Leveraging the Cloud for Software Security Services

by

Jonathan Clarke Oberheide

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor Farnam Jahanian, Chair
Professor Peter M. Chen
Associate Professor Zhuoqing Mao
Assistant Professor Eytan Adar
Assistant Research Scientist Michael Donald Bailey
Craig Partridge, BBN Technologies

To my mother Julie, father Clarke, sister Kristin, and girlfriend Heidi.

ACKNOWLEDGEMENTS

First and foremost, I'd like to thank my advisor Farnam Jahanian. Throughout my undergrad, masters, and doctoral studies at the University of Michigan, Farnam has been a close advisor to me on personal, academic, and professional levels. I cannot thank Farnam enough for the opportunities enabled by his guidance over the past decade of my life.

I want to thank the rest of my doctoral committee, including Michael Bailey, Morley Mao, Peter Chen, Craig Partridge, and Eytan Adar, all of whom have provided guidance, feedback, and encouragement throughout the doctoral process. I'd like to thank Bailey, especially, who has acted as a great mentor and friend in the doctoral process and provided invaluable leadership and direction in our research group. Special thanks to Morley as well for allowing me to work with her research group as an undergrad, providing me an early glimpse into the academic process.

I also want to thank the many people and partners outside the University that made this dissertation possible. This includes our colleagues at Arbor Networks, who have collaborated with our research group on projects ranging from malware analysis to large-scale network traffic analysis. Thanks to Jose Nazario, Dug Song, Craig Labovitz, Rob Malan, and the rest of the Arbor crew. I'd like to thank the folks at Merit Network for their collaboration and material support throughout the years. Manish Karir, Bert Rossi, and Larry Blunk were instrumental in our cooperation with Merit and its member institutions. Special thanks to Manish, who introduced me to the wonderful world of academia and guided me through my first steps in the research process as an undergrad. Thanks also to the IT and security staff at the University, spanning groups such as ITSS, CITI, CAEN, and DCO. Folks including Jim Rees, Paul Howell, Matt Bing, Don Winsor, and Laura Fink have been a great help in providing infrastructure services, access to data, and operational advice.

I'd like to thank all of the software faculty at the University that I would have loved to have on my doctoral committee if there were no limit on size, including Peter Honeyman, Jason Flinn, Alex Halderman, Brian Noble, and Atul Prakash. And, of course, I'd like to thank all the fellow doctoral students at the University that have provided friendship and entertainment in our CSE office, including Evan Cooke, Sushant Sinha, Yunjing Xu, Timur Alperovich, Mona Attariyan, Kaushik Veeraraghavan, Jodie Su, Dan Peek, Eric Vander Weele, Kelsey Harris, and Kaustubh Nyalkalkar. In particular, Evan acted as a great mentor, sounding board, and friend during my early years in the doctoral program.

Finally, and most importantly, I wish to thank my family. This thesis is dedicated to my mother Julie, my father Clarke, my sister Kristin, and my girlfriend Heidi.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

ABSTRACT

Leveraging the Cloud for Software Security Services

by

Jonathan Clarke Oberheide

Chair: Farnam Jahanian

This thesis seeks to leverage the advances in cloud computing in order to address modern security threats, allowing for completely novel architectures that provide dramatic improvements and asymmetric gains beyond what is possible using current approaches. Indeed, many of the critical security problems facing the Internet and its users are inadequately addressed by current security technologies. Current security measures often are deployed in an exclusively network-based or host-based model, limiting their efficacy against modern threats. However, recent advancements in the past decade in cloud computing and high-speed networking have ushered in a new era of software services. Software services that were previously deployed on-premise in organizations and enterprises are now being outsourced to the cloud, leading to fundamentally new models in how software services are sold, consumed, and managed.

This thesis focuses on how novel software security services can be deployed that leverage the cloud to scale elegantly in their capabilities, performance, and management. First, we introduce a novel architecture for malware detection in the cloud. Next, we propose a cloud service to protect modern mobile devices, an ever-increasing target for malicious attackers. Then, we discuss and demonstrate the ability for attackers to leverage the same benefits of cloud-centric services for malicious purposes. Next, we present new techniques for the large-scale analysis and classification of malicious software. Lastly, to demonstrate the benefits of cloud-centric architectures outside the realm of malicious software,

we present a threshold signature scheme that leverages the cloud for robustness and resiliency.

Thesis Statement: By leveraging properties inherent to cloud computing, it is possible to design new classes of cloud-centric software security services that scale elegantly in their capabilities, performance, and management.

CHAPTER 1

Introduction

Security threats have plagued Internet-connected devices for some time. In particular, malicious software has enabled attackers to achieve financial gains from botnets, credential theft, spam, denial-of-service, phishing, and other attacks. In recent years, the scale and sophistication of attacks on end users have increased dramatically. Therefore, it is vital to the overall health of the Internet and its users that we develop effective and efficient security mechanisms that are able to deter, detect, and defend against modern malicious threats.

1.1 Previous Approaches

1.1.1 The Evolution of Threats

Over the past decade, we've observed a distinct evolution of malicious threats. In the first half of the decade, we saw the explosion of network-based threats on the Internet. Early denial-of-service (DoS) attacks took down high-profile websites such as Yahoo, Amazon, and CNN with little difficulty by flooding them with excessive traffic and requests [146]. As attackers realized that taking control of an end host was more powerful than simply knocking it offline, the era of the flash worm began. Flash worms such as Code Red [131], Slammer [130], and Witty [160] targeted vulnerabilities in network-facing operating systems and services. These threats wreaked havoc on the availability and integrity of network infrastructure and vulnerable services exposed and listening on the Internet.

As the attack surface of network-facing services was minimized through proper isolation and access control, broad patching and mitigation of remote code execution vulnerabilities, and other mechanisms, the efficacy of network-based threats was reduced. In the second half of the decade, malicious threats evolved to target end hosts and users directly. Instead of exploiting network-facing services, attackers began to target the large host-based attack surface presented by client-side applications such as web browsers, PDF viewers, and office suites [182]. By simply tricking a user into visiting an untrusted link and exploiting a web browser vulnerability, the attacker could take full control of the user's host by installing their own malicious software, also known as *malware*. Many attackers realized that persistent, stealthy control of a large number of compromised end hosts enabled powerful attacks and new monetization models. This realization led to the era of botnets [50, 19]. Even as recent efforts to reduce the client-side attack surface of end hosts have made progress, attackers have continued to evolve their host-based attacks to include social engineering in order to trick users into installing malicious software [178, 92].

With concern, we've also observed an evolution in the sophistication of the actors behind the malicious attacks. In the early 2000s, many attacks were perpetrated by amateurs looking to explore the bounds of computing and security. Bored teenagers, politically-motivated hackivists, and feuding hacker groups were frequently responsible for website defacements and denial-of-service attacks. As bad actors realized that many types of attacks such as financial fraud, spam, and denial-of-service could be monetized, Internet-based attacks turned into a lucrative business opportunity. In the current day, it is well-understood that rich underground crimeware markets, organized cybercrime groups, and well-funded and sophisticated adversaries are active and responsible for many attacks on the Internet [80, 77]. With accusations of state-sponsored attacks and terms like cyberwar being thrown around in the current day [72, 39], it is clear that the security landscape has drastically evolved in the past decade.

To address the increasing sophistication and scale of malicious threats, researchers have proposed a broad range of software security technologies over the past decade, including systems such as intrusion detection systems, intrusion prevention systems, firewalls, and antivirus software. These systems have become essential components in detecting mali-

cious attacks and protecting end hosts and users.

While software security mechanisms can take many forms, a frequent goal for many such approaches is to observe network or host activity and distinguish between legitimate and malicious activity. We classify common software security mechanisms as network-centric or host-centric, depending on their deployment model and which type of activity they observe and inspect. For example, we would classify a sensor device placed on a network link to passively listen for malicious traffic destined for an enterprise network as a network-centric approach. On the other hand, we would classify antivirus software installed on every end host in an enterprise that inspects host activity to detect malicious software as a host-centric approach. As we will discuss, both models have pros and cons, depending on which type of threats they are designed to address.

1.1.2 Network-Centric Security Mechanisms

In a network-centric model, mechanisms such as network access control (NAC), intrusion detection (NIDS), and intrusion prevention (NIPS) are commonly deployed on an organization's network to inspect traffic sourced and destined from the end hosts to be protected. Network-based intrusion detection systems may employ both signature-based matching to detect known malicious attacks [37, 139] and anomaly detection algorithms to detect unknown attacks [93, 25]. Other detection systems may operate on a flow-based granularity using formats such as NetFlow [95] instead of operating on live network traffic.

A network-centric approach typically offers of a wide breadth of visibility across hosts on the network, but lacks the depth of visibility because it can only observe the bits present on the network link and not what those bits represent when received and processed by an end host. Network-centric mechanisms were prevalent during the early 2000s when dealing with the propagation and detection of flash worms [131, 130, 160]. More recently, network-based sensors have focused on the detection of malicious traffic and botnet activity [51, 86, 85].

However, as we've seen threats migrate from network attacks to host attacks, it has become increasingly difficult for network-centric approaches to maintain their efficacy.

Inspecting network traffic delivered over SSL transports that are becoming more commonplace is just one of the challenges. As attackers increasingly target users via client-side applications such as web browsers, PDF viewers, and office suites, network-centric mechanisms must be aware of a wide variety of complex file formats and have deep inspection capabilities to detect malicious code [182]. For example, attackers may deliver a browser exploit via obfuscated JavaScript, making it extremely difficult for a network-based sensor to detect the presence of malicious intent. Lastly, it is common nowadays to see encrypted botnet command and control traffic [17], causing challenges in identifying infected end hosts for network-centric mechanisms.

1.1.3 Host-Centric Security Mechanisms

As malicious threats migrated more and more towards targeting end hosts in the second half of the decade, more emphasis was placed on researching and developing host-centric security mechanisms. In a host-centric model, software such as antivirus and host-based intrusion detection systems (HIDS) is deployed on end hosts to monitor host activity and block malicious attacks. The most common approach for protecting end hosts is traditional antivirus engines [48, 129]. Such antivirus engines commonly operate on a file-based granularity to scan for malicious software that may enter an end host via a number of vectors. HIDS can also be used by monitoring and restricting system-level activity at execution time to detect malicious activity. Such approaches have been extensively explored in academia, whether observing instruction and system-call level information [44], analyzing OS and application information flow [188], or monitoring host activity from a hypervisor perspective [101].

Host-centric mechanisms systems may be able to inspect deep information and activity on a single end host, but they lack the global network-wide visibility that network-centric systems can often provide [118, 49, 151]. While both network-centric and host-centric systems are commonly deployed to attempt to detect and defend against malicious threats, it is clear that Internet-connected devices and users remain under continued attack. Therefore, this thesis advocates for exploring new deployment models beyond host-centric and

network-centric perspectives as threats continue to evolve in sophistication and scale.

1.2 Our Approach

1.2.1 The Evolution of Computing

While security threats have evolved considerably over the past decade, we've also witnessed significant advancements in the realm of computing. In particular, the introduction of cloud computing [172, 179, 126], bolstered by the popularization of x86 virtualization and increased availability of high-speed, low-latency networking [20, 156, 3], has ushered in a new era of software services. Cloud computing has enabled a wide range of new service models, including infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS), that change the way modern software services are sold, consumed, and managed [78, 144, 190, 42]. Cloud computing is succinctly defined in [14] as follows:

> Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services.
>
> The services themselves have long been referred to as Software as a Service (SaaS). The datacenter hardware and software is what we will call a Cloud. When a Cloud is made available in a pay-as-you-go manner to the general public, we call it a Public Cloud; the service being sold is Utility Computing.
>
> Thus, Cloud Computing is the sum of SaaS and Utility Computing.

Software services that were previously deployed on-premise in organizations and enterprises in a network-centric or host-centric model are now being outsourced to the cloud. Cloud-delivered services are leveraging the beneficial properties of cloud computing, such as reliability, scalability, performance, management, and cost, to more effectively deliver software technologies [14].

1.2.2 Cloud-Centric Software Security Services

While the revolution in cloud computing presents interesting security challenges in itself [100, 105, 40], this thesis seeks to understand how cloud computing and a SaaS deployment model can enable more effective software security services. That is, rather than investigating how we can "protect the cloud", this thesis seeks to understand how the cloud can protect us. While previous work has considered host-centric and network-centric approaches, this thesis focuses on how novel *cloud-centric architectures* can provide improved efficacy against modern security threats. While this thesis advocates for the exploration of cloud-centric services, it is not aimed at discounting existing approaches, just as host-centric services didn't completely replace network-centric services as threats moved from the network to the end host. Rather, we see cloud-centric services as the next phase in the deployment model of security services that follows from the intersection of the evolution of threats and evolution of cloud computing capabilities observed over the past decade.

1.2.3 Properties of Cloud-Centric Services

This thesis argues that many of the properties inherent to cloud computing enable us to construct effective and efficient cloud-centric software security services. While most of the properties may provide beneficial gains for software security services, there may also be trade-offs or potential negative consequences that must be carefully considered. As this thesis is deeply rooted in practical issues, these properties may have significant impact on the design of software security services and on real-world efficacy, deployment, and management. Several of the properties of cloud computing that are relevant to the security realm include the following:

- **Global visibility and network effects:** The potential for global visibility offered by a cloud-hosted service can provide significant value to a security service. In many scenarios, the more users or devices participating in a cloud-centric security service, the more intelligence can be collected, which can result in a network effect that increases the efficacy of the security service [118, 49, 151]. The visibility and network

effects of cloud-centric security services may span across hosts, networks, organizations, and even globally across the entire Internet. On the other hand, aggregating globally-scoped data in a centralized multi-tenant cloud environment may have an impact on cross-user and cross-organization privacy and confidentiality.

- **Scalability and elasticity:** Given the dramatic increase in malicious software and attacks [128, 127], it is imperative that any security mechanisms be rapidly scalable. In addition, such mechanisms need to be sufficiently elastic to handle highly variable security workloads. Failing to scale and adapt to heavy or variable loads, whether legitimately or intentionally generated, can potentially result in the failure of the security service. Cloud computing and public IaaS providers allow for the construction of highly scalable and elastic security services.

- **Agility and flexibility:** Cloud computing enables agility and flexibility through the rapid deployment of new functionality and software updates compared to the slow delivery model of traditional on-premise software [14, 82]. Cloud-centric security services can leverage such agility and flexibility in order to keep pace with an adaptive adversary. For example, updates or heuristics for the latest security threats may be deployed in real-time in a cloud-centric service, as opposed to pushing down software updates to an enterprise that will deploy them on-premise. In the field of security where threats change on a daily basis, this agility has very practical benefits. On the other hand, agility and flexibility may also empower attackers to build effective malicious cloud services themselves, which we will explore later in the thesis.

- **Availability and survivability:** Contrary to popular belief about the reliability of public cloud computing environments [113], the functionality offered by many IaaS providers and cloud computing platforms can enable the construction of highly available and survivable systems beyond what is typically feasible with fixed infrastructure [14]. Cloud-centric security services can leverage such functionality to achieve global accessibility and reliable connectivity. High availability is an absolute necessity when deploying security services that are often in the critical hot-path of computing systems.

- **Trustworthiness and host-proofing:** Moving security functionality to a cloud-centric service can introduce risk to confidentiality and integrity if the cloud provider is untrusted. However, some classes of cloud-centric services can be designed and deployed in such a way that does not sacrifice trustworthiness, even in the face of a compromised or malicious IaaS provider [65, 185]. By host-proofing software security services [60], one may achieve guarantees about the confidentiality and integrity of data handled by the cloud service. While achieving full host-proofing may not be feasible for all types of security services, maintaining as much trustworthiness as possible is an important goal to keep in mind when constructing cloud-centric services.

By leveraging these advancements in cloud computing, we are able to develop and deploy novel security services in a cloud-centric architecture that can scale elegantly in terms of their capabilities, performance, and management.

1.3 Contributions

The main contributions of this thesis are the following:

- **N-Version Malware Detection in the Cloud**

  While it's generally known in the security community that antivirus software is not a perfect solution, we provide novel quantification of the failure of commercial host-centric antivirus in terms of its detection coverage, attack surface, vulnerability window, and classification capabilities. To address these limitations, we introduce the CloudAV architecture, a core piece of research that moves the complexity of malware detection off of the end host and into the cloud. Deploying CloudAV security services in the cloud enables N-version protection, retrospective detection, forensic data collection, centralized management and policy definition, and a wide range of practical deployment benefits. Through a real-world implementation and evaluation, we find that migrating malware detection functionality from a host-centric to cloud-centric model is an effective approach.

- **Protecting Mobile Devices with a Cloud Service**

  While our results indicate that a cloud-centric approach to malware detection is quite effective when deployed on traditional end hosts, offloading analysis to the cloud lends itself naturally to the resource-constrained mobile environment. As mobile devices and applications become an enticing target for attackers, protecting these devices in an efficient and scalable manner is vital. Such protection is complicated by the diversity of mobile platforms and the intricacies of each platform's security model. Our research indicates that cloud-based malware detection is indeed an energy and resource efficient approach for mobile device security through implementations and evaluations on a range of modern mobile platforms.

- **The Dark Side of Cloud Services: Crimeware as a Service**

  While cloud-based security services have great potential for deploying protective mechanisms, it would be incomplete to ignore the fact that the positive attributes of cloud-centric services may also be abused to benefit attackers. To demonstrate the potential for malicious cloud services, we construct PolyPack, an example of an emerging model known as crimeware-as-a-service (CaaS). PolyPack is a cloud-based service for automated malware obfuscation and antivirus evasion that integrates a wide range of packers in front of CloudAV's backend antivirus engines, allowing researchers to understand the impact of malware packers of antivirus efficacy. Based on our observations in the underground crimeware markets, we anticipate that CaaS will present an attractive approach for attackers and concerning development for defenders in the near future.

