrospective detection can greatly minimize the impact of this delay.

Finally, we analyze the performance and scalability of the system using deployment results and show that while the total number of executables run by all the systems in a computing lab is quite large (an average of 20,500 per day), the number of unique executables run per day is two orders of magnitude smaller (an average of 217 per day). This means that the caching mechanisms employed in the network service achieves a hit rate of over 99.8%, reducing the load on the network and, in the rare case of a cache miss, we show that the average time required to analyze a file using CloudAV's detection engines is approximately 1.3 seconds.

2 Limitations of Antivirus Software

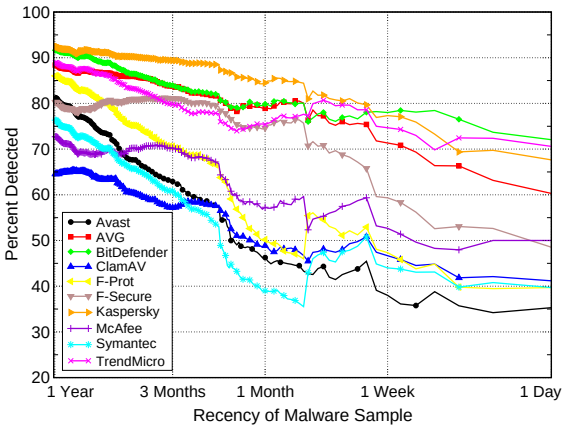
Antivirus software is one of the most successful and widely used tools for detecting and stopping malicious and unwanted software. Antivirus software is deployed on most desktops and workstations in enterprises across the world. The market for antivirus and other security software is estimated to increase to over $10 billion dollars in 2008 [10].

The ubiquitous deployment of antivirus software is closely tied to the ever-expanding ecosystem of malicious software and tools. As the construction of malicious software has shifted from the work of novices to a commercial and financially lucrative enterprise, antivirus vendors must expend more resources to keep up. The rise of botnets and targeted malware attacks for the purposes of spam, fraud, and identity theft present an evolving challenge for antivirus companies. For example, the recent Storm worm demonstrated the use of encrypted peer-to-peer command and control, and the rapid deployment of new variants to continually evade the signatures of antivirus software [4].

However, two important trends call into question the long term effectiveness of products from a single antivirus vendor. The first is that antivirus software fails to detect a significant percentage of malware in the wild. Moreover, there is a significant vulnerability window between when a threat first appears and when antivirus vendors generate a signature or modify their software to detect the threat. This means that end systems with the

AV Vendor	Version	3 Months	1 Month	1 Week
Avast	4.7.1043	62.7%	45.8%	39.6%
AVG	7.5.503	83.8%	78.6%	72.2%
BitDefender	7.1.2559	83.9%	79.7%	78.5%
ClamAV	0.91.2	57.5%	48.8%	46.8%
CWSandbox	2.0	N/A	N/A	N/A
F-Prot	6.0.8.0	70.4%	49.6%	46.0%
F-Secure	8.00.101	80.9%	74.4%	60.3%
Kaspersky	7.0.0.125	89.2%	84.0%	78.5%
McAfee	8.5.0i	70.5%	56.7%	53.9%
Norman	1.8	N/A	N/A	N/A
Symantec	15.0.0.58	60.8%	38.8%	45.2%
Trend Micro	16.00	79.4%	74.6%	75.3%

(a)



(b)

Figure 1: Detection rate for ten popular antivirus products as a function of the age of the malware samples.

latest antivirus software and signatures can still be vulnerable for long periods of time. The second important trend is that the increasing complexity of antivirus software and services has indirectly resulted in vulnerabilities that can and are being exploited by malware. That is, malware is actually using vulnerabilities in antivirus software as means to infect systems.

2.1 Vulnerability Window

The sheer volume of new threats means that it is difficult for any one antivirus vendor to create signatures for all new threats. The ability of any single vendor to create signatures is dependent on many factors such as detection algorithms, collection methodology of malware samples, and response time to 0-day malware. The end result is that there is a significant period of time between when a threat appears and when a signature is created by antivirus vendors (the *vulnerability window*).

To quantify the vulnerability window, we analyzed the detection rate of multiple antivirus engines across malware samples collected over a one year period. The dataset included 7220 samples that were collected between November 11th, 2006 to November 10th, 2007. The malware dataset is described in further detail in Section 6. The signatures used for the antivirus were updated the day after collection ended, November 11th, 2007, and stayed constant through the analysis.

In the first experiment, we analyzed the detection of recent malware. We created three groups of malware: one that included malware collected more recently than 3 months ago, one that included malware collected more recently than 1 month ago, and one that included malware collected more recently than 1 week ago. The antivirus engine and signature versions along with their associated detection rates for each time period are listed in Figure 1(a). The table clearly shows that the detection rate decreases as the malware becomes more recent. Specifically, the number of malware samples detected in the 1 week time period, arguably the most recent and important threats, is quite low.

In the second experiment, we extended this analysis across all the days in the year over which the malware samples were collected. Figure 1(b) shows significant degradation of antivirus engine detection rates as the age, or recency, of the malware sample is varied. As can be seen in the figure, detection rates can drop over 45% when one day's worth of malware is compared to a year's worth. As the plot shows, antivirus engines tend to be effective against malware that is a year old but much less useful in detecting more recent malware, which pose the greatest threat to end hosts.