- **Large-Scale Analysis and Classification of Malicious Software**

  Given the staggering growth of malicious software [128, 127], malware classification has emerged as an important mechanism to understand and quantify the malware epidemic. To address the limitations of existing automated classification and analysis tools, we developed and evaluated a dynamic analysis approach, based on the execution of malware in virtualized environments that are hosted in a cloud-centric service

and the causal tracing of the operating system objects affected during malware execution. The results of our analysis and clustering of runtime behaviors exhibit strong consistency, completeness, and conciseness. Furthermore, our research illustrates the necessity for scalable and performable cloud-centric architectures for heavyweight analysis of malicious software.

- **A Cloud-Centric Service for Robust and Resilient Threshold Signatures**

  Lastly, to demonstrate that the benefits of a cloud-oriented architecture apply to software security services beyond those directly related to malicious software, we design a threshold signature scheme that leverages a cloud service. Threshold cryptography offers attractive properties for requiring a number of independent parties to cooperate in order to perform a cryptographic operation, such as generating a RSA signature across multiple devices. We introduce an architecture, called CloudCard, that employs a threshold signature scheme with key material split across a user's host, a user's mobile device, and a cloud service. Using this (3,3) threshold RSA scheme, our CloudCard approach enables improved secrecy of private key material, flexible and fast revocation, and out-of-band signature confirmation, all while maintaining full compatibility with existing PKCS#11-enabled applications and interfaces. CloudCard also demonstrates that trustworthiness can be maintained even when moving security functionality to a cloud-centric service.

1.4 Structure of Thesis

This thesis is organized into five primary chapters. Chapter 2 presents a novel architecture for the deployment of malware detection services in the cloud. In Chapter 3, we discuss leveraging a cloud service to protect modern mobile devices. Chapter 4 explores the dark side of cloud services and the potential for attackers to leverage the same benefits of a cloud-centric service for malicious purposes. In Chapter 5, we discuss new techniques for the large-scale analysis and classification of malicious software. In Chapter 6, we present a threshold signature scheme that leverages the cloud for robustness and resiliency. Finally,

Chapter 7 concludes the thesis with a review of the lessons learned throughout this research and a look toward the future work in cloud-centric software security services.

CHAPTER 2

N-Version Malware Detection in the Cloud

In this chapter, we explore the problem of detecting malicious software using a cloud-centric architecture called CloudAV. As with most approaches of "enumerating badness" in security, detecting malicious software is a non-trivial problem. The vast, ever-increasing ecosystem of malicious software and tools presents a daunting challenge for network operators and IT administrators. Currently, antivirus is one of the most widely-used host-centric software security mechanisms for detecting and mitigating malicious software. However, the elevating sophistication of modern malware means that it is increasingly challenging for any single vendor to develop signatures for each new threat.

To address the growing sophistication and scale of malware, we propose a new cloud-centric deployment model for the detection of malicious software that is a departure from the traditional host-centric model of antivirus software. This architectural shift is characterized by two key changes:

- **Antivirus as a cloud service:** First, we propose that the detection capabilities currently provided by host-based antivirus software can be more efficiently and effectively provided as a cloud-centric security 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 cloud service for analysis, and then permit access to or quarantine them based on a report returned by the cloud service.

- **N-version protection:** Second, the identification of malicious 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 leverage the detection capabilities of multiple, heterogeneous detection engines to more effectively determine malicious files.

This new cloud-centric model provides several important and practical benefits:

- **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 software.

- **Enhanced forensic capabilities:** Information about which hosts accessed which 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.

- **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 bot infection is detected, a cloud-based 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.

- **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 model of deploying anti-malware security software, we design a cloud-based architecture that consists of three major components: a lightweight *host agent* run on end hosts that identifies new files and sends them into the network for analysis; a *cloud service* that receives files from hosts and identifies malicious 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 cloud antivirus system called CloudAV. CloudAV includes a lightweight, cross-platform host agent for Windows, Linux, and FreeBSD and a cloud service consisting of 10 antivirus engines and two behavioral detection engines. We provide a detailed evaluation of the system using a dataset of 7,220 malware samples collected in the wild over a period of a year [136] and a production deployment of our system on a campus network in computer labs spanning multiple departments for a period of over six 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 show how our architecture enables advanced functionality, such as retrospective detection, which can greatly mitigate the impact of the large window of vulnerability presented by antivirus products.

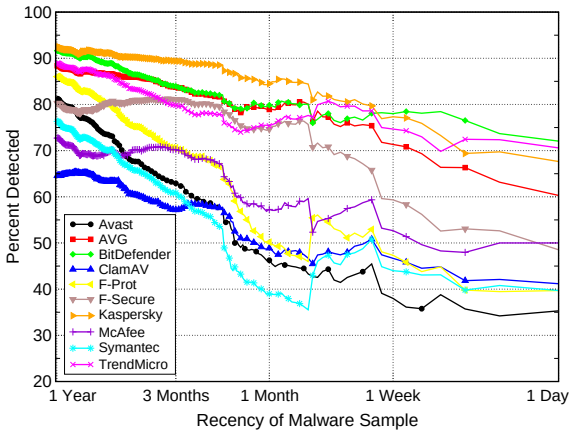
Finally, we analyze the performance and scalability of the system using deployment results and show that while the total number of executables run by all the systems in a computing lab is quite large (an average of 20,500 per day), the number of unique executables run per day is two orders of magnitude smaller (an average of 217 per day). This means that the caching mechanisms employed in the cloud 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.

2.1 Limitations of Antivirus Software

The ubiquitous deployment of antivirus software is closely tied to the ever-expanding ecosystem of malicious software and tools. 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 [17].

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 2.1: Detection rate for 10 popular antivirus products as a function of the age of the malware samples.

However, two important trends, that we detail in this section, call into question the long-term effectiveness of products from commercial antivirus vendors. 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 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 be and are being exploited by malware. That is, malware is actually exploiting vulnerabilities in antivirus software as means to infect systems.

2.1.1 Vulnerability Window

The sheer volume of new threats means that it is difficult for any single antivirus vendor to create signatures for all new threats. The ability of a vendor to create signatures for new threats 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 often a significant period of time between when a threat appears and when a signature is created by antivirus vendors. This period of time is known as the *vulnerability window*.

To quantify the vulnerability window, we analyzed the detection rate of multiple antivirus engines across malware samples that were collected over a one-year period. The dataset included 7,220 samples that were collected between November 11th, 2006, and November 10th, 2007. The malware dataset is described in further detail in our evaluation. The signatures used for the antivirus were updated the day after collection ended (November 11th, 2007) and stayed constant throughout 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 three months ago, one that included malware collected more recently than one month ago, and one that included malware collected more recently than one week ago. The antivirus engine and signature versions, along with their associated detection rates for each time period, are listed in Figure 2.1(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 one 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 during which the malware samples were collected. Figure 2.1(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 one 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 poses the greatest threat to end hosts.

2.1.2 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 [57, 58], overflows in decompression routines [59], etc.), remote exploits in management interfaces have been observed in the wild [55]. Due to the common need for elevated privileges by

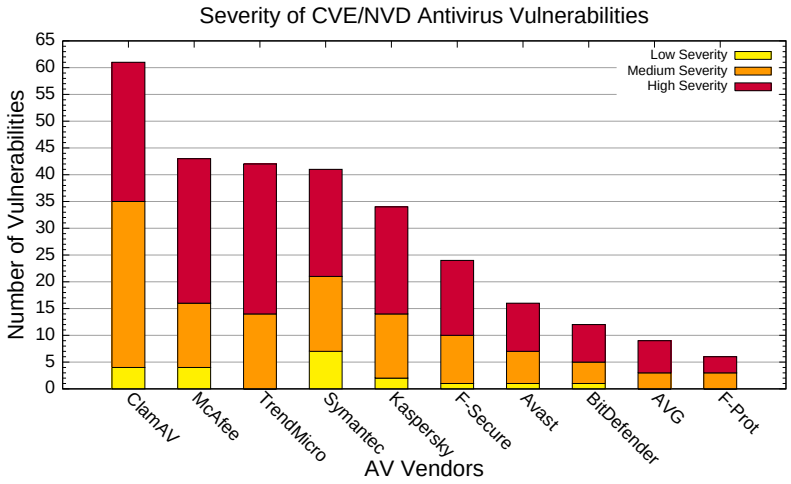


Figure 2.2: Number of vulnerabilities reported in the National Vulnerability Database (NVD) for 10 antivirus vendors between 2005 and 2007

antivirus software, many of these vulnerabilities result in a complete compromise of the affected end host.

Figure 2.2 shows the number of vulnerabilities reported in the National Vulnerability Database [140] for 10 popular antivirus vendors between 2005 and 2007. This large number of reported vulnerabilities demonstrates not only the risk involved in deploying antivirus software but also an evolution in tactics, as attackers are now targeting vulnerabilities in antivirus software itself.

2.2 Approach

This thesis 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 a cloud service. Second, the identification of malicious software should be determined by multiple, heterogeneous detection engines in parallel.

2.2.1 Deployment Environment

Before discussing the details of the approach, it is important to understand the environment in which such an architecture is most effective. While such an architecture can be very effective in a standalone deployment, the CloudAV system can also be deployed alongside existing antivirus engines and host-based intrusion detection systems. Some possible deployment environments include the following:

- **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.

Privacy implications: Shifting file analysis to a central location provides significant benefits but also has important privacy implications. It is critical that users of a cloud-based antivirus solution understand that their files may be transferred to another computer for analysis. There may be situations in which 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 of less 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 which files are cached can be controlled, depending on the policies of the network.

2.2.2 Cloud-Based Detection

The core of the proposed approach is moving the detection of malicious files off of end hosts and into the cloud. By moving the complexity of the detection engines off of the

end host, we significantly lower 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 [110, 55]. Finally, a lightweight host agent allows the service to be extended to resource-limited devices that lack sufficient processing power but remain an enticing target for malware.

While moving detection to the cloud has a number of benefits, the approach is not without trade-offs. Certainly, availability and connectivity to the cloud service is of paramount concern when moving services from a host-centric to cloud-centric model. We believe that cloud computing and high-speed networking makes the deployment of cloud-based detection more practical and worthy of exploration. Issues and policy surrounding disconnected operation are discussed in further detail later in this chapter.

2.2.3 N-Version Protection

Moving to a cloud-centric architecture also enables us to employ a set of heterogeneous detection engines that are used to provide analysis results on a file, which we call *N-version protection*. This approach is analogous 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 [15]. While N-version programming uses multiple implementations to increase fault tolerance in complex software, N-version protection uses multiple independent implementations of detection engines in order to increase coverage against a highly complex and ever-evolving ecosystem of malicious software.

2.3 Architecture

In order to move the detection of malicious files off of end hosts and into the network, several important challenges must be overcome: (1) unlike existing antivirus software, files must be transported into the network for analysis; (2) an efficient analysis system must

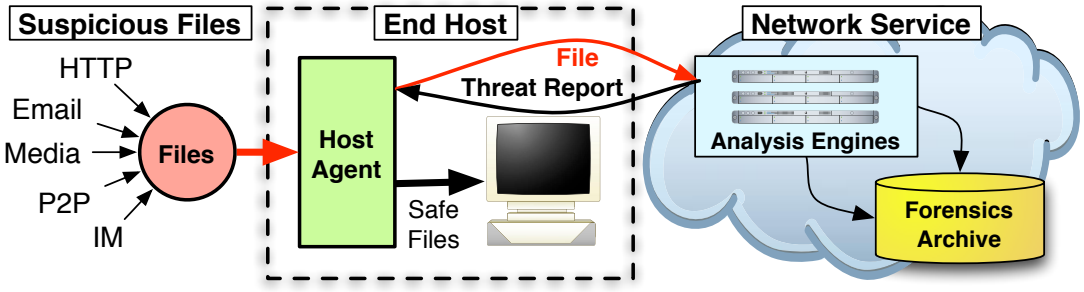


Figure 2.3: Architectural approach for cloud-centric file analysis service.

be constructed in order 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 to 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 hosts that identifies new files and sends them into the network for analysis. The second is a *cloud service* that receives files from the host agent, identifies malicious content, and indicates to 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 2.3 shows the high-level architecture of the cloud-based approach.

2.3.1 Client Software

Malicious files can enter an organization from many sources. For example, 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 cloud 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.

2.3.1.1 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 previously discussed, 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.

2.3.1.2 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 cloud service. One of the simplest methods of generating a UID is using 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 a completely different UID. To combat polymorphic threats, a more complex UID generator algorithm could be employed to avoid impacting cache performance. For example, a method such as locality-preserving hashing in multidimensional spaces [97] could be used to track differences between two files in a compact manner.

2.3.1.3 User Interface

We envision three majors modes of operation that affect how users interact with the host agent. These modes range from less to more interactive, depending on the policy and security requirements of the organization deploying CloudAV.

- **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 the user is presented a warning about why the file is suspicious. The user is then allowed to choose whether to proceed in accessing the file.

- **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 about it by an error dialog.

2.3.1.4 Other File Acquisition Methods

While the host agent is the primary method of acquiring candidate files and transmitting them to the cloud 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 that monitors 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, because network traffic can be encrypted, files may be encapsulated in unknown protocols, and the network is only one source of malicious content.

2.3.2 Cloud Service

The second major component of the architecture is the cloud service responsible for file analysis. The core task of the cloud service is to determine whether a file is malicious. 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 is made by aggregating these individual results into a threat report.

2.3.2.1 Detection Engines

A cluster of servers can quickly analyze files using multiple detection techniques. Additional detection engines can easily be integrated into a cloud 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 who use 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 [141], CWSandbox [36]) or virtual machine and records host state changes and network activity. Such deep analysis is difficult or impossible to accomplish on resource-constrained devices 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 correlate information between engines. For example, if a detector finds that the behavior of an unknown file is similar to that of a file previously classified as malicious by antivirus engines, the unknown file can be marked as suspicious.

2.3.2.2 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. For example, 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 whether 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. 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 explore the balance between coverage and confidence further in the discussion section.

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.

2.3.2.3 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 cloud 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 cloud service may also periodically be *pushed* to the host agent, to speed up future accesses, and invalidated when deemed necessary.

2.3.3 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 [106, 68] 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 cloud service, in addition to the behavioral profiles of malicious software generated by the behavioral detection engines. Depending on the privacy policy of an 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, because the exact operations performed by a piece of malicious software can be analyzed in detail. From a general administration perspective, knowledge about which 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 0-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*. This means that the complete archive of files that are transmitted to the cloud service may be re-scanned by the antivirus engines whenever a signature update occurs. Retrospective detection allows previously undetected malware that has infected a host to be identified and quarantined.

2.4 CloudAV Implementation

To explore and validate the proposed cloud-based 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.

2.4.1 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 cloud service, as executables are a common source of malicious content. We discuss how the agent can be extended to acquire DLLs, docu-

ments, and other common malcode-bearing files types in the discussion section.

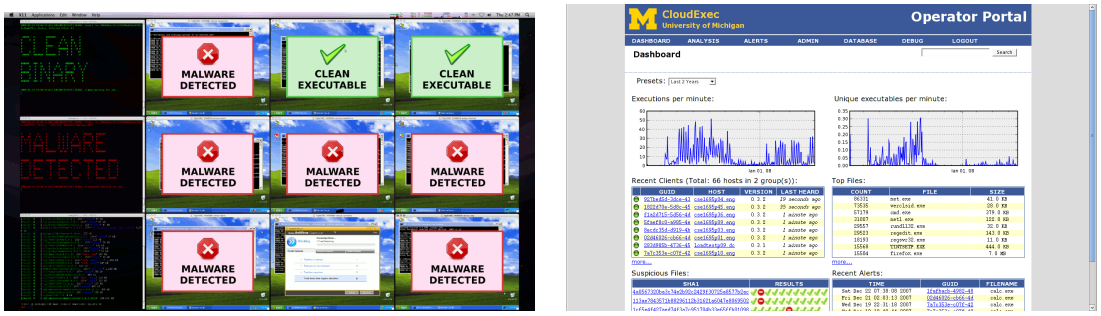
While the exact APIs are platform dependent (CreateProcess on Win32, execve syscall on Linux 2.4, LSM hooks on Linux 2.6, etc.), the host agent hooks and interposes on system events. This interposition is implemented via the MadCodeHook [152] package on the Win32 platform and via the Dazuko [147] framework 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 in order to identify new files entering a host and preemptively transfer them to the cloud 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 software, the host agent was designed to be simple and lightweight, both in code size and resource requirements. The Win32 agent is approximately 1,500 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. This Milter frontend allows us to scan all attachments on incoming emails. Using the Pymilter API, the Milter frontend weighs in at approximately 100 lines of code.

2.4.2 Cloud Service

The cloud 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 cloud service uses a HTTP wire protocol protected by mutually authenticated SSL/TLS. Between the components within the cloud service itself, communication is performed via a publish/subscribe bus to allow modularization and scalability.



(a) (b)

Figure 2.4: 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).

The cloud service allows for various priorities to be assigned to analysis requests to aid latency-sensitive applications and penalize misbehaving hosts. For example, application scanning may take higher analysis priority than background analysis tasks such as retroactive detection and mail scanning. This also enables the system to penalize or temporarily suspend misbehaving hosts that 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, 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 allow the network service to spin up multiple instances of a particular engine when demand for its services increases.

Our current implementation employs 12 engines: 10 traditional antivirus engines (Avast [5], AVG [6], BitDefender [7], ClamAV [168], F-Prot [8], F-Secure [9], Kaspersky [10], McAfee [11], Symantec [12], and Trend Micro [13]) and two behavioral engines (Norman Sandbox [141] and CWSandbox [36]). The exact version of each detection engine is listed in Figure 2.1(a). Nine of the backend engines run in a Windows XP environment using Xen's HVM capabilities, while the other three 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.

2.4.3 Management Interface

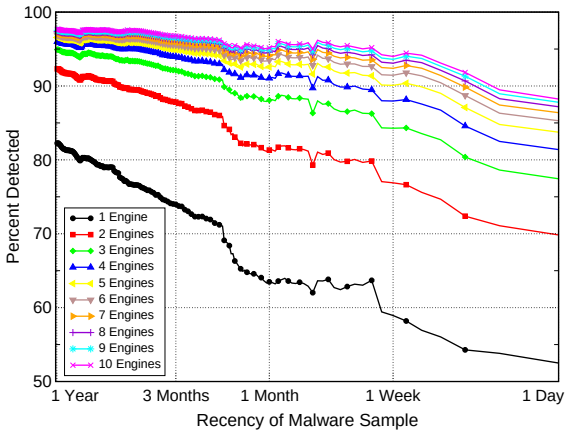
The third component of the CloudAV architecture is a management interface that provides access to the forensics archive, policy enforcement, alerting, and report generation. These interfaces are exposed to network administrators via a web-based management interface. The web interface is implemented using Cherrypy, a Python web development framework. A screen capture of the dashboard of the management interface is provided in Figure 2.4.

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 that describe 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 the enforcement of administrator-defined policies.

Network administrators may want, for example, 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 multiplayer 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.

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 2.5: 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.

2.5 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 one 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 the vulnerability window.

The second data source is derived from a production deployment of the system on a campus network in computer labs that span multiple departments for a period of over six 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 cloud 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.

2.5.1 Malware Dataset Results

Our evaluation is based on a malware dataset obtained through Arbor Networks' Arbor Malware Library (AML) [136]. 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 [16]. The AML dataset we used consists of 7,220 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.

We used the AML malware dataset to assess the effectiveness of a set of heterogeneous detection engines. Figure 2.5(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 three months of malware was 87.7%.

Figure 2.5(a) demonstrates how the use of multiple heterogeneous engines allows CloudAV to significantly improve the aggregate detection rate. Figure 2.5(b) shows the detection rate over malware samples ranging from one day old to one year old. The graph shows how using 10 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 10 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. Because 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 2.5 is the decrease in incremental coverage. Moving from one to two engines results in a large jump in detection rate, moving from two to three is smaller, moving from three to four 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 the discussion section.

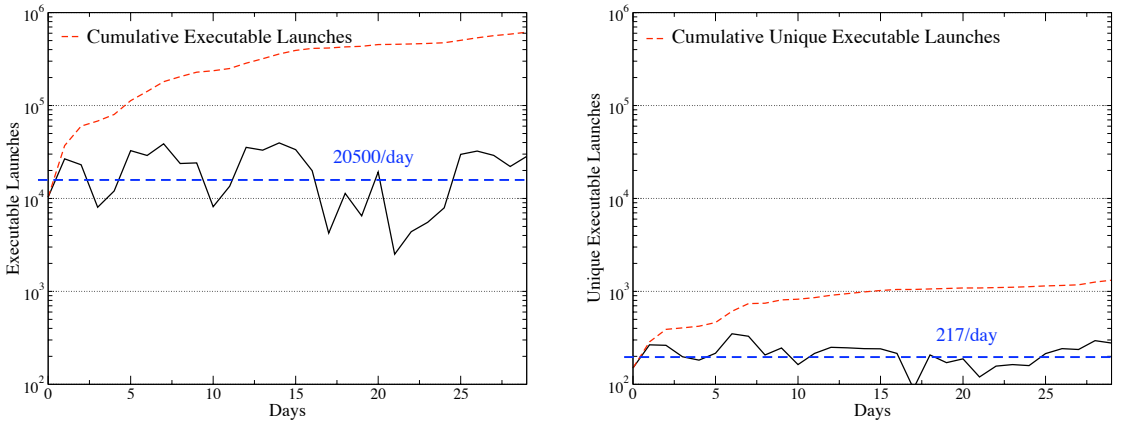
In addition to the averages presented in Figure 2.5, the minimum and maximum detection coverage for a given number of engines is of interest. For the one week time range, the maximum detection coverage when using only a single engine is 78.6% (Kaspersky) and the minimum is 39.7% (Avast). When using three 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.

2.5.2 Deployment Results

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

2.5.2.1 Executable Events

One of the core variables that impacts the resource requirements of the cloud 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 2.6 shows the



(a) (b)

Figure 2.6: 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.

number of total execution events and unique executables observed during a one month period in a university computing lab.

Figure 2.6 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 fewer unique executables.

We also investigated the origins of these executables based on the file path of 1,000 unique executables stored in the forensics archive. Table 2.1 shows the break down of these sources. The majority of executables originated 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 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.

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 2.1: A distribution of the sources of 1,000 executables observed during the deployment of our host agent over a six-month period.

2.5.2.2 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 file transfers to the cloud service. The hosts instrumented as a part of the deployment were heavily loaded Windows XP workstations. The Windows Start Menu contained more than 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 loaded to when the login screen appeared, 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 once 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 2.6. We recorded almost 615,000 total execution events over one month yet only observed 1,300 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 cloud 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 networks 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.

2.5.2.3 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 alongside 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 by 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 that 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 be 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.

2.6 Discussion and Limitations

Moving detection functionality into the 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.

2.6.1 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 cloud, there are two general techniques that a cloud-based 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 systems 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 cloud 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 complement cloud-based detection.

2.6.2 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 cloud 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 cloud

service. However, for files that have not yet been analyzed, a policy decision is necessary. Security-conscious organizations may select a strict policy that requires users to 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.

2.6.3 Sources of Malicious Behavior

Code or input that cause malicious 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 [96] and Microsoft Word [173] 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.

Our current CloudAV implementation focuses on executable files, 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 similarly to the executable itself: the local and remote cache is checked to determine whether it was previously analyzed, and if not, it is transmitted to the cloud 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 2.2: The percentage increase in detection coverage obtained when ClamAV, a truly free engine, is added to a deployment with only a single engine.

2.6.4 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 2.5, 10 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 three months, one month, and one 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 2.2.

2.6.5 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 2.3. At a threshold of four 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

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

Table 2.3: The number of false positives observed at each engine threshold, and the associated detection coverage over the full malware dataset.

actual number of false positives encountered during operation.

A second method of handling false positives is enabled by the centralized management of the cloud 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 inconvenience 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.

2.6.6 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, including 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 cloud service. Therefore, even if only one detection engine is employed in the cloud service, a network administrator can easily replace it with another vendor's

offering if desired, without an upheaval of existing infrastructure.

2.7 Related Work

Our approach of moving the detection of malicious software into the cloud is aligned with a strong trend toward moving services from end host and monolithic servers into the cloud. For example, in-network email [137, 46, 163] and HTTP [133, 149] 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 cloud services as overlay networks [164, 186].

Our use of N-version protection is closely related 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 [15]. Traditionally, N-version programming has been applied to systems requiring high availability such as distributed filesystems [155]. N-version programming has also been applied to the security realm to detect implementation faults in web services that may be exploited by an attacker [135].

A handful of online services have recently been constructed that implement N-version detection techniques. For example, there are online web services for malware submission and analysis [36, 90, 141]. 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, which results in vastly different architectural decisions and performance characteristics.

2.8 Summary

To address the ever-growing sophistication and scale of modern malicious software, we have proposed a new cloud-centric model for the deployment of antivirus functionality. By adapting a previously host-centric deployment model to leverage the cloud, our CloudAV approach provides significant advantages over traditional antivirus including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and

improved deployability and management.

2.8.1 Leveraging the Cloud

CloudAV offers a boost to dismal endpoint security mechanisms to protect against host-based threats. Just a few years ago, sending files to an external network-based service for analysis may have been perceived as infeasible from a performance perspective. However, the introduction of high-speed, low-latency networking and the widespread adoption of cloud computing technologies has made such a cloud-centric model for anti-malware services a feasible and enticing approach. In particular, the following properties of cloud computing proved invaluable in the design and implementation of the CloudAV service:

- **Availability:** With a security service like antivirus that is in the critical path of end host operation and user productivity, it is imperative that high availability be maintained. Cloud computing offers the ability to offer highly available cloud-based antivirus services at affordable costs.

- **Global Visibility:** Our cloud-hosted anti-malware service offers global visibility of threats across all the hosts participating in the service. Network effects can also be observed through higher cache hit rates when more hosts are participating in the CloudAV service as well as increased opportunities of retrospective detection by collecting a large sample of potentially malicious binaries.

- **Elasticity:** The file analysis workloads in CloudAV can be quite variable in normal operation and even more so if malicious parties intentionally increase workloads in an attempt to cause denial-of-service. Therefore, the elasticity provided by cloud computing services is key to adapt to and handle such variable workloads by spinning up additional backend detection engines on demand.

- **Isolation:** By isolating the complexity of the detection engine from the end host, CloudAV can protect end hosts from the impact of AV engine vulnerabilities. The cloud service also insulates itself from these vulnerabilities by hosting the backend detection engines in isolated and disposable virtual machines.

CHAPTER 3

Protecting Mobile Devices with a Cloud Service

In the previous chapter, we discussed our CloudAV approach to protecting traditional end hosts from malicious software using a cloud-centric security service. One of the key side-effects of moving the complexity of CloudAV's detection engines to the cloud was the simplification of the software agent that resides on each end host. We discussed some of the benefits of a lightweight host agent including a reduced attack surface and increased portability across operating systems. One important side-effect not as thoroughly investigated was the reduced performance impact on the host itself by alleviating that host from having to execute expensive detection routines locally. While such performance gains may not be as observable or important on common desktop platforms with plenty of spare CPU and memory, resource-constrained devices may observe considerable gains by the CloudAV approach. One common class of resource-constrained devices is consumer mobile devices, which are receiving much more attention from both attackers and defenders in recent years. In this chapter, we discuss our motivation and approach to adapting the CloudAV cloud service and host agent to consumer mobile devices.

Modern mobile platforms such as Google's Android, Apple's iPhone, and Nokia's Maemo run near-complete versions of commodity operating systems like BSD and Linux. Functionality like complete multi-protocol networking stacks, UI toolkits, and file systems provide developers with a rich environment to quickly build applications but open up devices to the same wide range of threats that target desktops. Over a thousand native third-party applications were developed for the iPhone platform before the official SDK was even

released [142], several hundred have been developed for Nokia's Maemo platform [54], and thousands of developers are creating applications for Google's Android platform [83].

To date, security vendors have marketed mobile-specific versions of antivirus software [112, 56, 53]. However, as the complexity of mobile platforms and threats increase, we argue that mobile antivirus solutions will look more like their desktop variants. The functionality required to detect sophisticated malware can have significant drain on computation and power which are critical resources on mobile devices.

To conserve scarce mobile resources and improve detection of modern threats we advocate moving mobile antivirus functionality to an off-device cloud service. The core of this approach is expending bandwidth to reduce on-device CPU and memory resources and thereby save power. We foresee three important benefits:

- **Better detection of malicious software:** Once detection functionality is offloaded to a cloud service, significantly more resources can be dedicated to evaluating each suspicious file. Our approach uses detection engines running inside a cloud service providing mobile devices with the protection capabilities of heavyweight behavioral detection engines.

- **Reduced on-device resource consumption:** By transferring files to a cloud-centric service for analysis, we argue that overall CPU use, memory use, and power can be reduced compared to performing the analysis on-device. Even more important, the cloud service can scale and be extended with new signatures and detection engines without using additional resources on mobile devices. In exchange, the approach may be more taxing on the mobile device's radio and network data usage.

- **Reduced on-device software complexity:** Modern threats have become extremely sophisticated, requiring complex antivirus software to detect and mitigate threats. By deploying a relatively simple agent on mobile devices and pushing complex detection software into the network, the complexity of mobile software can be minimized. This reduces the on-device attack surface and the effort required to port the agent to the numerous and evolving mobile platforms.

Engine Combination	Detected	Coverage
CM	229/469	48.82%
CM, SM	290/469	61.83%
CM, SM, MA	358/469	76.33%
CM, SM, MA, BD	417/469	88.91%
CM, SM, MA, BD, FS	430/469	91.68%

Table 3.1: An example of the increased detection coverage against a dataset of a recent month's worth of malware samples when using multiple engines in parallel: ClamAV (CM), Symantec (SM), McAfee (MA), BitDefender (BD), and F-Secure (FS).

To explore the idea of a cloud-based malware detection service for mobile devices, we extend the CloudAV platform with an on-device mobile agent and an off-device mobile-specific behavioral detection engine. Through a series of benchmarks comparing CloudAV to existing on-device antivirus software, we find that our mobile agent consumes an order of magnitude less CPU and memory, consumes less power in common scenarios, and offers greater protection capabilities that scale against future threats.

3.1 Mobile CloudAV

When designing the CloudAV architecture, we transitioned the complexity of the detection engines from the end host into the cloud. The result was a very lightweight agent that remained on the end host. While not initially designed for mobile platforms, the lightweight host agent was an intuitive application for protecting resource constrained mobile devices.

3.1.1 Mobile Agent

Extending the benefits of the CloudAV platform requires that an agent be deployed on a mobile platform. Given that the CloudAV platform inherently encourages a simple on-device agent, few fundamental modifications to the architecture are necessary to develop and support a mobile agent. One of the notable differences between the traditional host agent and our newly developed mobile agent is the constrained power, computation, and network resources available to the mobile device. Therefore, the file identifier algorithms

and communications protocol with the cloud service are important, as the agent spends most of its cycles on those activities.

We developed a mobile agent to interface with the CloudAV service for the Linux-based Maemo platform and deployed it on a Nokia N800 mobile device. The mobile agent is implemented in Python and uses the Dazuko [147] framework to interpose on system events. Specifically, we hook the `execve(2)` syscall and file system operations to acquire and process candidate files before permitting their access. The mobile agent required only 170 lines of code.

3.1.2 Mobile-Specific Behavioral Engine

A more resource-intensive method of detecting malicious activity is through behavioral analysis. Behavioral engines attempt to emulate or run real operating systems and applications to determine whether a file is performing malicious behavior at runtime. While these engines usually require a great deal of resources, which would not be suitable for a mobile device, deploying such an engine in the cloud service allows us to gain the protection benefits without the resource costs.

To demonstrate this point, we developed a mobile-specific behavioral detection engine. The behavioral engine runs candidate mobile applications in a virtualized Maemo operating environment hosted in the cloud service and monitors the application's system calls and D-Bus interprocess communication for malicious behavior. To exemplify the capability to detect and mitigate a malicious application, the behavioral engine flags potentially malicious actions such as modifying or destroying a user's personal data, initiating outgoing calls to unrecognized numbers via Skype, and initiating socket communications to untrusted or blacklisted destinations.

3.1.3 Connectivity and Mobile Data Usage

Mobile devices may enter a disconnected state where the mobile agent may not be able to effectively utilize the network-based security services. However, because connectivity will often be required to acquire new applications and content, the need for analysis in a

disconnected state may be minimal. Furthermore, mobile devices are rapidly increasing in connectivity capabilities with multiple radios for high-speed data transmission.

Users of mobile devices may also have concerns around the amount of cellular data usage incurred by such a service. While we anticipate that caching mechanisms and the commonality of applications will result in reasonable data usage amounts, to not consider the practical deployment aspects of such a cloud service would be incomplete. Non-technical aspects of the deployment may assist in resolving this issue. For example, if the cloud service is deployed by a mobile service provider to protect its subscribers, the provider may waive the data usage for users of the service.

3.1.4 Additional Security Services

The mobile security services that can be hosted in a cloud service are not limited to antivirus functionality. We envision a cloud-centric platform enabling a range of different security services.

- **SMS Spam Filtering:** SMS spam filtering functionality, which is currently implemented in an ad-hoc manner by some mobile antivirus products [112], can be much more accurate in a cloud-centric deployment model through the aggregation of data from a large corpus of users.

- **Phishing Detection:** Just as a centralized view of the web has helped Google develop strong anti-phishing tools [84], a centralized view of mobile activity in the service provider can help mobile operators detect and prevent phishing attacks against their customers.

- **Centralized Blacklists:** Blacklists of various communication addresses such as Bluetooth and IP may be implemented as an off-device security service. These blacklists can be maintained on a global level by a service provider for known malicious entities or on a personal user-specified level. These centralized policies may be opportunistically pushed to client devices for enhanced performance.

Agent	Startup Time	Avg Mem	Peak Mem	User Jiffies	Total Jiffies
ClamAV	57 sec	25967 KB	39556 KB	13349	15684
MA-CL+CR	0.2 sec	1502 KB	2154 KB	1502	2185
MA-CL+WR	0.2 sec	1486 KB	2124 KB	1486	1854
MA-WL+WR	0.2 sec	1189 KB	1812 KB	1189	1714

Table 3.2: Comparison of the mobile agent with ClamAV in memory consumption and CPU jiffies on the Nokia N800.

Many of the above services are currently deployed in an ad-hoc manner on individual mobile devices, potentially losing out on the network effect and security gains that can be gleaned when numerous users participate in the service and provide feedback to a central service.

Most importantly, the proposed architecture significantly lowers the bar for extending novel security services to mobile devices. For example, if a security vendor develops a new algorithm that is effective against detecting malicious mobile applications, that technique can be seamlessly integrated into the network service and put into operation without affecting any of the existing mobile devices. This transparent extensibility is a very powerful tool as mobile platforms and their needs rapidly evolve.