2.2 Antivirus Software Vulnerabilities

A second major concern about the long term viability of host-based antivirus software is that the complexity of antivirus software has resulted in an increased risk of security vulnerabilities. Indeed, severe vulnerabilities have been discovered in the antivirus engines of nearly every vendor. While local exploits are more common (`ioctl` vulnerabilities, overflows in decompression routines, etc), remote exploits in management interfaces have been observed in the wild [30]. Due to the inherent need for elevated privileges by antivirus software, many of these vulnerabilities result in a complete compromise of the affected end host.

Figure 2 shows the number of vulnerabilities reported in the National Vulnerability Database [21] for ten popular antivirus vendors between 2005 and November 2007. This large number of reported vulnerabilities demonstrates not only the risk involved in deploying antivirus

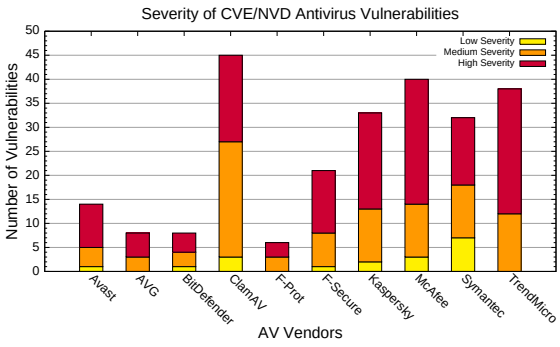


Figure 2: Number of vulnerabilities reported in the National Vulnerability Database (NVD) for ten antivirus vendors between 2005 and November 2007

software, but also an evolution in tactics as attackers are now targeting vulnerabilities in antivirus software itself.

3 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.

3.1 Deployment Environment

Before getting into details of the approach, it is important to understand the environment in which such an 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:** Enterprise networks tend to be highly controlled environments in which IT administrators control both desktop and server software. In addition, enterprises typically have good network connectivity with low latencies and high bandwidth between workstations and back office systems.

- **Government networks:** Like enterprise networks, government networks tend to be highly controlled with strictly enforced software and security practices. In addition, policy enforcement, access control, and forensic logging can be useful in tracking sensitive information.

- **Mobile/Cellular networks:** The rise of ubiquitous WiFi and mobile 2.5G and 3G data networks also provide an excellent platform for a provider-managed antivirus solution. As mobile devices become increasingly complex, there is an increasing need for mobile security software. Antivirus software has recently become available from multiple vendors for mobile phones [9, 13, 31].

Privacy implications: Shifting file analysis to a central location provides significant benefits but also has important privacy implications. It is critical that users of an in-cloud antivirus solution understand that their files may be transferred to another computer for analysis. There are may be situations where this might not be acceptable to users (e.g. many law firms and many consumer broadband customers). However, in controlled environments with explicit network access policies, like many enterprises, such issues are less of a concern. Moreover, the amount of information that is collected can be carefully controlled depending on the environment. As we will discuss later, information about each file analyzed and what files are cached can be controlled depending on the policies of the network.

3.2 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 [23] 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. For example, in-cloud email [5, 7, 28] and HTTP [18, 25] filtering systems are already popular and are used to provide an additional layer of security for enterprise networks. In addition, there have been several attempts to provide network services as overlay networks [29, 33].

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 [15, 30]. 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.

3.3 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 [2]. Traditionally, N-version programming has been applied to systems requiring high availability such as distributed filesystems [26]. N-version programming has also been applied to security realm to detect implementation faults in web services that may be exploited by an attacker [19]. 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.

A few online services have recently been constructed that implement N-version detection techniques. For example, there are online web services for malware submission and analysis [6, 11, 22]. However, these services are designed for the occasional manual upload of a virus sample, rather than the automated and real-time protection of end hosts.

4 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: (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; and (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 3 shows the high level architecture of the approach.

4.1 Client Software

Malicious and unwanted files can enter an organization from many sources. For example, mobile devices, USB drives, 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 filesystem (e.g., via file-copy, installation, or download). Doing so amortizes the transmission and analysis cost over the time elapsed between file creation and system or user-initiated access.

4.1.1 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 many vulnerabilities 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.

4.1.2 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. For example,

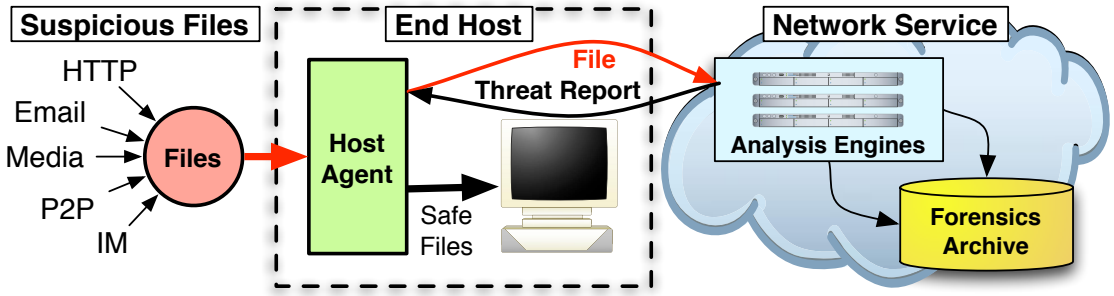


Figure 3: Architectural approach for in-cloud file analysis service.

a method such as locality-preserving hashing in multi-dimensional spaces [12] could be used track differences between two files in a compact manner.

4.1.3 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.

- **Transparent mode:** In this mode, the detection software is completely transparent to the end user. Files are sent into the cloud for analysis but the execution or loading of a file is never blocked or interrupted. In this mode end hosts can become infected by known malware but administrators can use detection alerts and detailed forensic information to aid in cleaning up infected systems.
- **Warning mode:** In this mode, access to a file is blocked until an *access directive* has been returned to the host agent. If the file is classified as unsafe then a warning is presented to the user instructing them why the file is suspicious. The user is then allowed to make the decision of whether to proceed in accessing the file or not.
- **Blocking mode:** In this mode, access to a file is blocked until an *access directive* has been returned to the host agent. If the file is classified as suspicious then access to the file is denied and the user is informed with an error dialog.

4.1.4 Other File Acquisition Methods

While the host agent is the primary method of acquiring candidate files and transmitting them to the network service for analysis, other methods can also be employed to increase the performance and visibility of the system. For example, a network sensor or tap monitoring the traffic of a network may pull files directly out of a network stream using deep packet inspection (DPI) techniques. By identifying files and performing analysis before the file even reaches the destination host, the need to retransmit the file to the network service is alleviated and user-perceived latencies can be reduced. Clearly this approach cannot completely replace the host agent as network traffic can be encrypted, files may be encapsulated in unknown protocols, and the network is only one source of malicious content.

4.2 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.

4.2.1 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.

A wide range of both lightweight and heavyweight detection techniques can be used in the backend. For example, lightweight detection systems like existing *antivirus engines* can be used to evaluate candidate files. In addition, more heavyweight detectors like *behavioral analyzers* can also be used. A behavioral system executes a suspicious file in a sandboxed environment (e.g., Norman

Sandbox [22], CWSandbox [6]) or virtual machine and records host state changes and network activity. Such deep analysis is difficult or impossible to accomplish on resource-constrained devices like mobile phones but is possible when detection is moved to dedicated servers. In addition, instead of forcing signature updates to every host, detection engines can be kept up-to-date with the latest vendor signatures at a central source.

Finally, running multiple detection engines within the same service provides the ability to correlation information between engines. For example, if a detector finds that the behavior of an unknown file is similar to that of an file previously classified as malicious by antivirus engines, the unknown file can be marked as suspicious [23].

4.2.2 Result Aggregation

The results from the different detection engines must be combined to determine whether a file is safe to open, access, or execute. Several variables may impact this process.

First, results from the detection engines may reach the aggregator at different times – if a detector fails, it may never return any results. In order to prevent a slow or failed detector from holding up a host, the aggregator can use a subset of results to determine if a file is safe. Determining the size of such a quorum depends on the deployment scenario and variables like the number of detection engines, security policies, and latency requirements.

Second, the metadata returned by each detector may be different so the detection results are wrapped in a container object that describes how the data should be interpreted. For example, behavioral analysis reports may not indicate whether a file is safe but can be attached to the final aggregation report to help users, operators, or external programs interpret the results.

Lastly, the threshold at which a candidate file is deemed unsafe or malicious may be defined by security policy of the network's administrators. For example, some administrators may opt for a strict policy where a single engine is sufficient to deem a file malicious while less security-conscious administrators may require multiple engines to agree to deem a file malicious. We discuss the balance between coverage and confidence further in Section 7.

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.

4.2.3 Caching

Once a threat report has been generated for a candidate file, it can be stored in both a local cache on the host agent and in a shared remote cache on the server. This means that once a file has been analyzed, subsequent accesses to that file by the user can be determined locally without requiring network access. Moreover, once a single host in a network has accessed a file and sent it to the network service for analysis, any subsequent access of the same file by other hosts in the network can leverage the existing threat report in the shared remote cache on the server. Cached reports stored in the network service may also periodically be *pushed* to the host agent to speed up future accesses and invalidated when deemed necessary.

4.3 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 [14, 8] 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 behavioral

engines 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.

5 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.

5.1 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. We discuss how the agent can be extended to acquire DLLs, documents, and other common malcode-bearing files types in Section 7.

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 [16] package on the Win32 platform and via the Dazuko [24] 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.

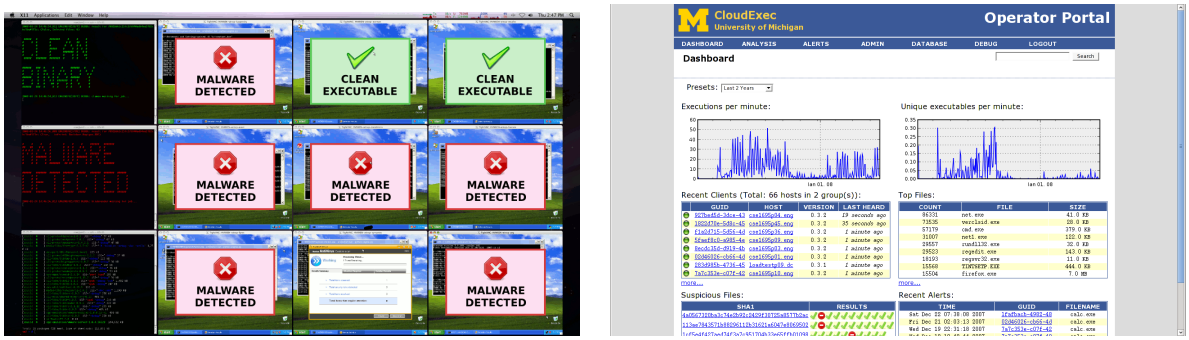
While the host agent is primarily targeted at end hosts, our architecture is also effective in other deployment scenarios such as mail servers. To demonstrate this, we also implemented a milter (mail filter) frontend for use with mail transfer agents (MTAs) such as Sendmail and Postfix to scan all attachments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

5.2 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. For example, application and mail scanning may take higher analysis priority than background analysis tasks such as retroactive detection (described in Section 7). 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. Given the numerous vulnerabilities in existing antivirus software discussed in Section 2, isolation of the antivirus engines from the rest of the system is vital. 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 partic-



(a) (b)

Figure 4: Screen captures of the detection engine VM monitoring interface (a) and the web management portal which provides access to forensic data and threat reports (b).

ular 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). The exact version of each detection engine is listed in Figure 1(a). 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.