3.2 Evaluation

For our evaluation, we perform a series of benchmarks on two Nokia mobile devices. We measure the resource and power consumption of these devices and compare our mobile agent with existing commercial antivirus products. For each experiment, we provide results for three cache states for our mobile agent (MA): CL+CR: cold local, cold remote; CL+WR: cold local, warm remote; and WL+WR: warm local, warm remote.

3.2.1 Computational Resources

In the first experiment, we compare the CPU and memory consumption of the ClamAV [168] engine with our mobile agent on the Nokia N800. This benchmark serially runs common applications: the built-in N800 web browser, the Skype VoIP client, the Pid-

Agent	Avg / Peak / Total Energy
None (Baseline)	0.36 / 0.63 / 43.2 W
Kaspersky	0.86 / 1.27 / 89.4 W
MA-CL+CR (EDGE)	1.51 / 2.31 / 250.6 W
MA-CL+CR (WiFi)	1.31 / 2.44 / 165.1 W
MA-CL+WR (EDGE)	1.22 / 2.13 / 126.9 W
MA-CL+WR (WiFi)	0.92 / 1.83 / 74.5 W
MA-WL+WR	0.82 / 1.20 / 59.9 W

Table 3.3: Comparison of the mobile agent with Kaspersky Mobile Security on the Nokia N95.

gin IM client, the Kagu media player, and a PDF viewer. The application binaries and associated shared libraries, 346 files in total, are all processed by the particular engine. CPU usage is measured in both the number of jiffies the process has been scheduled for in userspace (utime) and total jiffies (utime + stime). The memory is based on the resident set size (RSS) of the process, or the number of non-shared memory pages currently in use by the process.

The results of the benchmark are listed in Table 3.2. ClamAV requires approximately 18 times as much memory and over eight times as much CPU time than the worst-case cache configuration for the mobile agent. In addition, the ClamAV engine has an extremely lengthy initialization process due to the loading of its signature database.

3.2.2 Power Consumption

In the second experiment, we perform a micro benchmark with a Nokia N95 smartphone. We measure the power consumption required to analyze files locally with Kaspersky's Mobile Security [112] software and compare it to using the mobile agent and network service. For instances in which the mobile agent needs to access the cloud service for cache queries or file transfers, we compare both the WiFi and GRPS/EDGE radios on the N95. The files analyzed are a collection of third-party applications and games totaling approximately 25 megabytes.

The results of the experiment are listed in Table 3.3. This experiment exemplifies the importance of the local and remote caching mechanisms. While the cold-remote cache

Detection Engine	Signature Database Size
Symantec Mobile	27 signatures
Kaspersky Mobile	284 signatures
ClamAV	262289 signatures
Mobile Agent	$>$ 5 million sigs + behavioral

Table 3.4: The number of threats addressed in the signature database of various detection engines.

states result in increased power consumption due to the energy of the radio transmission, a cold cache configuration is the worst case scenario which rarely occurs in practice. Both the warm-local/warm-remote and cold-local/warm-remote cache states, which are arguably the most common scenario, outperform the local Kaspersky engine in terms of consumed power. While we observed extremely high remote cache hit rates in a desktop environment, it is unclear whether the commonality of applications and associated cache hit rate would be similar in a mobile environment.

3.2.3 Scale of Detection Algorithms

Table 3.4 shows the number of threats in each detection engine's signature database. Our mobile agent vastly outperformed ClamAV on the N800 device while protecting against an order of magnitude more threats. While the power overhead of the mobile agent in the worst case was greater than Kaspersky's antivirus software, Kaspersky only scanned for 284 threats, roughly four orders of magnitude less than the CloudAV service.

Our results demonstrate that the current model of on-device antivirus software is not scalable. As the number and complexity of mobile threats increase, on-device engines and their signature databases will require more processing, storage, and power. On the other hand, our mobile agent remains constant in its resource requirements and can easily accommodate new signatures and entirely new engines in the cloud service.

3.2.4 On-Device Software Complexity

Our anecdotal experience with on-device antivirus software exemplifies their complexity and inability to deal with mobile platform diversity. First, the ClamAV software running on the N800 caused the device to randomly reboot when performing a normal system scan, making reliable evaluation tedious. Second, the N95 evaluation was originally planned to be with Symantec's Norton Smartphone Security software which advertises compatibility with N95's OS (Symbian Series 60 version 3). However, when we initiated a basic file scan on the N95, Norton would simply return `error -15` and stop execution, with no further information. In comparison, our model of using a lightweight mobile agent greatly reduces on-device software complexity and failures.

3.3 Related Work

Several mobile services [73, 79, 111, 158, 180, 189] have advocated leveraging remote execution by moving services off-device to minimize resource consumption while achieving performance targets. Our work is novel in the proposition of migrating complex security services to a network-based detection service to provide enhanced protection capabilities to mobile devices, while also achieving reduced complexity and resource consumption.

Furthermore, work such as [38] shows how security practitioners increasingly leverage virtualization to improve host security. Researchers have also explored the use of on-device virtualization for mobile security applications [61].

3.4 Summary

In this chapter, we investigated the emerging field of mobile security and demonstrated the application of a cloud-centric security service for the detection of malicious applications on consumer mobile devices. While mobile attacks are not yet widespread, taking an early, proactive look at mobile security may prove to be beneficial. As attackers begin to feel out the space of mobile security and discover economic incentives to target mobile users and devices, such a cloud-centric approach will undoubtedly be necessary to address mobile

threats.

3.4.1 Leveraging the Cloud

Throughout this chapter, we learned that a cloud-centric model for deploying software security services is an intuitive fit for the unique characteristics of mobile devices. In particular, the following properties of cloud computing proved to be invaluable in the design and implementation of the mobile CloudAV service:

- **Scalability and Availability:** Just like with CloudAV, cloud computing allows us to build a highly-scalable and highly-available service for protecting mobile devices on Internet-scale deployments. In addition, by scaling up detection capabilities in the cloud instead of on the device, the on-device resource requirements for anti-malware functionality remain static.

- **Flexibility:** Given that mobile security is a rapidly emerging and evolving field of security, having the ability to rapidly develop and deploy new security mechanisms in a flexible way is critical.

- **Isolation:** As our mobile-specific behavioral detection engine executes potentially malicious applications during analysis, the isolation provided by virtualized environments common to cloud computing environments is vital in order to maintain the integrity of the cloud service.

CHAPTER 4

The Dark Side of Cloud Services: Crimeware as a Service

While we have shown that cloud-centric security services have great potential for protecting end hosts and mobile devices against malicious software, it would be incomplete to ignore the fact that the positive attributes of cloud services may also be abused by attackers. Based on our observations in the underground crimeware markets, we anticipate that malicious cloud services, known as Crimeware-as-a-Service (CaaS), will present an attractive model for attackers. In this chapter, we explore the potential of CaaS, investigate the weaknesses of antivirus unpacking capabilities, and construct and evaluate a proof-of-concept cloud-centric service called PolyPack, which demonstrates an effective CaaS model for evading malware detection techniques.

PolyPack is a cloud-centric service that employs an array of packers, tools employed by malware authors to obfuscate their malicious software, to produce a binary that is able to evade detection by antivirus engines. A typical usage of the PolyPack service is as follows: (1) a user submits an unpacked binary via PolyPack's interface, (2) the binary is packed using an array of packers and each packed version is analyzed by an array of antivirus engines, and (3) the detection results from the antivirus engines are analyzed to select the optimally-packed version that provides the most antivirus evasion, and (4) the results of the PolyPack service are returned to the user.

PolyPack is targeted at providing a useful service for penetration testers who require the ability to create payloads that will evade the signature detection of a number of antivirus engines. For example, a penetration tester may be unaware of the antivirus product

running on a target host and may desire maximal evasion. While it is certainly possible to manually pack a binary and test it against local antivirus engines, we believe that offering an automated service can offload the burden of a manual effort and offer advanced features and considerable extensibility.

Beyond its legitimate utility for penetration testers, PolyPack is an interesting concept with respect to the crimeware industry. While crimeware such as LuckySploit, Mpack, WebAttacker, phishing kits, and others have traditionally been sold in an ad-hoc manner, crimeware authors may find that a software-as-a-service (SaaS) or subscription model is an attractive alternative. Many benefits like centralized development, management, and updating that drive the SaaS model for traditional software may translate effectively for crimeware tools as well. We believe that PolyPack represents the sophistication and automation of a crimeware-as-a-service (CaaS) deployment that is likely to emerge in the underground in the near future.

As with many penetration testing tools, PolyPack's offensive capabilities may be used for both good and evil. While unrestricted public access to such a service may border on irresponsible, we believe that PolyPack can be responsibly deployed or licensed in a similar vein as many of the existing penetration testing frameworks such as CANVAS [94]. In addition to its offensive capabilities, PolyPack allows defensive researchers to better understand the current state of packer efficacy and the resulting impact that packers may have on host-based protection mechanisms such as antivirus. Better understanding the offensive capabilities of attackers can lead researchers to create more effective and resilient defenses.

Toward these goals, we make three primary contributions:

- An analysis of a large dataset of malware of nearly 100,000 samples detailing the detection coverage of 23 antivirus engines against malware packed by 35 packer classes.

- The development of the cloud-based PolyPack service, complete with a backend of 10 popular antivirus engines, 10 common packers, and supplementary features such as live updating.

- An evaluation of the increased evasion capabilities of the PolyPack cloud service using 208 distinct malware samples compiled from source, each packed by 10 packers for a total of 2,288 samples.

4.1 AvP: Antivirus vs. Packers

Before delving into the architecture of the PolyPack cloud service, we investigate the diversity of packer usage observed in the wild as well as the varying detection capabilities of antivirus engines using a dataset of 98,801 malware samples from Arbor Networks' Arbor Malware Library (AML) [136]. The 98,801 samples represent over a year's worth of AML collection.

4.1.1 Packer Classification

It is important to understand the current use of packers in the wild before investigating their efficacy against modern antivirus engines. Many factors may influence the diversity of packers used by malware authors including their feature sets, resistance to antivirus and reverse engineering, and availability of the packing tool.

Table 4.1 shows the breakdown of packer usage in our dataset as determined by SigBuster [174] and PEiD [102] respectively, both signature-based packer identification tools. The top 10 packer classes only represent 55.3% and 33.3% of the total samples respectively, indicating a substantial distribution of packers.

In addition, a significant portion of the malware samples remain unidentified by SigBuster and PEiD, two of the most common tools for identifying packers. Of the 98,801 samples, 28,827 and 39,731 were unsuccessfully identified by SigBuster and PEiD (with BoB's userdb.txt) respectively. Of the 39,731 unidentified by PEiD, only 8,174 (20.6%) compress by more than 20%, indicating that the vast majority of these samples are indeed packed by unidentified packers. In addition, analysis of the overall binary entropy and the number of IAT entries when comparing the malicious binaries with a set of legitimate ones indicates that over 90% of all the samples in our dataset appear to be packed.

SigBuster Identifier	Count
Allaple	22050
UPX	11324
PECompact	5278
FSG	5080
Upack	3639
Themida	1679
NsPack	1645
ASPack	1505
tElock	1332
Nullsoft	1058

PEiD Identifier	Count
UPX	11244
Upack	6079
PECompact	4672
Nullsoft	2295
Themida	1688
FSG	1633
tElock	1398
NsPack	1375
ASpack	1283
WinUpack	1234

Figure 4.1: The top 10 packers classes in our AML dataset as determined by PEiD and SigBuster.

4.1.2 Antivirus Detection

More interesting than packer diversity is the wide range of detection capabilities that antivirus engines have when analyzing packed binaries. Such information can help us determine whether there are antivirus engines that are more effective against a wide range of packers or packers that are effective at evading a wide range of antivirus engines.

Figure 4.2 shows a set of 23 antivirus products (a subset of those supported by VirusTotal [90]) crossed with the top 35 most popular packer families determined by PEiD in our malware dataset (including non-packed and compiler classes such as Borland). Each square represents the detection coverage of the particular antivirus engine against the malware samples packed by the particular packer. The greyscale shade of the square represents the percentage of samples successfully detected as malicious, ranging from 0% (white) to 100% (black).

More important than individual antivirus or packer performance is observing the diversity of detection coverages across the board. The detection ranges can vary widely not only within a particular antivirus vendor against different packers but also across vendors with a particular packer. These results hint that the selection of a packer to ensure optimal evasion of antivirus engines may not be a trivial task. These results also provide the primary motivation for our construction of the PolyPack service to automate this process.

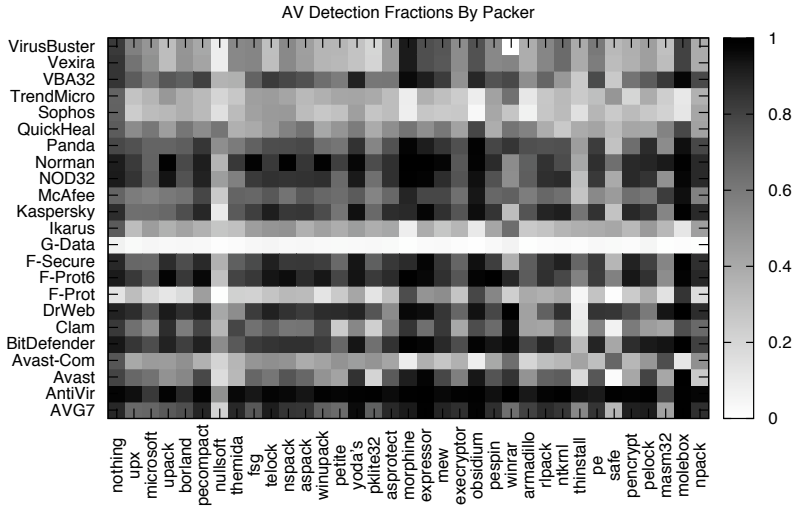


Figure 4.2: The fraction of detected binaries for 23 antivirus products and 35 most popular packers.

4.2 The PolyPack Cloud Service

In this section, we briefly describe the overall architecture and implementation of the PolyPack cloud service and then discuss several of its additional features that may be beneficial to penetration testers or other offensive parties.

4.2.1 PolyPack Architecture

The PolyPack architecture consists of three components: the web frontend, the packer service, and the antivirus service. An overview of the architecture is described in Figure 4.3.

4.2.1.1 Web Frontend

Users of PolyPack interact with the system through a simple web interface. The user can upload an unpacked binary and have it submitted for processing. PolyPack will process the binary with its collection of packers and antivirus engines and return the packed binary for optimal antivirus evasion to the user. Results from the PolyPack service can either be displayed in real-time in the user's browser or simply emailed to the user for later retrieval.

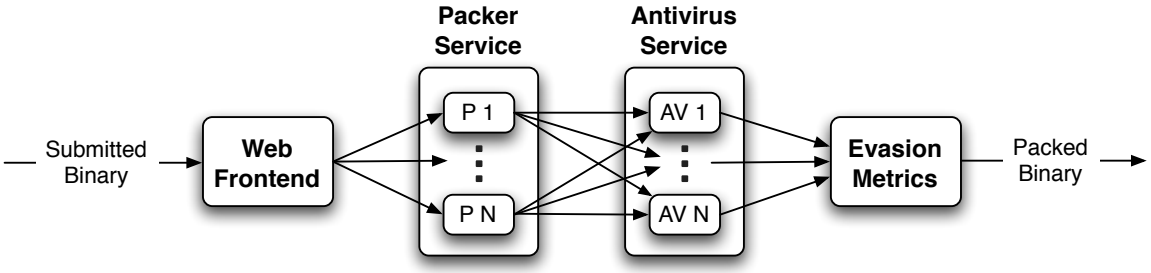


Figure 4.3: Conceptual overview of the PolyPack architecture.

4.2.1.2 Packer Service

Submitted binaries are first delivered to the packer service to be processed by an array of packers before being passed on to the antivirus service.

PolyPack's packing service currently supports 10 packers: ASPack [171], FSG [21], NsPack [166], Nullsoft [143], PECompact [175], tElock [177], Themida [176], UPX [145], WinUpack [69], and Yoda [34]. These packers were chosen as they represent some of the most common packers used by malware observed in the wild. While choosing common packers may result in greater detection rate by antivirus than more obscure packers, it provides a more representative view of the detection coverage experienced in the real world. We plan to significantly extend the set of supported packers in the future as adding additional packers is a trivial task. Only 32 lines of code on average are needed to support a packer.

4.2.1.3 Antivirus Service

After submitted binaries are processed by the packer service, they are fed into the antivirus service to be analyzed by a number of antivirus engines. The results from the various engines are collected and evaluated to determine the optimal packing scheme.

PolyPack's antivirus service supports a backend of 10 popular antivirus engines including Avast [5], AVG [6], BitDefender [7], ClamAV [168], F-Prot [8], F-Secure [9], Kaspersky [10], McAfee [11], Symantec [12], and Trend Micro [13]. These engines are hosted in disposable virtualized environments to maintain the integrity of the service. Again, the framework is designed around being extremely extensible for new antivirus engines, re-

quiring only 42 lines of code on average to support an engine.

4.2.2 PolyPack Features

While the use of antivirus engines as a feedback mechanism to ensure evasion before releasing malware is a common practice among attackers, we believe that automating the process in a centralized service allows us to offer considerable extensibility and unique features beyond what may be feasible by a manual effort.

4.2.2.1 Evasion Metrics

What constitutes "optimal" evasion may differ based on the goals of the user of PolyPack. The default metric for determining how effectively a particular samples evades antivirus is by a simple count of the number of engines that fail to detect it. However, in the real world, not all antivirus vendors are deployed uniformly and some are significantly more popular and commonly deployed than others. Therefore, PolyPack offers the ability to weight evasion metrics by antivirus vendor market share size. In the future, we will allow users to specify their own weighting preferences based on the specific engines they would like to evade.

4.2.2.2 Live Updating

Once a malware sample is released into the wild, it may fall into the hands of an analyst or antivirus vendor that produces a signature to block it. Continually monitoring a large number of antivirus engines may be a tiresome task for a malware author trying to determine whether a new signature update detects their malware. Since PolyPack possesses both the submitted binary and the signature update feeds of the antivirus engines, it can automatically re-pack and push out a new optimally-packed binary to the PolyPack user whenever a antivirus signature update is received. For up-to-the-minute evasion, live binaries may even be pulled from PolyPack directly and delivered to the victim via HTTP instead of being pushed to the PolyPack user.

4.2.2.3 Submission Confidentiality

As payloads may contain sensitive information or techniques, it is important that submissions to such a service be kept confidential. Unlike services such as VirusTotal [90] that share submitted samples with antivirus vendors, PolyPack maintains the confidentiality of submitted binaries. In addition, the backend antivirus engines are isolated from the network to ensure that samples or hash identifiers are not collected by the engine and leaked back to the antivirus vendor.

4.2.3 Future Capabilities

While our current implementation offers useful functionality, we plan to extend PolyPack to support more sophisticated capabilities:

- **Additional Packers:** Including additional exotic and sophisticated packers and techniques [165] in addition to the current set of well-known packers should significantly increase the flexibility of PolyPack's evasion.

- **Additional File Formats:** Besides the currently supported Portable Executable (PE) binaries, it would be useful to support additional binary formats, such as ELF and Mach-O, as well as other common file types used to transport malcode, such as PDFs, documents, and media files.

- **Automated Unpacking Resistance:** A significant number of projects have attempted to tackle the problem of automated unpacking [124, 161, 104, 18, 157, 150]. Providing resistance to these tools, in addition to antivirus engine evasion, is desirable.

- **Behavioral Antivirus:** Many antivirus engines augment their signature databases with the runtime behavioral detection of malware. Instrumenting the backend to provide feedback on whether a binary tripped an antivirus engine's behavioral detection would be valuable to the PolyPack user.

Packer	Total	Median	Average
Unpacked	212	1	1.02
ASpack	+128	+0	+0.61
FSG	+39	+0	+0.19
NsPack	+239	+1	+1.15
Nullsoft	+646	+3	+3.11
PECompact	+509	+2	+2.45
tElock	+424	+2	+2.04
Themida	+935	+5	+4.50
UPX	+91	+0	+0.44
WinUpack	+230	+1	+1.11
Yoda	+654	+3	+3.14
Average	+389	+2	+1.87
PolyPack	+1005	+5	+4.83

Table 4.1: For each packer, we list the increase over the unpacked binaries of the total number of antivirus evasions across all binaries (out of 2,080) and the median/average number of evasions per binary (out of 10).

4.3 Evaluation

To evaluate the efficacy of the PolyPack cloud service, we analyze a dataset of 208 malware samples compiled from source. Each of these unpacked binaries is fed through the PolyPack service, resulting in a total of 2,288 (208 unpacked and 2,080 packed) samples to be scanned by the antivirus engines. Since we can analyze both the packed and unpacked binaries, we can detail exactly how effective each individual packer is at evading the antivirus engines and determine how much benefit the PolyPack service provides.

Table 4.1 details the results of our evaluation. The first row shows the baseline for the unpacked binaries. Across all of the 208 unpacked binaries and the 10 antivirus engines, a sample goes undetected only 212 times, out of a total 2,080 scan events. On average across the 208 binaries, a sample only evades 1.02 of the 10 engines. After establishing this baseline for the unpacked binaries, we can measure how much additional evasion each individual packer provides. For example, the best individual packer, Themida, goes undetected in 935 more scan events than the baseline and corresponds to an average of evading 4.50 antivirus engines for a single sample. Table 4.1 also details the evasion efficacy if one were to choose a packer that represented the average of the 10 common packers in our

Packer	Optimal Occurrences
Themida	122
Nullsoft	59
Yoda	24
PECompact	3

Table 4.2: The number of occurrences a packer produced the optimal packing for each of the 208 distinct samples.

evaluation.

At first glance, it may be tempting to select Themida for packing all binaries since it appears to be the "best" of our evaluated packers. However, while Themida has the highest evasion overall, it is not the optimal choice for *all* of the binaries. Since PolyPack is able to evaluate the best packer choice for each binary individually, it is able to achieve a greater evasion rate as seen in the last row of Table 4.1. Overall, PolyPack is over 250% more effective at evading antivirus engines than picking a packer at random. Futhermore, while Themida is the best packer individually, PolyPack picks a better packer for over 40% of the samples.

Table 4.2 breaks down the cases when packers other than Themida provide the optimal packing (for 86 binaries of the 208 binaries). Nullsoft provides the best evasion for 59 of the binaries, Yoda for 24 of the binaries, and PECompact for 3 of the binaries. These results clearly indicate that one packer does not fit all and the variety provided by PolyPack has a real and measurable effect on a sample's evasion of antivirus engines. As a more diverse set of packers are added to PolyPack, we expect these results to become even more pronounced.

4.4 Related Work

Packers, crypters, protectors, and other tools that assist in the evasion of antivirus software have been used by malware authors for years. For example, the popular UPX packer [145] was first released in 1998 and used to legitimately compress executables distributed over the Internet before malware authors adopted its functionality for evading

Cloud Type	Legitimate	Crimeware
IaaS	Amazon EC2, Mosso	Renting out infected bots
PaaS	Google App Engine, Azure	Botnet-backed spam services
SaaS	SalesForce, SAP ByDesign	Packing services, Decaptcha

Table 4.3: Parallels exist between the cloud computing models of legitimate services and crimeware services.

signature-based antivirus software. In response to the alarming efficacy of simple packers against modern antivirus products, researchers have attempted to tackle the difficult problem of automated unpacking [124, 161, 104, 18, 157, 150]. While such research has shown promising results, those results have yet to have an observable impact on the real-world efficacy of antivirus products.

More generally, PolyPack is an illustrative example of crimeware-as-a-service. Malicious parties are keen to adopt the latest technological advances in order to increase the efficacy of their attacks, and cloud computing platforms offer numerous benefits. While PolyPack represents a novel approach for deploying malware packing services, it is not the first cloud-centric service that can be used for illegitimate purposes. For example, botnet rentals [116] for DDoS, spam, and other attacks, exploit kits [24], phishing kits [88], and automated or crowd-sourced CAPTCHA-solving services [63, 31, 32] are all commonly used and widely available crimeware services.

4.5 Summary

In this chapter, we explored the construction, deployment, and real-world evaluation of a cloud-centric service for the packing and analysis of binaries. We believe such a service is of value to researchers and penetration testers and is likely to represent the sophistication and automation that we will continue to see in the crimeware industry. In fact, as seen in Table 4.3, attackers have already capitalized on the advantages of various cloud computing models to deploy crimeware-as-a-service (CaaS). Understanding the deficiencies of our current defensive measures and the ease with which attackers can offensively innovate using CaaS can aid our own efforts in developing adequate protection mechanisms.

4.5.1 Leveraging the Cloud

PolyPack represents a glimpse into the future of sophisticated cloud-centric crimeware services while simultaneously demonstrating the weaknesses in current host-centric security mechanisms. The packing of malware samples for antivirus evasion, previously a manual non-optimal operation performed by malware authors, can now be effectively deployed as a cloud service with the support of cloud computing. In particular, the following properties of cloud computing proved invaluable in the design and implementation of the PolyPack service and future CaaS:

- **Empowerment:** PolyPack displays the power offered by cloud computing services by demonstrating how a simple cloud-hosted service run by an unsophisticated party can single-handedly defeat the combined capabilities of the entire antivirus industry. Such empowerment for crimeware services should serve as an alarming warning for the security community.

- **Agility:** PolyPack greatly benefits from the agility provided by cloud-delivered services as it is able to adapt in real-time to the latest signature updates from antivirus vendors. Users of the PolyPack service can receive the freshest results and optimally-packed binaries instantly.

- **Elasticity:** The typical workloads experienced by a service such as PolyPack can benefit from the elasticity of cloud computing. Individual packers experience vastly different performance characteristics and dynamic scaling of particular packer under a heavy workload is important for a cost-effective and practical deployment.

CHAPTER 5

Large-Scale Analysis and Classification of Malicious Software

As discussed in the previous chapters, effectively detecting malicious software is a complex and difficult problem. However, detection is not the only game in town in the realm of malware. Indeed, classifying how malicious software operates is becoming an increasingly important problem. Unfortunately, due to the thousands of new malware samples flooding in each day, the sheer scale of the task of efficient and effective malware classification is daunting, yet an intuitive fit for a cloud-based service. Deploying such a classification service in a cloud-centric model not only allows us to scale flexibly and leverage virtualization technology for safely executing and tracing malware, but it also offers the opportunity to open the service to other researchers, thereby resulting in more effective classification through the network effect. In this chapter, we explore, construct, and evaluate such a service to classify and analyze malicious software.

Previous efforts to automatically classify and analyze malware focused primarily on content-based signatures. Content-based signatures are inherently susceptible to inaccuracies due to polymorphic and metamorphic techniques [41]. As a result, antivirus products often characterize malware in ways that are inconsistent across products, incomplete across malware, and fail to be concise in their semantics. This creates an environment in which defenders are limited in their ability to share intelligence across organizations, to detect the emergence of new threats, and to assess risk in the quarantining and cleanup of infections.

Dataset Name	Number of Unique MD5s	Number of Unique Labels				
		McAfee	F-Prot	ClamAV	Trend	Symantec
legacy	3,637	116	1,216	590	416	57
small	893	112	379	253	246	90
large	3,698	310	1,544	1,102	2,035	50

Table 5.1: The number of unique labels provided by five antivirus engines is listed for each dataset.

To address the limitations of existing automated classification and analysis tools, we have developed and evaluated a dynamic analysis approach, based on the execution of malware in a cloud-based virtualized environment and the causal tracing of the operating system objects affected during malware execution. The reduced collection of these user-visible system state changes (e.g., files written, processes created) is used to create a fingerprint of the malware's behavior. These fingerprints can be used in assessing the potential damage incurred, enabling detection and classification of new threats, and assisting in the risk assessment of these threats in mitigation and clean up. To address the sheer volume of malware and the diversity of its behavior, we provide a service to automatically categorize these malware profiles at-scale into groups that reflect similar classes of behaviors.

5.1 Understanding Malware Labeling

The primary task of antivirus engines is to detect and prevent malicious software from compromising end hosts. As a normal part of this process, these engines also provide a label and short description for the malware they detected. The ability of these products to successfully label and characterize these threats has far-reaching effects: from facilitating sharing across organizations, to detecting the emergence of new threats, and assessing risk in quarantine and cleanup. However, for this information to be effective, the descriptions provided by these engines must be meaningful.

Antivirus engines rarely use the exact same labels for a threat, and users of these engines have come to expect simple naming differences (e.g., W32Lovsan.worm.a versus Lovsan versus WORM_MSBLAST.A) across vendors. It has always been assumed, however, that

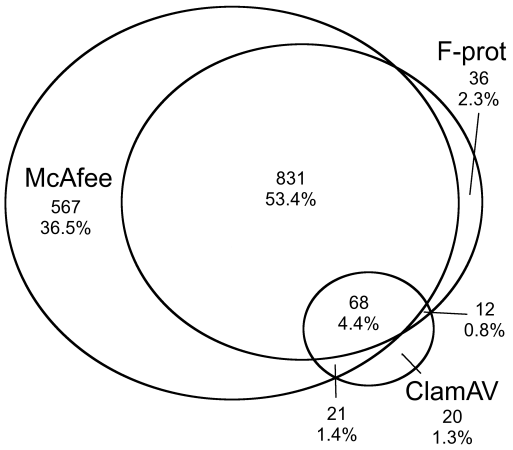


Figure 5.1: A Venn diagram of malware samples labeled as SDBot variants by three antivirus products.

there existed a simple mapping from one vendor's name space to another, and recently, investigators have begun creating projects to unify these name spaces [22]. Unfortunately, the task appears daunting. Consider, for example, the number of unique labels created by various engines. The result in Table 5.1 is striking, because there exists a substantial difference in the number of unique labels created by each antivirus engine. While one might expect small differences, it is clear that antivirus vendors disagree not only on what to label a piece of malware, but also on how many unique labels exist for malware in general.

One simple explanation for these differences in the number of labels is that some of these antivirus engines provide a finer level of detail into the threat landscape than the others. For example, the greater number of unique labels in Table 5.1 for F-Prot may be the result of F-Prot's ability to more effectively differentiate small variations in a family of malware. To investigate this conjecture, we examined the labels of the legacy dataset produced by the antivirus engines and, using a collection of simple heuristics for the labels, created a pool of malware classified by F-Prot, McAfee, and ClamAV as SDBot [125]. We then examined the percentage of time each of the three antivirus engines classified these malware samples as part of the same family. The result of this analysis can be seen in Figure 5.1. Each antivirus classifies a number of samples as SDBot, yet the intersection of

these different SDBot families is not clean, since there are many samples that are classified as SDBot by one antivirus and as something else by the others.

These differences in classification go beyond simple differences in labeling: antivirus products assign distinct semantics to differing pieces of malware. Consistent, complete, and concise labels are clearly necessary for effective malware classification.

5.2 Properties of a Labeling System

Our analysis of the classification capabilities of current antivirus engines in the previous section has provided a great deal of evidence indicating that labeling across antivirus products does not operate in a way that is useful to researchers, operators, or end users. Before we evaluate these systems any further, it is important to precisely define the properties an ideal labeling system should have. We have identified three key design goals for such a labeling system:

- **Consistency:** Identical items must and similar items should be assigned the same label.
- **Completeness:** A label should be generated for as many items as possible.
- **Conciseness:** The labels should be sufficient in number to reflect the unique properties of interest, while avoiding superfluous labels.

5.3 Limitations of Antivirus Labeling

Having identified consistency, completeness, and conciseness as the design goals of a labeling system, we are now prepared to investigate the ability of antivirus products to meet these goals.

	legacy				
	McAfee	F-Prot	ClamAV	Trend	Symantec
McAfee	100%	13%	27%	39%	59%
F-Prot	50%	100%	96%	41%	61%
ClamAV	62%	57%	100%	34%	68%
Trend	67%	18%	25%	100%	55%
Symantec	27%	7%	13%	14%	100%
	small				
	McAfee	F-Prot	ClamAV	Trend	Symantec
McAfee	100%	25%	54%	38%	17%
F-Prot	45%	100%	57%	35%	18%
ClamAV	39%	23%	100%	32%	13%
Trend	45%	23%	52%	100%	16%
Symantec	42%	25%	46%	33%	100%

Table 5.2: The percentage of time two binaries classified as the same by one antivirus are classified the same by other antivirus products. Malware is inconsistently classified across antivirus vendors.

5.3.1 Consistency

To investigate consistency, we grouped malware into categories based on the labels provided by antivirus vendors. For each pair of distinct malware labeled as the same by a particular vendor, we compared the percentage of time the same pair was classified by other antivirus products as the same. For example, two binaries in our legacy dataset with different MD5 checksums were labeled as W32-Blaster-worm-a by McAfee. These two binaries were labeled consistently by F-Prot (both as msblast) and Trend (both as msblast), but inconsistently by Symantec (one blaster and one not detected) and ClamAV (one blaster, one dcom.exploit). We then selected each vendor in turn and used its classification as the base. For example, Table 5.2 shows that malware classified by McAfee as the same was only classified as the same by F-Prot 13% of the time. However, malware classified by F-Prot as the same was only classified as the same by McAfee 50% of the time. Not only do antivirus products place malware into different categories, but these categories don't hold the same meaning across vendors.

Dataset Name	antivirus Updated	Percentage of Malware Samples Detected				
		McAfee	F-Prot	ClamAV	Trend	Symantec
legacy	20 Nov 2006	100%	99.8%	94.8%	93.73%	97.4%
small	20 Nov 2006	48.7%	61.0%	38.4%	54.0%	76.9%
small	31 Mar 2007	67.4%	68.0%	55.5%	86.8%	52.4%
large	31 Mar 2007	54.6%	76.4%	60.1%	80.0%	51.5%

Table 5.3: The percentage of malware samples detected across datasets and antivirus vendors. Antivirus does not provide a complete categorization of the datasets.

	legacy (3,637 binaries)		small (893 binaries)	
	Unique Labels	Clusters or Families	Unique Labels	Clusters or Families
McAfee	116	34	122	95
F-Prot	1216	37	379	62
ClamAV	590	41	253	65
Trend	416	46	246	72
Symantec	57	31	90	81

Table 5.4: The ways in which various antivirus products label and group malware. Antivirus labeling schemes vary widely in how concisely they represent the malware they classify.

5.3.2 Completeness

As discussed earlier, the design goal for completeness is to provide a label for each and every item to be classified. For each of the datasets and antivirus products, we examined the percentage of time the antivirus product detected a given piece of malware (and hence provided a label). A small percentage of malware samples are still undetected a year after the collection of the legacy datasets (Table 5.3). The results for more recent samples are even more profound, with almost half the samples undetected in the small dataset and one quarter in the large dataset. The one quarter undetected for the large set is likely an overestimate of the ability of the antivirus, as many of the binaries labeled at that point were many months old (e.g., compare the improvement over time in the two labeling instances of small). Thus, antivirus products do not provide a complete labeling system.