5.3 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. A screen capture of the dashboard of the management interface is depicted in Figure 4.

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.

For example, network administrators may desire to block certain applications from being used on end hosts. 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.

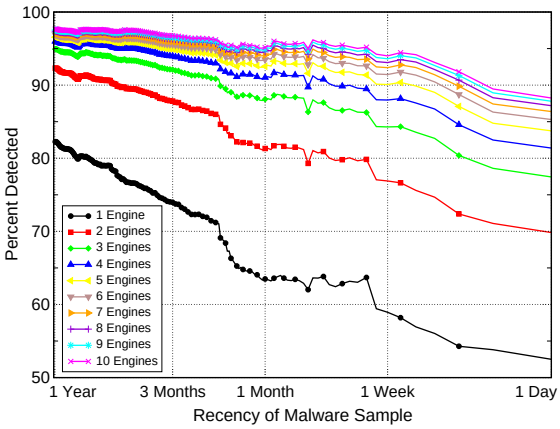
6 Evaluation

In this section, we provide an evaluation of the proposed architecture through two distinct sources of data. The first source is a dataset of malicious software collected over a period of a year. Using this dataset, we evaluate the effectiveness of N-version protection and retrospective detection. We also utilize this malware dataset to empirically quantify the size of vulnerability window.

The second data source is derived from a production deployment of the system on a campus network in computer labs spanning multiple departments for a period of over 6 months. We use the data collected from this deployment to explore the performance characteristics of CloudAV. For example, we analyze the number of files handled by the network service, the utility of the caching system, and the time it takes the detection engines to analyze individual files. In addition, we use deployment data to demonstrate the forensics capabilities of the approach. We detail two real-world case studies from the deployment, one involving an infection by malicious software and one involving a suspicious, yet legitimate executable.

Engines	3 Months	1 Month	1 Week
1	73.9%	63.1%	59.6%
2	87.7%	81.0%	77.6%
3	92.0%	87.8%	84.8%
4	93.8%	90.9%	88.4%
5	94.8%	92.4%	90.5%
6	95.4%	93.4%	91.8%
7	95.9%	94.0%	92.8%
8	96.2%	94.5%	93.5%
9	96.5%	94.8%	94.0%
10	96.7%	95.0%	94.4%

(a)



(b)

Figure 5: The average detection coverage for the various datasets (a) and the continuous coverage over time (b) when a given number of engines are used in parallel.

6.1 Malware Dataset Results

The first component of the evaluation is based on a malware dataset obtained through Arbor Network's Arbor Malware Library (AML) [20]. AML is composed of malware collected using a variety of techniques such as distributed darknet honeypots, spam traps, and honeyclient spidering. The use of a diverse set of collection techniques means that the malware samples are more representative of threats faced by end hosts than malware datasets collected using only a single collection methodology such as Nepenthes [3]. The AML dataset used in this paper consists of 7220 unique malware samples collected over a period of one year (November 12th, 2006 to November 11th, 2007). An average of 20 samples were collected each day with a standard deviation of 19.6 samples.

6.1.1 N-Version Protection

We used the AML malware dataset to assess the effectiveness of a set of heterogeneous detection engines. Figure 5(a) and (b) show the overall detection rate across different time ranges of malware samples as the number of detection engines is increased. The detection rates were determined by looking at the average performance across all combinations of N engines for a given N. For example, the average detection rate across all combinations of two detection engines over the most recent 3 months of malware was 87.7%.

Figure 5(a) demonstrates how the use of multiple heterogeneous engines allows CloudAV to significantly improve the aggregate detection rate. Figure 5(b) shows the detection rate over malware samples ranging from one day old to one year old. The graph shows how using ten engines can increase the detection rate for the entire year-long AML dataset as high as 98%.

The graph also reveals that CloudAV significantly improves the detection rate of more recent malware. When a single antivirus engine is used, the detection rate degrades from 82% against a year old dataset to 52% against a day old dataset (a decrease of 30%). However, using ten antivirus engines the detection coverage only goes from 98% down to 88% (a decrease of only 10%). These results show that not only do multiple engines complement each other to provide a higher detection rate, but the combination has resistance to coverage degradation as the encountered threats become more recent. As the most recent threats are typically the most important, a detection rate of 88% versus 52% is a significant advantage.

Another noticeable feature of Figure 5 is the decrease in incremental coverage. Moving from 1 to 2 engines results in a large jump in detection rate, moving from 2 to 3 is smaller, moving from 3 to 4 is even smaller, and so on. The diminishing marginal utility of additional engines shows that a practical balance may be reached between detection coverage and licensing costs, which we discuss further in Section 7.