5.3.3 Conciseness

Conciseness refers to the ability of the labeling system to provide a label that reflects the important characteristics of the sample without superfluous semantics. In particular, we find that a label that carries either too much or too little meaning has minimal value. To investigate this property, we examined the number and types of labels and groups provided by the antivirus products. Table 5.4 shows the number of unique labels provided by the antivirus products as well as the number of unique families these labels belong to. In this analysis, the family is a generalized label heuristically extracted from the literal string, which contains the portion intended to be human-readable. For example, an antivirus product returned the literal labels W32-Sdbot.AC and Sdbot.42, which are both in the "sdbot" family. An interesting observation from this table is that these products vary widely in how concisely they represent malware. Vendors such as Symantec appear to employ a general approach, reducing samples to a small handful of labels and families. On the other extreme, FProt appears to aggressively label new instances, providing thousands of unique labels for malware, but still maintaining a small number of groups or families to which these labels belong.

5.4 Behavior-based Malware Clustering

As we described in the previous section, any meaningful labeling system must achieve consistency, completeness, and conciseness, and existing approaches, such as those used by antivirus products, fail to perform well on these metrics. To address these limitations, we propose an approach based on the actual execution of malware samples and observation of their persistent state changes. These state changes, when taken together, make a behavioral fingerprint, which can then be clustered with other fingerprints in order to define classes and subclasses of malware that exhibit similar state change behaviors. In this section, we discuss our definition and generation of these behavioral fingerprints and the techniques for clustering them.

5.4.1 Defining and Generating Malware Behaviors

Previous work in behavioral signatures has been based at the abstraction level of low-level system events, such as individual system calls. In our approach, the intent is to capture what the malware actually does on the system. Individual system calls may be at a level that is too low for abstracting semantically meaningful information: a higher abstraction level is needed in order to effectively describe the behavior of malware. We define the behavior of malware in terms of non-transient state changes that the malware causes on the system. State changes are a higher level abstraction than individual system calls, and they avoid many common obfuscation techniques that foil static analysis as well as low-level signatures, such as encrypted binaries and non-deterministic event ordering. In particular, we extract simple descriptions of state changes from the raw event logs obtained from malware execution. Spawned process names, modified registry keys, modified file names, and network connection attempts are extracted from the logs, and the list of such state changes becomes a behavioral profile of a sample of malware.

Observing the malware behavior requires actually executing the binaries. We send each malware sample to a cloud service that hosts virtualized VMware-based environments [181] with Windows XP installed. Such a cloud-based service allows us to scale up and out as the number of malware samples increases. The cloud service and its virtual machines are partially firewalled so that the external impact of any immediate attack behaviors (e.g., scanning, DDoS, and spam) is minimized during the limited execution period. The system events are captured and exported to a centralized repository using the Backtracker system [107]. In addition to exporting system events, the Backtracker system provides a means of building causal dependency graphs of these events. The benefit of this approach is that we can validate that the changes we observe are a direct result of the malware, and not of some normal system operation.

5.4.2 Clustering of Malware

While the choice of abstraction and generation of behaviors provides useful information to users, operators, and security personnel, the sheer volume of malware makes manual

Label	MD5	P/F/R/N	McAfee	Trend
A	71b9...	8/13/27/0	Not detected	W32/Backdoor.QWO
B	be5f...	8/13/27/0	Not detected	W32/Backdoor.QWO
C	df1c...	1/1/6/1	W32/Mytob.gen@MM	W32/IRCBot-based!Maximus
D	5bf1...	1/1/6/2	W32/Mytob.gen@MM	Not detected
E	eef8...	1/2/8/3	PWS-Banker.gen.i	W32/Bancos.IQK
F	80f6...	2/11/28/1	IRC/Flood.gen.b	W32/Backdoor.AHJJ
G	1258...	1/4/3/1	W32/Pate.b	W32/Parite.B
H	ff0f...	1/2/8/1	Not detected	W32/Bancos.IJG
I	36f6...	3/22/29/3	IRC/Generic Flooder	IRC/Zapchast.AK@bd
J	c13f...	5/32/28/1	Generic BackDoor.f	W32/VB-Backdoor!Maximus

Table 5.5: 10 unique malware samples. For each sample, the number of process, file, registry, and network behaviors observed and the classifications given by various antivirus vendors are listed.

analysis of each new malware intractable. Our malware source observed 3,700 samples in a six-month period, representing over 20 new pieces per day. Each generated fingerprint, in turn, can exhibit many thousands of individual state changes (e.g., infecting every .exe on a Windows host). For example, consider the tiny subset of malware in table 5.5. The 10 distinct pieces of malware generate from 10 to 66 different behaviors with a variety of different labels, including disjoint families, variants, and undetected malware. While some items obviously belong together in spite of their differences (e.g., C and D), even the composition of labels across antivirus products can not provide a complete grouping of the malware. Obviously, for these new behavioral fingerprints to be effective, similar behaviors need to be grouped and appropriate meanings assigned.

Our approach to generating meaningful labels is achieved through clustering the behavioral fingerprints. In the following subsections, we introduce this approach and the various issues associated with effective clustering, including how to compare fingerprints, combine them based on their similarity, and determine which are the most meaningful groups of behaviors.

	A	B	C	D	E	F	G	H	I	J
A	0.06	0.07	0.84	0.84	0.82	0.73	0.80	0.82	0.68	0.77
B	0.07	0.06	0.84	0.85	0.82	0.73	0.80	0.82	0.68	0.77
C	0.84	0.84	0.04	0.22	0.45	0.77	0.64	0.45	0.84	0.86
D	0.85	0.85	0.23	0.05	0.45	0.76	0.62	0.43	0.83	0.86
E	0.83	0.83	0.48	0.47	0.03	0.72	0.38	0.09	0.80	0.85
F	0.71	0.71	0.77	0.76	0.72	0.05	0.77	0.72	0.37	0.54
G	0.80	0.80	0.65	0.62	0.38	0.78	0.04	0.35	0.78	0.86
H	0.83	0.83	0.48	0.46	0.09	0.73	0.36	0.04	0.80	0.85
I	0.67	0.67	0.83	0.82	0.79	0.38	0.77	0.79	0.05	0.53
J	0.75	0.75	0.86	0.85	0.83	0.52	0.85	0.83	0.52	0.08

Table 5.6: A matrix of the NCD between each of the 10 malware samples in our example.

5.4.3 Comparing Individual Malware Behaviors

While examining individual behavioral profiles provides useful information on particular malware samples, our goal is to classify malware and give them meaningful labels. Thus malware samples must be grouped. One way to group the profiles is to create a distance metric that measures the difference between any two profiles, and then use the metric for clustering. Our initial naive approach to defining similarity was based on the concept of edit distance [52]. In this approach, each behavior is treated as an atomic unit and we measure the number of inserts of deletes of these atomic behaviors required to transform one behavioral fingerprint into another. The method is fairly intuitive and straightforward to implement; however, it suffers from two major drawbacks:

- **Overemphasizing size:** When the size of the number of behaviors is large, the edit distance is effectively equivalent to clustering based on the length of the feature set. This overemphasizes differences over similarities.

- **Behavioral polymorphism:** Many of the clusters we observed had few exact matches for behaviors. This is because the state changes made by malware may contain simple behavioral polymorphism (e.g., random file names).

To solve these shortcomings we turned to normalized compression distance (NCD). NCD is a way to provide approximation of information content, and it has been successfully applied in a number of areas [1, 183]. NCD is defined as:

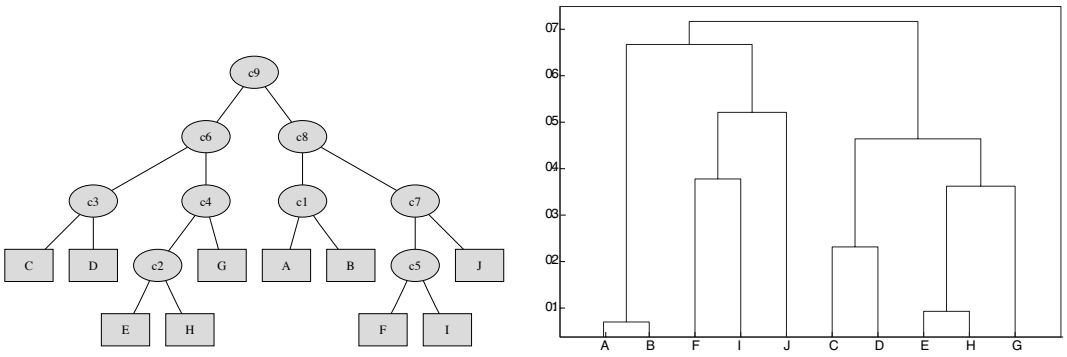


Figure 5.2: On the left, a tree consisting of the malware from Table 5.5 has been clustered via a hierarchical clustering algorithm whose distance function is normalized compression distance. On the right, a dendrogram illustrating the distance between various subtrees.

$$NCD(x,y) = \frac{C(x+y) - min(C(x),C(y))}{max(C(x),C(y))}$$

where "x + y" is the concatenation of x and y, and C(x) is the zlib-compressed length of x. Intuitively, NCD represents the overlap in information between two samples. As a result, behaviors that are similar, but not identical, are viewed as close (e.g., two registry entries with different values, random file names in the same locations). Normalization addresses the issue of differing information content. Table 5.6 shows the normalized compression distance matrix for the malware described in Table 5.5.

5.4.4 Constructing Relationships Between Malware

Once we know the information content shared between two sets of behavioral fingerprints, we can combine various pieces of malware based on their similarity. In our approach, we construct a tree structure based on the well-known hierarchical clustering algorithm [87]. In particular, we use pairwise single-linkage clustering, which defines the distance between two clusters as the minimum distance between any two members of the clusters. We output the hierarchical cluster results as a tree graph in graphviz's dot format [109]. Figure 5.2 shows the generated tree for the malware in Table 5.5.

Cluster	Elements	Overlap	Example
c1	C, D	67.86%	scans 25
c2	A, B	97.96%	installs a cygwin rootkit
c3	E, G, H	56.60%	disables antivirus
c4	F, I, J	53.59%	IRC

Table 5.7: The clusters generated via our technique for the malware listed in Table 5.5.

5.4.5 Extracting Meaningful Groups

While the tree-based output of the hierarchical clustering algorithm does show the relationships between the information content of behavioral fingerprints, it does not focus attention on areas of the tree in which the similarities (or lack thereof) indicate an important group of malware. Therefore, we need a mechanism to extract meaningful groups from the tree. A naive approach to this problem would be to set a single threshold of the differences between two nodes in the tree. However, this can be problematic as a single uniform distance does not accurately represent the distance between various subtrees. For example, consider the dendrogram in Figure 5.2. The height of many U-shaped lines connecting objects in a hierarchical tree illustrates the distance between the two objects being connected. As the figure shows, the difference between the information content of subtrees can be substantial. Therefore, we require an automated means of discovering where the most important changes occur.

To address this limitation, we adopt an "inconsistency" measure that is used to compute the difference in magnitude between distances of clusters so that the tree can be cut into distinct clusters. Clusters are constructed from the tree by first calculating the inconsistency coefficient of each cluster, and then thresholding based on the coefficient. The inconsistency coefficient characterizes each link in a cluster tree by comparing its length with the average length of other links at the same level of the hierarchy. The higher the value of this coefficient, the less similar are the objects connected by the link. The inconsistency coefficient calculation has one parameter, which is the depth below the level of the current link to consider in the calculation. All the links at the current level in the hierarchy, as well as links down to the given depth below the current level, are used in the inconsistency calculation.

In Table 5.7 we see the result of the application of this approach to the example malware in Table 5.5. The 10 unique pieces of malware generate four unique clusters. Each cluster shows the elements in that cluster, the average number of unique behaviors in common between the clusters, and an example of a high-level behavior in common between each binary in the cluster. For example, cluster one consists of C and D and represents two unique behaviors of mytob, a mass mailing scanning worm. Five of the behaviors observed for C and D are identical (e.g., scans port 25), but several others exhibit some behavioral polymorphism (e.g., different run on reboot registry entries). The other three clusters exhibit similar expected results, with cluster two representing the cygwin backdoors, cluster three the bancos variants, and cluster four a class of IRC backdoors.

5.5 Evaluation

To demonstrate the effectiveness of behavioral clustering, we evaluate our technique on the large and small datasets. We begin by demonstrating the runtime performance and the effects of various parameters on the system. We then show the quality or goodness of the clusters generated by our approach by comparing existing antivirus groups (e.g., those labeled as SDBot) to our clusters. Next we discuss our clusters in the context of our completeness, conciseness, and consistency criteria presented earlier. Finally, we illustrate the utility of the clusters by answering relevant questions about the malware samples.

5.5.1 Performance and Parameterization

We now examine the memory usage and execution time for the hierarchical clustering algorithm. To obtain these statistics, we take random sub-samples of length between 1 and 526 samples from the small dataset. For each sub-sample, we analyze its run time and memory consumption by running 10 trials for each. The experiments were performed on a Dell PowerEdge 4600 with two Intel Xeon MP CPUs (3.00GHz), 4 GB of DDR ECC RAM, 146G Cheetah Seagate drive with an Adaptec 3960D Ultra160 SCSI adapter, running Fedora Core Linux.

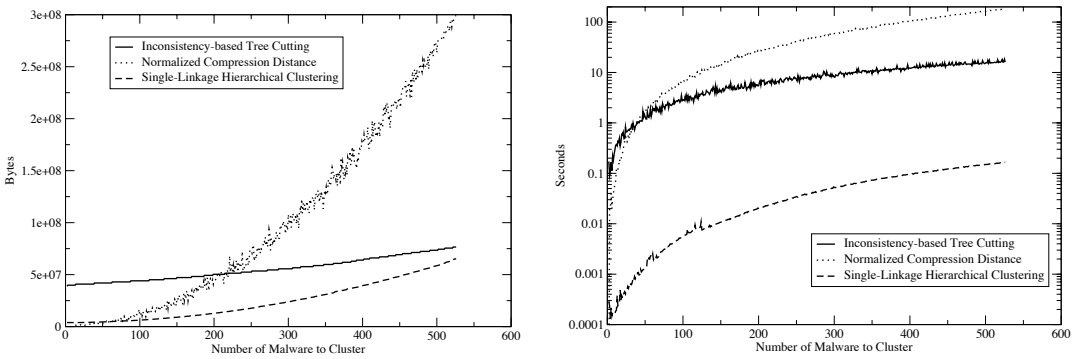


Figure 5.3: The memory and runtime required for performing clustering based on the number of malware clustered (for a variety of different-sized malware behaviors).

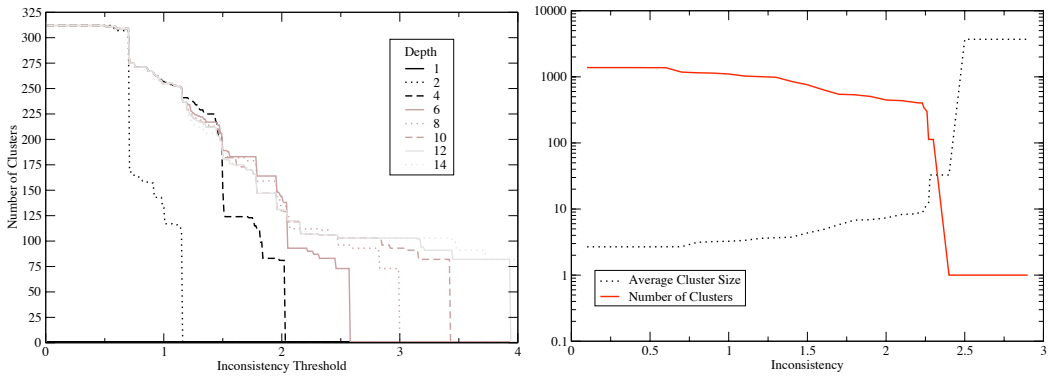


Figure 5.4: On the left, the number of clusters generated for various values of the inconsistency parameter and depth. On the right, the trade-off between the number of clusters, the average cluster size, and the inconsistency value.

We first decompose the entire execution process into five logical steps: (1) trace collection, (2) state change extraction, (3) NCD distance matrix computation: an $O(N^2)$ operation, (4) clustering the distance matrix into a tree, (5) cutting the tree into clusters. We focus on the latter three operations specific to our algorithm for performance evaluation. Figure 5.3 shows the memory usage for those three steps. As expected, computing NCD requires the most memory with quadratic growth with an increasing number of malware for clustering. However, clustering 500 malware samples requires less than 300MB of memory. The memory usage for the other two components grows at a much slower rate. Examining the run-time in Figure 5.3 indicates that all three components can complete within hundreds of seconds for clustering several hundred malware samples.

Phases 1-4 of the system operate without any parameters. However, the tree-cutting algorithm of phase 5 has two parameters: the inconsistency measure and the depth value. Intuitively, larger inconsistency measures lead to fewer clusters and larger depth values for computing inconsistency result in more clusters. Figure 5.4 illustrates the effects of depth on the number of clusters produced for the small dataset for various inconsistency values. Values of between 4-6 for the depth (the 3rd and 4th colored lines) appear to bound the knee of the curve. In order to evaluate the effect of inconsistency, we fixed the depth to 4 and evaluated the number of clusters versus the average size of the clusters for various inconsistency values in the large dataset. The results of this analysis, shown in Figure 5.4, show a smooth trade-off until an inconsistency value between 2.2 and 2.3, where the clusters quickly collapse into a single cluster. In order to generate clusters that are as concise as possible without losing important feature information, the experiments in the next selection utilize values of depth and inconsistency just at the knee of these curves. In this case, it is a depth value of 4 and an inconsistency value of 2.22.