In addition to the averages presented in Figure 5, the minimum and maximum detection coverage for a given number of engines is of interest. For the one week time range, the maximum detection coverage when using only a single engine is 78.6% (Kaspersky) and the minimum is 39.7% (Avast). When using 3 engines in parallel, the maximum detection coverage is 93.6% (BitDefender, Kaspersky, and Trend Micro) and the minimum is 69.1% (ClamAV, F-Prot, and McAfee). However, the optimal combination of antivirus vendors to achieve the most comprehensive protection against malware may not be

a simple measure of total detection coverage. Rather, a number of complex factors may influence the best choice of detection engines, including the types of threats most commonly faced by the hosts being protected, the algorithms used for detection by a particular vendor, the vendor's response time to 0-day malware, and the collection methodology and visibility employed by the vendor to collect new malware.

6.2 Retrospective Detection

We also used the AML malware dataset to understand the utility of retrospective detection. Recall that retrospective detection is the ability to use historical information and archived files stored by CloudAV to retrospectively detect and identify hosts infected that with malware that has previously gone undetected. Retrospective detection is an especially important post-infection defense against 0-day threats and is independent of the number or vendor of antivirus engines employed. Imagine a polymorphic threat not detected by any antivirus or behavioral engine that infects a few hosts on a network. In the host-based antivirus paradigm, those hosts could become infected, have their antivirus software disabled, and continue to be infected indefinitely.

In the proposed system, the infected file would be sent to the network service for analysis, deemed clean, archived at the network service, and the host would become infected. Then, when any of the antivirus vendors update their signature databases to detect the threat, the previously undetected malware can be re-scanned in the network service's archive and flagged as malicious. Instantly, armed with this new information, the network service can identify which hosts on the network have been infected in the past by this malware from its database of execution history and notify the administrators with detailed forensic information.

Retrospective detection is especially important as frequent signature updates from vendors continually add coverage for previously undetected malware. Using our AML dataset and an archive of a year's worth of McAfee DAT signature files (with a one week granularity), we determined that approximately 100 new malware samples were detected each week on average (with a standard deviation of 57) by the McAfee updates. More importantly, for those samples that were eventually detected by a signature update (5147 out of 7220), the average time from when a piece of malware was observed to when it was detected (i.e. the vulnerability window) was approximately 48 days. A cumulative distribution function of the days between observation and detection is depicted in Figure 6.

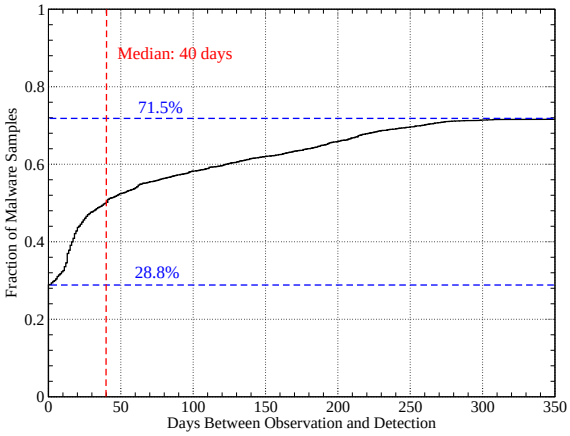


Figure 6: Cumulative distribution function depicting the number of days between when a malware sample is observed and when it is first detected by the McAfee antivirus engine.

6.3 Deployment Results

With the aid of network operations and security staff we deployed CloudAV across a large campus network. In this section, we discuss results based on the data collected as a part of this deployment.

6.3.1 Executable Events

One of the core variables that impacts the resource requirements of the network service is the rate at which new files must be analyzed. If this rate is extremely high, extensive computing resources will be required to handle the analysis load. Figure 7 shows the number of total execution events and unique executables observed during a one month period in a university computing lab.

Figure 7 shows that while the total number of executables run by all the systems in the lab is quite large (an average of 20,500 per day), the number of unique executables run per day is two orders of magnitude smaller (an average of 217 per day). Moreover, the number of unique executables is likely inflated due to the fact that these machines are frequently used by students to work on computer science class projects, resulting in a large number of distinct executables with each compile of a project. A more static, non-development environment would likely see even less unique executables.

We also investigated the origins of these executables based on the file path of 1000 unique executables stored in the forensics archive. Table 1 shows the break down of these sources. The majority of executables originate from the local hard drive but a significant portion were launched from various network sources. Executables from the temp directory often indicate that they were

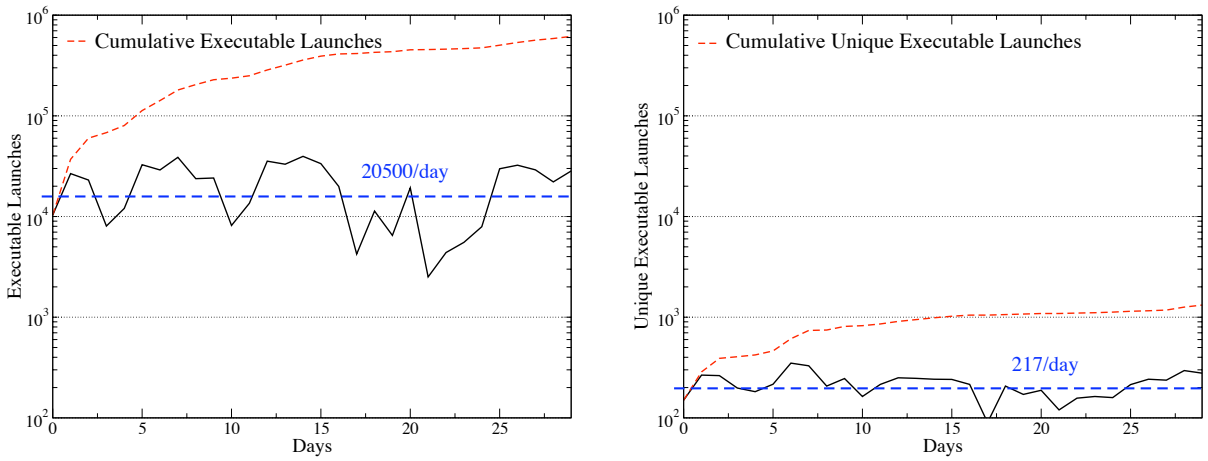


Figure 7: Executable launches (a) and unique executable launches (b) per day over a one month period in a representative sample of 50 machines in the deployment.

Local Drives 52.4%	Program Files	22.3%
	Temp Directory	14.2%
	Windows Directory	13.4%
	Other	2.4%
Network Drives 43.3%	Engineering Apps	23.6%
	User Desktop Shares	9.3%
	User AFS Shares	8.3%
	Other	2.1%
External Media 4.4%	USB Flash	2.4%
	CDROM Drive	2.0%

Table 1: A distribution of the sources of 1000 executables observed in during the deployment of our host agent over a six-month period.

downloaded via a web browser and executed, contributing even more to networked origins. In addition, a non-trivial number of executables were introduced to the system directly from external media such as a CDROM drive and USB flash media. This diversity exemplifies the need for a host agent that is capable of acquiring files from a variety of sources.

6.3.2 Caching and Performance

A second important variable that determines the scalability and performance of the system is the cache hit rate. A hit in the local cache can prevent network requests, and a hit in the remote cache can prevent unnecessary files transfers to the network service. The hosts instrumented as a part of the deployment were heavily loaded Windows XP workstations. The Windows Start Menu contained over 250 executable applications including a wide range of internet, multimedia, and engineering packages.

Our results indicate that 10 processes were launched from when the host agent service loads to when the login screen appears and another 52 processes were launched before the user's desktop loaded. As a measure of overhead, we measured the number of bytes transferred between a specific client and network service under different caching conditions. With a warm remote cache, the boot-up process took 8.7 KB and the login process took 46.2 KB. In the case of a cold remote cache, which would only ever occur a single time when the first host in the network loaded for the first time, the boot-up process took 406 KB and the login process took 12.5 MB. For comparison, the Active Directory service installed on the deployment machines took 171 KB and 270 KB on boot and login respectively.

It is also possible to evaluate the performance of the caching system by looking at Figure 7. We recorded almost over 615,000 total execution events over one month yet only observed 1300 unique executables. As a remote cache miss only happens when a new executable is observed, the remote cache hit rate is approximately 99.8%. Even more significant, the local cache can be pre-seeded with known installed software during the host agent installation process, improving the hit rate further. In the infrequent case when a miss occurs in both the local and remote cache, the candidate file must be transferred to the network service. Network latency, throughput, and analysis time all affect the user-perceived delay between when a file is acquired by the host agent and a threat report is returned by the network service. As local net-

works usually have low latencies and high bandwidth, the analysis time of files will often dominate the network latency and throughput delay. The average time for a detection engine to analyze a candidate file in the AML dataset was approximately 1.3 seconds with a standard deviation of 1.8 seconds.

6.3.3 Forensics Case Studies

We review two case studies from the deployment concerning two real-world events that demonstrate the utility of the forensics archive.

Malware Case Study: While running the host agent in transparent mode in the campus deployment, the CloudAV system alerted us to a candidate executable that had been marked as malicious by multiple antivirus engines. It is important to note that this malicious file successfully evaded the local antivirus software (McAfee) that was installed along side our host agent. Immediately, we accessed the management interface to view the forensics information associated with the tracked execution event and runtime behavioral results provided the two behavioral engines employed in our network service.

The initial executable launched by the user was `warcraft3keygen.exe`, an apparent serial number generator for the game Warcraft 3. This executable was just a bootstrap for the `m222.exe` executable which was written to the Windows temp directory and subsequently launched via CreateProcess. `m222.exe` then copied itself to `C:\Program Files\Intel\Intel`, made itself hidden and read-only, and created a fraudulent Windows service via the Service Control Manager (SCM) called `Remote Procedure Call (RPC) MO` to launch itself automatically at system startup. Additionally, the malware attempted to contact command and control infrastructure through DNS requests for several names including `50216.ipread.com`, but the domains had already been blackholed.

Legitimate Case Study: In another instance, we were alerted to a candidate executable that was flagged as suspicious by several engines. The executable in question was the PsExec utility from SysInternals which allows for remote control and command execution. Given that this utility can be used for both malicious and legitimate purposes, it was worthy of further investigation to determine its origin.

Using the management interface, we were able to immediately drill down to the affected host, user, files, and environment of the suspected event. The PsExec service `psexesvc.exe` was first launched from the parent process `services.exe` when an incoming remote execution request arrived from the PsExec client. The next execution event was `net.exe` with the command line argument `localgroup administrators`, which results in the listing of all the users in the local administrators group.

Three factors led us to dismiss the event as legitimate. First, the operation performed by the net command was not overtly malicious. Second, the user performing this action was a known network administrator. Lastly, we were able to determine the net.exe executable was identical to the one deployed across all the hosts in the network, ruling out the case where the net.exe program itself may have been a trojaned version. While this event could be seen as a false positive, it is actually an important alert that needs to dealt with by a network administrator. The forensic and historical information provided through the management interface allows these events to be dealt with remotely in an accurate and efficient manner.

7 Discussion and Limitations

Moving detection functionality into the network cloud has other technical and practical implications. In this section we attempt to highlight limitations of the proposed model and then describe a few resulting benefits.