5.5.2 Comparing Antivirus Groupings and Behavioral Clustering

To evaluate its effectiveness, we applied our behavioral clustering algorithm on our large dataset. Our algorithm created 403 clusters from the 3,698 individual pieces of malware using the parameters discussed above.

As a first approximation of the quality of the clusters produced, we returned to our example in Section 2 and evaluated the clustering of various malware samples labeled as SDBot by the antivirus products. Recall from our previous discussions that various antivirus products take differing approaches to labeling malware. Some adopt a general approach with malware falling into a few broad categories and others apply more specific, almost per sample, labels to each binary. We expect that a behavior-based approach would separate out these more general classes if their behavior differs, and aggregate across the more specific classes if behaviors are shared. Looking at these extremes in our sample, Symantec, who adopts a more general approach, has two binaries identified as back-door.sdbot. They were divided into separate clusters in our evaluation based on differing processes created, differ-

Antivirus	Completeness		Conciseness		Consistency
	Detected	% Detected	Unique Labels	Clusters or Families	Identical Behavior Labeled Identically
McAfee	2018	54.6%	308	84	47.2%
F-Prot	2958	80.0%	1544	194	31.1%
ClamAV	2244	60.7%	1102	119	34.9%
Trend	2960	80.0%	2034	137	44.2%
Symantec	1904	51.5%	125	107	68.2%
Behavior	3387	91.6%	403	403	100%

Table 5.8: The completeness, conciseness, and consistency of the clusters created with our algorithm on the large dataset as compared to various antivirus vendors.

ing back-door ports, differing methods of process invocation or reboot, and the presence of antivirus avoidance in one of the samples. On the other extreme, FProt, which has a high propensity to label each malware sample individually, had 47 samples that were identified as belonging to the sdbot family. FProt provided 46 unique labels for these samples, nearly one unique label per sample. In our clustering, these 46 unique labels were collapsed into 15 unique clusters reflecting their overlap in behaviors. As we noted in Section 2, these groupings have differing semantics: both Symantec labels were also labeled by FProt as SDBot, but obviously not all FProt labels were identified as SDBot by Symantec. Both of these extremes demonstrate the utility of our approach in moving toward a labeling scheme that is more concise, complete, and consistent.

5.5.3 Measuring Completeness, Conciseness, and Consistency

We previously examined how the clusters resulting from the application of our algorithm to the large dataset compared to classification of antivirus products. In this section, we examine more general characteristics of our clusters in an effort to demonstrate their quality. In particular, we demonstrate the completeness, conciseness, and consistency of the generated clusters. Our analysis of these properties, summarized in Table 5.8, are highlighted each in turn:

5.5.3.1 Completeness

To measure completeness, we examined the number of times we created a meaningful label for a binary and compared this to the detection rates of the various antivirus products. For antivirus software, "not detected" means no signature matched, despite the up-to-date signature information. For behavioral clustering, "not detected" means that we identified no behavior. A unique aspect of this system is that our limiting factor is not whether we have seen a particular binary before, as in a signature system, but whether we are able to extract meaningful behavior. Any such behavior can be clustered in the context of any number of previously observed malware instances and differentiated, although this differentiation is clearly more valuable as more instances are observed. In our experiments, roughly 311 binaries exhibited no behavior. The root cause of these errors, and a more complete discussion of the general limitations of dynamic path exploration, is available in the Limitations section. A striking observation from the table is that many antivirus products provide detection rates as low as 51%, compared to around 91% using behavioral clustering. It should be noted that these numbers are as much an indication of the mechanisms the vendors use to collect malware as the antivirus products themselves, since signature systems can clearly only detect what they have seen before. While it would be unfair to judge the detection rates based on previously unseen malware, we hesitate to point out that our approach for collection of these binaries is not unique. In fact, while individual antivirus product rates may vary, over 96 percent of the malware samples were detected by at least one of the antivirus products. These samples are seen significantly more broadly than our single collection infrastructure and many antivirus products fail to detect them.

5.5.3.2 Conciseness

Conciseness represented the ability of the labeling system to group similar items into groups that both reflected the important differences in samples, but were devoid of superfluous labels. As in Section 2, we evaluate conciseness by examining the characteristics of the grouping, or clusters, created by antivirus products with those created by our approach. We examine the number of unique labels generated by the antivirus products and

a heuristically-generated notion of families or groups of these labels extracted from the human readable strings. For example, the labels W32-Sdbot.AC and Sdbot.42, are both in the "sdbot" family. As we noted before, antivirus products vary widely in how concisely they represent malware. Relative to other systems, our clusters strike a middle ground in conciseness, providing fewer labels than the total unique labels of antivirus products, but more than the number of antivirus product families. This observation is consistent with the previous section in that the antivirus product families exhibit multiple different behaviors, but these behaviors have much in common across individual labels.

5.5.3.3 Consistency

Consistency referred to the ability of a labeling system to identify similar or identical items in the same way. In the context of our behavioral system goals, this implies that identical behaviors are placed in the same clusters. In order to measure the consistency of the system, we examined the binaries that exhibited exactly identical behavior. In the large sample, roughly 2,200 binaries shared exactly identical behavior with another sample. When grouped, these 2,200 binaries created 267 groups in which each sample in the group had exactly the same behavior. We compared the percentage of time the clusters were identified as the same through our approach, as well as the various antivirus products. As expected, our system placed all of the identical behaviors in the same clusters. However, because consistency is a design goal of the system, the consistency value for our technique is more a measure of correctness than quality. What is interesting to note, however, is that antivirus products obviously do not share this same goal. Antivirus products only labeled exactly identical behavior with the same label roughly 31% to 68% percent of the time.

5.5.4 Application of Clustering and Behavior Signatures

In this subsection we look at several applications of this technique, in the context of the clusters, created by our algorithm from the large dataset.

Network	Process	Files	Registry
80/tcp	execs cmd.exe	writes winhlp32.dat	uses wininet.dll
25/tcp	execs iexplore.exe	writes tasklist32.exe	uses PRNG
6667/tcp	execs regedit.exe	writes change.log	modifies registered applications
587/tcp	execs tasklist32.exe	writes mirc.ini	modifies proxy settings
80/tcp scan	execs svchost.exe	writes svchost.exe	modifies mounted drives

Table 5.9: The top five malware behaviors observed by type.

5.5.4.1 Classifying Emerging Threats

Behavioral classification can be effective in characterizing emerging threats not yet known or not detected by antivirus signatures. For example, cluster c156 consists of three malware samples that exhibit malicious bot-related behavior, including IRC command and control activities. Each of the 75 behaviors observed in the cluster is shared with other samples of the group (96.92% on average), meaning the malware samples within the cluster have almost identical behavior. However, none of the antivirus vendors detect the samples in this cluster except for F-Prot, which only detects one of the samples. It is clear that our behavioral classification would assist in identifying these samples as emerging threats through their extensive malicious behavioral profile.

5.5.4.2 Resisting Binary Polymorphism

Similarly, behavioral classification can also assist in grouping an undetected outlier sample (due to polymorphism or some other deficiency in the antivirus signatures) together with a common family that it shares significant behaviors with. For example, cluster c80 consists of three samples that share identical behaviors with distinctive strings “bling.exe” and “m0rgan.org”. The samples in this cluster are consistently labeled as a malicious bot across the antivirus vendors except Symantec, which fails to identify one of the samples. To maintain completeness, this outlier sample should be labeled similarly to the other samples based on its behavioral profile.

5.5.4.3 Examining Malware Behaviors

Clearly one of the values of any type of automated security system is not to simply provide detailed information on individual malware, but also to provide broad analysis on future directions of malware. Using the behavioral signatures created by our system, we extracted the most prevalent behaviors for each of the various categories of behaviors we monitor. The top five such behaviors in each category are shown in Table 5.9.

The network behavior seems to conform with agreed notions of how the tasks are being performed by most malware today. Two of the top five network behaviors involve the use of mail ports, presumably for spam. Port 6667 is a common IRC port and is often used for remote control of the malware. Two of the ports are HTTP ports used by systems to check for jailed environments, download code via the web, or tunnel command and control over what is often an unfiltered port. The process behaviors are interesting in that many process executables are named like common Windows utilities to avoid arousing suspicion (e.g., svchost.exe, tasklist32.exe). In addition, some malware uses iexplore.exe directly to launch popup ads and redirect users to potential phishing sites. This use of existing programs and libraries will make simple anomaly detection techniques more difficult. The file writes show common executable names and data files written to the filesystem by malware. For example, the winhlp32.dat file is a data file common to many Bancos trojans. Registry keys are also fairly interesting indications of behavior and the prevalence of wininet.dll keys shows heavy use of existing libraries for network support. The writing to PRNG keys indicates a heavy use of randomization, as the seed is updated every time a PRNG-related function is used. As expected, the malware does examine and modify the registered application on a machine, the TCP/IP proxy settings (in part to avoid antivirus), and it queries mounted drives.

5.6 Related Work

Our work is the first to apply automated clustering to understand malware behavior using resulting state changes on the host to identify various malware families. Related work in malware collection, analysis, and signature generation has primarily explored static and

byte-level signatures [138, 117]. Content-based signatures are insufficient to cope with emerging threats due to intentional evasion. Behavioral analysis has been proposed as a solution to deal with polymorphism and metamorphism, where malware changes its visible instruction sequence (typically the decryptor routine) as it spreads. Similar to our work, emulating malware to discover spyware behavior by using anti-spyware tools has been used in measurements studies [132].

There are several abstraction layers at which behavioral profiles can be created. Previous work has focused on lower layers, such as individual system calls [114, 4], instruction-based code templates [45], the initial code run on malware infection (shellcode) [121], and network connection and session behavior [187, 70]. Such behavior needs to be effectively elicited. In our work, we chose a higher abstraction layer for several reasons. In considering the actions of malware, it is not the individual system calls that define the significant actions that a piece of malware inflicts upon the infected host; rather, it is the resulting changes in the state of the host. Also, although lower levels may allow signatures that differentiate malware, they do not provide semantic value in explaining behaviors exhibited by a malware variant or family. In our work, we define malware by what it actually does, and thereby build in more semantic meanings to the profiles and clusters generated.

Recently, Kolter and Maloof [108] studied applying machine learning to classify malicious executables using n-grams of byte codes. Our use of hierarchical clustering based on normalized compression distance is a first step at examining how statistical techniques are useful in classifying malware, but the features used are the resulting state changes on the host to be more resistant to evasion and inaccuracies. Normalized information distance was proposed by Li *et al.* [115] as an optimal similarity metric to approximate all other effective similarity metrics. In previous work [183], NCD was applied to worm executables directly and to the network traffic generated by worms. Our work applies NCD at a different layer of abstraction. Rather than applying NCD to the literal malware executables, we apply NCD to the malware behavior.

5.7 Summary

In this chapter, we have proposed and evaluated a cloud-centric service for the large-scale analysis and classification of malware by its runtime behavior. As the malware epidemic continues, scalable classification architectures and techniques will become even more important to understand and respond quickly to new threats.

5.7.1 Leveraging the Cloud

Throughout this chapter, we've learned that malware classification operating at a higher abstraction layer of runtime behaviors imposes non-trivial architectural and performance requirements. As we've demonstrated, cloud computing can effectively meet such requirements and provide the capabilities necessary to operate such a classification service at a large scale. In particular, the following properties of cloud computing proved to be invaluable in the design and implementation of our analysis and classification service:

- **Scalability:** In order to analyze hundreds of thousands of malware samples in instrumented virtualized environments and run computationally-intensive classification and clustering algorithms on the results requires scalability of computing resources that cloud computing can provide. While running over 100 virtual machines in parallel in a cloud computing environment is a simple endeavor, such as a task with traditional computing resources would likely be prohibitively difficult or costly.

- **Network Effect:** The more contributors and malware samples that are collected and analyzed by the classification service, the increased potential for better clustering and overall intelligence of the wide spectrum of malicious software in the wild. In addition, a cloud-centric model offers the ability to open our classification service to external researchers that may further contribute to a strong network effect.

- **Isolation:** Given that our classification approach requires the execution of malware to elicit runtime behaviors, it is vital that our system insulates itself from any malicious activity. Virtualization in cloud computing environments offers an effective

isolation mechanism to protect the cloud service and quickly snapshot and restore virtual machines to clean states.

CHAPTER 6

A Cloud-Centric Service for Robust and Resilient Threshold Signatures

Thus far, this thesis has primarily explored the detection and analysis of malicious software. We've observed that protecting endpoints from malicious threats is a continually-evolving and never-ending challenge, even with effective anti-malware cloud-centric services such as CloudAV. With services such as PolyPack demonstrating severe weaknesses in anti-malware mechanisms, it's not unreasonable to assume that a substantial portion of end hosts on the Internet are compromised with malicious software.

In this chapter, we aim to design more resilient cloud-centric services that can provide security guarantees even if the end host is indeed compromised by malicious software. At the same time, we assert that the benefits of cloud-centric architecture apply to software security services beyond those directly related to malicious software. To demonstrate these points, we design a security service for computing cryptographic signatures that may securely authenticate a user's actions even if their host is compromised. While cryptographic signature computation is a task traditionally performed in a host-centric model, we propose a cloud-centric model for signature computation using threshold cryptography.

Threshold cryptography offers attractive properties for requiring a number of independent parties to cooperate to perform a cryptographic operation, such as generating a signature across multiple devices. In this chapter, we introduce an architecture called CloudCard that employs a threshold signature scheme with key material split across a user's host, a

user's mobile device, and a cloud service. We show that CloudCard and its cloud-centric architecture enables improved secrecy of private key material, flexible and fast revocation, and out-of-band signature confirmation, all while maintaining compatibility with existing applications and interfaces.

While threshold cryptography was first developed in the 1980s [30], it wasn't until the late 1990s and early 2000s that the first practical real-world applications were developed. Providing the building blocks for improved private key secrecy and potentially eliminating the pains of credential revocation, threshold cryptography received attention from the security community [26, 27], as well as improvements from the cryptographic community [162].

Since then, interest in practical applications of threshold cryptography has been minimal. However, much has changed in the past decade with respect to the security threats facing modern hosts and users. Endpoint security in the current day is often considered laughable, with standard desktop loadsets representing such a large client-side attack surface that they're frequently assumed to be compromised. Many of the threats that threshold cryptography schemes can help protect against are even more realistic and increasingly relevant. In addition, the modern computing landscape has evolved drastically. Mobile devices have exploded in popularity and are increasingly connected and powerful. In fact, mobile devices are now frequently used to provide two-factor authentication to address the problem of weak endpoint security. Lastly, commodity public cloud infrastructure has made highly-available services easily architected and deployed.

Observing these trends over the past decade, we propose that a new generation of threshold signature schemes may be appropriate and viable. In this chapter, we propose *CloudCard*, an architecture that employs a threshold signature scheme with private key material split across three parties: a user's host, a user's mobile device, and a cloud service. For a typical signing operation, partial signatures are individually computed by each of the parties, combined together on the user's host, and then typically passed along to a service for authentication purposes.

The CloudCard approach has very tangible benefits that result in a more secure and more practically deployed model than previous approaches. The primary benefits from our

proposed CloudCard approach include:

- **Improved secrecy of private key material:** By splitting private key material between the user's host, their mobile device, and a cloud service, we limit the exposure of the private key. An attacker must now compromise all three devices involved in the threshold signing operation in order to compromise the private key.

- **Flexible and fast revocation:** As detailed in [27], threshold signature schemes offer *fast revocation*, allowing provable revocation through the simple act of destroying one portion of the private key. In addition, we claim that CloudCard also enables *flexible revocation* as the user can effectively revoke one of the two private key shares under their control (e.g., if they lose their laptop or lose their mobile device).

- **Out-of-band signature confirmation:** Having a mobile device participate in the threshold signature scheme also offers the benefit of out-of-band signature confirmation. Even if the user's host is completely compromised and the attacker attempts to initiate a signature request, the user will be prompted in real-time on their mobile device to approve or deny the signature request.

- **Compatibility with existing applications and interfaces:** Instead of introducing a new approach incompatible with existing applications and services, our CloudCard implementation leverages existing interfaces (e.g., PKCS#11 [169]) supported in a wide-range of applications (e.g., web browsers, email clients, VPN clients, etc.) to provide CloudCard functionality without any modifications to the client-side application or server-side service.

More than a theoretical advancement, we see CloudCard as a missing link in the practical application and deployment of threshold cryptography and are excited to share our implementation with the security community to protect SSH logins and any other PKCS#11-enabled applications.

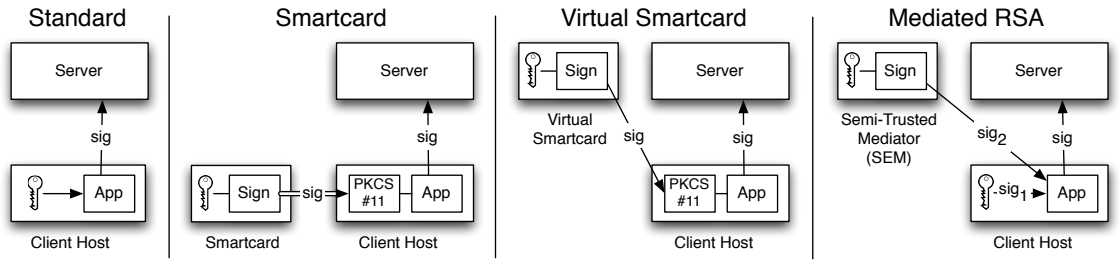


Figure 6.1: The evolution of past approaches designed to protect the secrecy of private key material.