7.1 User Context and Environment in Detection Engines

One important benefit of running detection engines on end systems is that local context such as user input, network input, operating system state, and the local filesystem are available to aid detection algorithms. For example, many antivirus vendors use behavioral detection routines that monitor running processes to identify misbehaving or potentially malicious programs.

While it is difficult to replicate the entire state of end systems inside the network cloud, there are two general techniques an in-cloud antivirus system can use to provide additional context to detection engines. First, detection engines can open or execute files inside a VM instance. For example, existing antivirus behavioral detection system can be leveraged by opening and running files inside a virtual antivirus detection instance. A second technique is to replicate more of the local end system state in the cloud. For example, when a file is sent to the network service, contextual metadata such as other running processes can be attached to the submission and used to aid detection. However, because complete local state can be quite large, there are many instances where deploying local detection agents may be required to compliment in-cloud detection.

7.2 Disconnected Operation

Another challenge with moving detection into the network is that network connectivity is needed to analyze files. An end host participating in the service may enter a disconnected state for many reasons including network outages, mobility constraints, misconfiguration, or denial of service attacks. In such a disconnected state, the host agent may not be able to reach the network service to check the remote cache or to submit new files for analysis. Therefore, in certain scenarios, the end host may be unable to complete its desired operations.

Addressing the issue of disconnected operation is primarily an issue of policy, although the architecture includes technical components that aid in continued protection in a disconnected state. For example, the local caching employed by our host agent effectively allows a disconnected user to access files that have previously been analyzed by the network service. However, for files that have not yet been analyzed, a policy decision is necessary. Security-conscious organizations may select a strict policy requiring that users have network connectivity before accessing new applications, while organizations with less strict security policies may desire more flexibility. As our host agent works together with host-based antivirus, local antivirus software installed on the end host may provide adequate protection for these environments with more liberal security policies until network access is restored.

7.3 Sources of Malicious Behavior

Malicious code or inputs that cause unwanted program behavior can be present in many places such as in the linking, loading, or running of the initial program instructions, and the reading of input from memory, the filesystem, or the network. For example, some types of malware use external files such as DLLs loaded at runtime to store and later execute malicious code. In addition, recent vulnerabilities in desktop software such as Adobe Acrobat [1] and Microsoft Word [32] have exemplified the threat from documents, multimedia, and other non-executable malcode-bearing file types. Developing a host agent that handles all these different sources of malicious behavior is challenging.

The CloudAV implementation described in this paper focuses on executables, but the host agent can be extended to identify other file types. To explore the challenges of extending the system we modified the host agent to monitor the DLL dependencies for each executable acquired by the host agent. Each dependent DLL of an application is processed similar to the executable itself: the local and remote cache is checked to determine if it has been previously analyzed, and if not, it is transmitted to the network service for analysis. Extending the host agent further to handle documents would be as simple as instructing the host agent to listen for filesystem events for the desired file types. In fact, the types of files acquired by the host agent could be dynamically configured at a central location by an administrator to adapt to evolving threats.

AV Vendor	3 Months	1 Month	1 Week
Avast	+14.8%	+16.6%	+24.6%
AVG	+5.9%	+6.8%	+8.7%
BitDefender	+4.0%	+5.3%	+3.1%
ClamAV	+0.0%	+0.0%	+0.0%
F-Prot	+9.9%	+15.3%	+12.6%
F-Secure	+7.9%	+9.3%	+15.0%
Kaspersky	+1.5%	+1.9%	+2.3%
McAfee	+10.6%	+14.0%	+14.2%
Symantec	+17.8%	+23.0%	+20.6%
Trend Micro	+9.8%	+11.5%	+12.6%

Table 2: The percentage increase in detection coverage obtained when ClamAV, a truly free engine, is added to a deployment with only a single engine.

7.4 Detection Engine Licensing

Most of the antivirus and behavioral engines employed in our architecture required paid licenses. Acquiring licenses for all the engines may be infeasible for some organizations. While we have chosen a large number of engines for evaluation and measurement purposes, the full amount may not be necessary to obtain effective protection. As seen in Figure 5, ten engines may not be the most effective price/performance point as diminishing returns are observed as more engines are added.

We currently employ four free engines in our system for which paid licenses were not necessary: AVG, Avast, BitDefender, and ClamAV. Using only these four engines, we are still able to obtain 94.3%, 92.0%, and 88.0% detection coverage over periods of 3 months, 1 month, and 1 week respectively. These detection coverage values for the combined free engines exceed every single vendor in each dataset period.

While the interpretation of the various antivirus licenses is unclear in our architecture, especially with regards to virtualization, it is likely that site-wide licenses would be needed for the "free" engines for a commercial deployment. Even if only one licensed engine is used, our system still maintains the benefits such as forensics and management. As an experiment for this scenario, we measured how much detection coverage would be gained by adding the only truly free (GPL licensed) antivirus

product, ClamAV, to an existing system employing only a single engine. Although ClamAV is not an especially effective engine by itself, it can add a significant amount of detection coverage, up to a 25% increase when paired with another engine as seen in Table 2.

7.5 Managing False Positives

The use of parallel detection engines has important implications for the management of false positives. While multiple detection engines can increase detection coverage, the number of false positives encountered during normal operation may increase when compared to a single engine. While antivirus vendors try hard to reduce false positives, they can severely impair productivity and take weeks to be corrected by a vendor.

The proposed architecture provides the ability to aggregate results from different detection techniques which enables the unique ability to trade-off detection coverage for false positive resistance. If an administrator wanted maximal detection coverage they could set the aggregation function to declare a candidate file unsafe if *any* detector indicated the file malicious. However, a false positive in any of the detector would cause the aggregator to declare the file unsafe.

In contrast, an administrator more concerned about false positives may set the aggregation function to declare a candidate file unsafe if at least half of the detectors deemed the file malicious. In this way multiple detection engines can be used to reduce the impact of false positives associated with any single engine.

To explore this trade-off, we collected 12 real-world false positives that impact different detectors in CloudAV. These files range from printer drivers to password recovery utilities to self-extracting zip files. We defined a threshold, or confidence index, of the number of engines required to detect a file before deeming it unsafe. For each threshold value, we measured the number of remaining false positives and also the corresponding detection rate of true positives.

The results of this experiment are seen in Table 3. At a threshold of 4 engines, all of the false positives are eliminated while only decreasing the overall detection coverage by less than 4%. As this threshold can be adjusted at any time via the management interface, it can set by an administrator based on the perceived threat model of the network and the actual number of false positives encountered during operation.

A second method of handling false positives is enabled by the centralized management of the network service. In the case of a standard host-based antivirus deployment, encountering a false positive may mean weeks of delay and loss of productivity while the antivirus vendor analyzes the false positive and releases an updated signature set to all affected clients. In the network-based architecture, the false positive can be added to a network-wide whitelist through the management interface in a matter of minutes by a local administrator. This whitelist management allows administrators to alleviate the pain of false positives and empowers them to cut out the antivirus vendor middle-man and make more informed and rapid decisions about threats on their network.

Threshold	False Positives	Detection
1	12	97.7%
2	5	96.3%
3	2	95.2%
4	0	93.9%

Table 3: The number of false positives observed at each engine threshold and the associated detection coverage over the full malware dataset.

7.6 Breaking Free of Vendor Lock-in

Finally, a serious issue associated with extensive deployments of host-based antivirus in a large enterprise or organizational network is vendor lock-in. Once a particular vendor has been selected through an organization's evaluation process and software is deployed to all departments, it is often hard to switch to a new vendor at a later point due to technical, management, and bureaucratic issues. In reality, organizations may wish to switch antivirus vendors for a number of reasons such as increased detection coverage, decreased licensing costs, or integration with network management devices.

The proposed antivirus architecture is innately vendor-neutral as it separates the acquisition of candidate files on the end host from the actual analysis and detection process performed in the network service. Therefore, even if only one detection engine is employed in the network service, a network administrator can easily replace it with another vendor's offering if so desired, without an upheaval of existing infrastructure.

8 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 deployability and management. Using a production implementation and real-world deployment of the CloudAV

platform, we evaluated the effectiveness of the proposed architecture and demonstrated how it provides significantly greater protection of end hosts against modern threats.

In the future, we plan to investigate the application of N-version protection to intrusion detection, phishing, 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.

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.

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.

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

[8] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza Basrai, and Peter M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementaiton (OSDI)*, December 2002.

[9] F-Secure Corporation. F-secure mobile anti-virus. http://mobile.f-secure.com/, 2007.

[10] Gartner, Inc. Forecast: Security software worldwide, 2006-2011, update. http://www.gartner.com/DisplayDocument?ref=g_search&id=510567&subref=advsearch, 2007.

[11] Hispasec Sistemas. Virus total. http://virustotal.com, 2004.

[12] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC 1997)*, May 1997.

[13] Kaspersky Lab. Kaspersky mobile security. http://usa.kaspersky.com/products_services/antivirus-mobile.php, 2007.

[14] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, 2003.

[15] 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.

[16] Mathias Rauen. madcodehook. http://madshi.net/, 2008.

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

[18] Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. Spyproxy: Execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium*, August 2007.

[19] Lajos Nagy, Richard Ford, and William Allen. N-version programming for the detection of zero-day exploits. In *IEEE Topical Conference on Cybersecurity*, Daytona Beach, Florida, USA, 2006.

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

[21] NIST/DHS/US-CERT. National vulnerability database. http://nvd.nist.gov/, 2007.

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

[23] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Rethinking antivirus: Executable analysis in the network cloud. In *2nd USENIX Workshop on Hot Topics in Security (HotSec 2007)*, August 2007.

[24] John Ogness. Dazuko: An open solution to facilitate on-access scanning. *Virus Bulletin*, 2003.

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

[26] Rodrigo Rodrigues, Miguel Castro, and Barbara Liskov. Base: using abstraction to improve fault tolerance. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2001.

[27] SANS. Top-20 2007 security risks. http://www.sans.org/top20/, 2007.

[28] S. Sidiroglou, J. Ioannidis, A.D. Keromytis, and S.J. Stolfo. An Email Worm Vaccine Architecture. *Proceedings of the 1st Information Security Practice and Experience Conference (ISPEC)*, pages 97–108, 2005.

[29] Stelios Sidiroglou, Angelos Stavrou, and Angelos D. Keromytis. Mediated overlay services (moses): Network security as a composable service. In *Proceedings of the IEEE Sarnoff Symposium*, Princeton, NJ, USA, 2007.

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

[31] Symantec Corporation. Symantec mobile antivirus for windows mobile. http://www.symantec.com/norton/products/overview.jsp?pcid=pf&pvid=smavwm, 2007.

[32] Symantec Security Response Team. Ms word exploit creation tool. http://www.symantec.com/enterprise/security_response/weblog/2007/04/ms_word_exploit_creation_tool.html, 2007.

[33] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the DOMINO overlay system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '04)*, San Diego, CA, February 2004.