6.1 Challenges in Cryptography and PKI

Two major challenges in cryptographic systems and public key infrastructure (PKI) deployments are maintaining key secrecy and enabling effective revocation of credentials [134, 71]. Both of these challenges have a long history of proposed solutions that have not optimally or adequately addressed the problems. In order to understand the motivation and advantages of our CloudCard approach, it is vital to understand the previous approaches to these fundamental challenges in cryptographic authentication systems.

6.1.1 Private Key Secrecy

One of the major challenges in any cryptographic system is maintaining secrecy of key material. For our purposes, we are primarily concerned with asymmetric and public-key cryptosystems [66] used for authentication, where it is imperative to maintain secrecy of the private key material. Over the years, a number of approaches have been proposed to provide better key secrecy. We illustrate several of these previous approaches in Figure 6.1 and discuss each of their advantages and disadvantages in the following section.

6.1.1.1 Standard

In the most common approach, private key material is stored on the same host that is going to be performing the signing operations [167]. For example, a user may store a private key on their filesystem or in a certificate store, which is then used for a signing operation by an application on their host.

On the positive side, storing key material directly on the host is cheap and relatively simple. On the negative side, a compromise of the host will most likely result in a compromise of the private key material as well, even if the key is encrypted on-disk with a passphrase. As the client software on a typical host presents quite a large attack surface, using this *standard* model does not provide the desired level of key secrecy for most security requirements.

6.1.1.2 Smartcard

Smartcards are dedicated hardware devices that can store private key material in tamper-resistant storage and perform specialized cryptographic operations using that key material [74]. Traditionally, a smartcard will physically interface with a computer to perform signing operations via a USB-connected smartcard reader.

On the positive side, smartcards can solve the problems presented by storing key material directly on the host in the *standard* model. Since the signing operations take place directly on the dedicated smartcard processor, even a fully-compromised host cannot extract the private key material from a connected smartcard. However, as smartcards are dedicated physical devices, they often come along with a large cost to purchase the hardware as well as the readers that must be attached to all hosts. In addition, previous studies have shown that many popular PKCS#11 smartcard implementations have suffered flaws that may inadvertently expose secret key material [29, 47, 64].

6.1.1.3 Virtual Smartcard

Virtual smartcards were introduced to address the high hardware costs of traditional smartcards, while maintaining some of their advantages in protecting private key secrecy. Virtual smartcards will commonly re-use the existing PKCS#11 interface but call out to an external server over the network instead of a physically-connected smartcard for signing operations [159].

On the positive side, virtual smartcards do eliminate the high hardware costs associated with smartcards, while maintaining some of the benefits of key secrecy. However, since

the external signing service is likely a general software platform, it commonly has a greater attack surface and may be more vulnerable to compromise than a tamper-resistant hardware smartcard. In addition, the external service is charged with storing all the key material, resulting in a single point of compromise. Nonetheless, the general approach of offloading a signing operation via PKCS#11 to an external service is very attractive.

6.1.1.4 Mediated RSA

While the virtual smartcard model offers advantages of the standard or smartcard approaches, its primary weakness is its requirement of storing all the private key material on a single host. Addressing such a weakness might seem difficult or infeasible, but fortunately cryptographers have developed a solution known as threshold cryptography [162, 35, 119, 89]. In a threshold cryptosystem, a private key may be split among n parties, each which can independently compute a partial signature with their partial key material and later combine those signatures into a full valid signature.

For example, in [27], Boneh *et al.* proposed an approach known as *mediated RSA* that employed threshold cryptography. In this model, illustrated in Figure 6.1, private key material is split between the user's host and a semi-trusted mediator (SEM) service. When the user attempts a signature operation, partial signatures are generated by both the host and the SEM independently and then combined at the host into a full valid signature. Since the split key material is never recombined, the mediated RSA scheme is resistant to compromise since an attacker must compromise both the host and the SEM to gather enough key material in order to forge signatures.

While the general approach of mediated RSA offers great security properties, one potential weakness is the fact that there is no secure out-of-band channel with which a user can confirm or deny a particular signature operation. While the approach does split the key between the user's host and the SEM, if the user's host is compromised and one portion of the key material is disclosed, an attacker is free to ask the SEM to generate the other necessary half of the signature without any sort of approval process like the traditional `SSH_ASKPASS` confirmation invoked by `ssh-agent` [148].

6.1.2 Failure of Revocation Mechanisms

As described in the previous section, keeping private key material secret is of utmost importance. However, no cryptosystem is completely bullet-proof and attackers will undoubtedly work their way through numerous layers of defense in order to compromise a user's private keys. Therefore, an effective cryptosystem should be designed to handle revocation gracefully. That is, in the event of the compromise of a user's private key material, the system should be able to revoke access to a compromised user, even if the attacker is able to forge valid signatures with the user's private key material.

Revocation techniques can vary depending on the type of cryptosystem employed. For example, if standard asymmetric cryptography is used (e.g., standard SSH pubkey authentication), revocation may be as easy as "deauthorizing" a user's public key if their private key is compromised. When a PKI model is employed, revocation becomes even more challenging. In this section, we discuss these models and the advantages and disadvantages of several approaches with respect to providing effective revocation.

6.1.2.1 SSH Revocation

Asymmetric cryptography offers clear advantages over shared secrets when used in authentication systems. In a simple case, such as SSH pubkey authentication, when a user establishes an account with a server, they provide the server their public key rather than a password. By using their private key to sign a random challenge from the server at login time, the user can prove to the server that they possess their private key without ever actually revealing it to the server.

In practice, users and organizations will often perform SSH pubkey authentication using a variant of the system described above. Consider an organization with 50 SSH servers: in lieu of heavier-weight centralized authentication systems, many such organizations will place public keys from each administrator on each server (e.g., in the `authorized_keys` file). In such a scenario, revoking a user's credentials would require removing the user's public key from each of the 50 servers. While certainly not an insurmountable task, such revocation is ineffective and incomplete if the user's public key is not removed from *every*

server that it was previously placed on.

6.1.2.2 PKI Revocation

If a user wants to prove their identity to a server with which they have no prior relationship, however, then standard asymmetric cryptography like is typically deployed for SSH logins is no longer sufficient. One model intended to address this problem is known as *public key infrastructure*. This type of system works along similar lines to identity cards in the real world. A central, trusted authority (e.g., Certificate Authority or CA) will by some means validate a user's identity, then provide an unforgeable credential to the user attesting to their identity, typically by signing a binding between the user's public key and their distinguished name in the form of a certificate. Then, the user can present the certificate to any server that trusts the CA, and the said server can authenticate the user by verifying that the certificate is genuine and asking the user to prove that they do indeed hold their private key.

Again, however, serious problems arise around revocation. If the user's private key is lost or stolen, thereby allowing someone else to impersonate the user, there are only two possible recourses: either simply wait for the certificate to expire, or the CA must somehow inform all parties relying on it about the revocation. In practice, certificate authorities employ several mechanisms to represent and distribute revocation information to ensure that a server does not accept credentials that may have been compromised and revoked [191, 184]. Two of the most popular and widely deployed mechanisms are CRLs and OCSP.

Certificate Revocation Lists (CRLs): A Certificate Revocation List (CRL), defined in RFC 3280 [76], is a list of serial numbers of revoked certificates that is signed by a CA and periodically updated and distributed to PKI-enabled servers. When a certificate needs to be revoked, the CA will add the affected certificate's serial number to the CRL. Upon validation of a user's credentials, a PKI-enabled server will check against CRL to ensure that the user's certificate has not been revoked by the CA.

While CRLs allow non-centralized distribution of revocation information, they are not without problems. For one, CRLs will continue to grow in size as more and more revoked signatures are added to the list (optionally, certificates may be removed from the CRL upon

expiration), increasing the distribution size of the CRL. In addition, as CRLs may only be updated and distributed periodically, they may not contain the most up to date revocation information, leading to potential use of revoked credentials.

Online Certificate Status Protocol (OCSP): The Online Certificate Status Protocol (OCSP), defined in RFC 2560 [75], allows a PKI-enabled server to query an OCSP server to check whether a particular certificate (based on its serial number) has been revoked. Instead of downloading a full list of revoked certificate serials, OCSP defines a request/response protocol to check for individual certification revocation status.

While OCSP was developed to address some of the distribution and liveness shortcomings of CRLs, it has serious deficiencies of its own:

- **Replay Attacks:** The OCSP protocol is vulnerable to replay attacks, where a previously captured "good" response can be replayed after a certificate has been revoked. After obtaining a user's revoked credentials and supplying them to a PKI-enabled application, an eavesdropping attacker can simply replay the "good" response when the PKI-enabled application makes an OCSP request to check the certificate's revocation status and successfully authenticate as that user. While the OCSP protocol introduced a nonce attribute to prevent replay attacks, many OCSP clients and servers do not support the nonce extension.

- **Implementation Flaws:** OCSP has suffered catastrophic implementation flaws such as the OCSP "tryLater" attack from Moxie Marlinspike. Marlinspike discovered that specifying an OCSP ResponseStatus of "tryLater" would cause most OCSP clients to accept that response without any warnings. Since the ResponseStatus structure is not included in fields signed by the CA in the response, an attacker can successfully spoof such "tryLater" responses and subvert the OCSP protocol.

It is clear from these failures that CRLs and OCSP are not sufficient revocation mechanisms for practical deployment of PKI-based authentication. The underlying danger with these revocation methods is that the full private key material still exists even after a certificate is revoked. If the revocation-checking mechanism is not bullet-proof, as the numerous

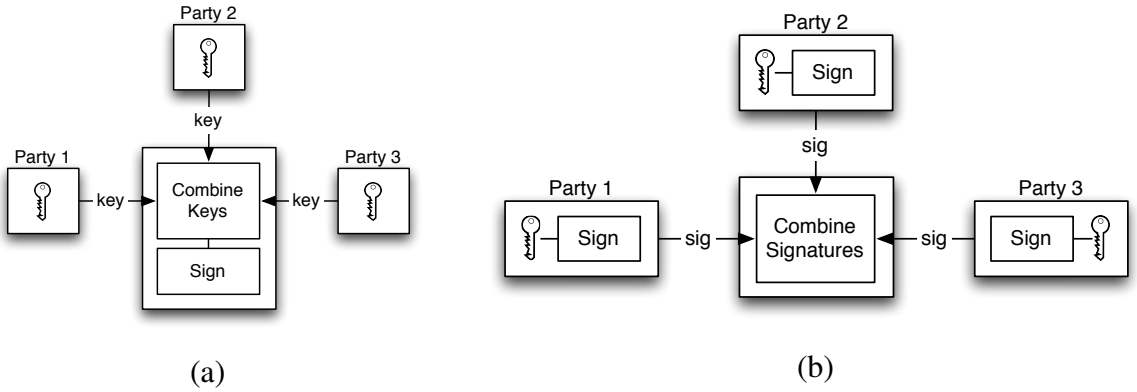


Figure 6.2: Traditional secret sharing schemes like have to combine the split key material in a single location in order to perform a cryptographic operation as illustrated in (a). Using threshold cryptography, partial signatures can be combined to generate a full signature without ever having to combine the split key material in a single location as illustated in (b).

limitations and vulnerabilities of CRLs and OCSP have shown to be true in practice, the private key associated with a revoked certificate may be re-used by an attacker and erroneously accepted by a PKI-enabled server, violating the fundamental intentions of revocation.

6.1.2.3 Fast Revocation with Mediated RSA

As mentioned previously, the downside of most revocations schemes is that they rely on broadcasting to all PKI-enabled services that a user's credentials were compromised, rather than actually being able to universally wipe those credentials from existence to ensure they're no longer usable.

When a mediated RSA scheme or other threshold signature scheme is deployed, proper revocation is near achievable [27, 26]. Since the private key material is split among the client and an SEM in mediated RSA, revocation can be effectively achieved by simply destroying one share of the private key, rendering any further valid signature computations impossible.

6.2 CloudCard Architecture

Our CloudCard architecture was designed to address the limitations of the previous approaches with respect to private key secrecy and revocation. Using the classification of previous approaches listed in Figure 6.1, CloudCard could be seen as a combination of the *Virtual Smartcard* and *Mediated RSA* approaches, gleaning the benefits of private key secrecy and revocation from these models while introducing several new benefits of its own.

In this section, we discuss the threshold signature scheme employed by CloudCard, the primary components involved in its operation, how it performs key generation and signature generation, and some of the notable features and benefits that arise from the CloudCard approach.

6.2.1 Threshold Signatures with CloudCard

A threshold signature scheme forms the basis for our CloudCard approach. Threshold signature schemes in asymmetric cryptosystems (e.g., RSA [154]) allow for the generation of multiple private key shares as opposed to a single private key. Each key share can be used to independently compute an individual signature, and all of the individual signatures can be combined together to produce a valid full signature. Such schemes offer attractive properties for requiring a number of independent parties to cooperate to perform a cryptographic operation, such as generating a signature across multiple devices.

When many hear of key splitting, they often think of secret sharing schemes where a secret key is split between n parties. Some secret sharing schemes such as Shamir's introduce a threshold where only k of n parties are necessary to reconstruct the key. However, these schemes are much different than the threshold signature schemes that CloudCard is based on. In a secret sharing scheme, when it comes time to perform some cryptographic operation, all the key material must be recombined in a single location to perform that operation, which is obviously not ideal from a security perspective. In a threshold signature scheme, the key material is never recombined in a single location. Rather, the individual signatures are computed independently by each participating party and then the signatures

(rather than the keys) are combined in a single location. This important distinction is illustrated in Figure 6.2.

In particular, CloudCard implements Shoup's RSA threshold signature scheme. As described in [162], Shoup's RSA threshold signature scheme enjoys the following properties:

- It is unforgeable and robust in the random oracle model, assuming the RSA problem is hard.
- Signature share generation and verification is completely non-interactive.
- The size of an individual signature share is bounded by a constant times the size of the RSA modulus.

The advances made in Shoup's threshold signature scheme eliminated some of the key roadblocks that had hampered practical threshold cryptography. For example, previous threshold RSA approaches were interactive, requiring a synchronous network with broadcast. In addition, the simplicity of Shoup's scheme makes it attractive from an implementation point of view. Most importantly, the resulting signature computed in Shoup's scheme can be verified with a normal RSA operation. Therefore, a client who has generated a signature using CloudCard, can pass that signature to a server who can verify it with a completely vanilla RSA verification routine. This is a key detail as it means the verifying component (e.g., an SSH server) does not require any modification and can be completely oblivious to the CloudCard scheme.

6.2.2 CloudCard Operation

Any threshold signature scheme has several components that are tasked with various functions during key generation and signature computation. Such components include:

- **Dealer:** The dealer is the component that is tasked with initially generating the key shares and distributing them as necessary to the participating signers.
- **Signer:** A signer is a party that holds one of the key shares and uses it to compute individual signatures.

- **Combiner:** The combiner is tasked with collecting the individual signatures generated by each signer and combining them into a full complete signature.

In our CloudCard system, we employ three signers and require individual signatures from all of the signers to combine into the full signature. Therefore, we say that CloudCard uses a (3,3) threshold RSA (TC-RSA) scheme. The three components are illustrated in Figure 6.3 and are described as follows:

- **User's Host:** The user's host acts as the dealer during key generation, a signer for one of three key shares, and the combiner for all the individual signature. The PKCS#11 module implements the signer and combiner functionality, while a simple command line tool implements the dealer functionality.

- **User's Mobile Device:** The user's mobile device acts as a signer for one of the three key shares. In addition, the mobile device implements signature confirmation functionality, which allows the user to approve or deny a signature request interactively on their mobile device.

- **Cloud Service:** The cloud service acts as a signer for one of the three key shares. In addition, the cloud service is tasked with acting as a communication broker between the user's host and the mobile device, shuttling signature requests and the resulting individual signatures back and forth between those components.

During key generation, the user's host acts as a dealer, generating the three private key shares and the corresponding public key. The three key shares are distributed as follows: one key share is stored locally on the filesystem of the user's host; one key share is distributed directly to the mobile device (e.g., via a QR code); and one key share is sent off to the cloud service.

During a signature operation, the user's host will generate its individual signature using its locally-stored key share and will notify the cloud service about the signing request. The cloud service will generate its individual signature and notify the user's mobile device about the signing request (preferably via a real-time push notification service such as Google's C2DM or Apple's APNS). The user's mobile device will then prompt the user to confirm

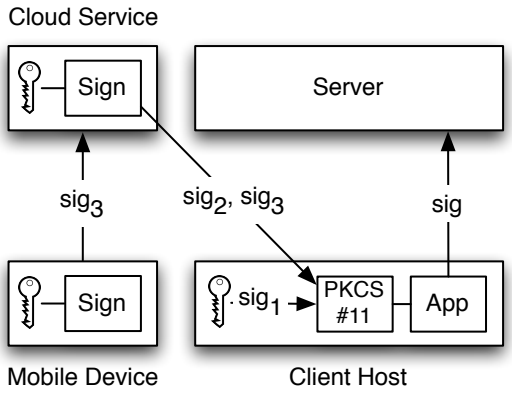


Figure 6.3: Our proposed CloudCard architecture uses (3,3) threshold RSA to generate a signature across a client's host, a cloud service, and a mobile device.

the signing request and, upon confirmation, generate and send the individual signature back to the cloud service. The cloud service would then return the individual signatures back to the user's host, which would combine all three individual signatures into a full valid signature. These described steps for signature generation and the communication between the components are illustrated in Figure 6.3.

6.2.3 Notable Features of CloudCard

CloudCard offers a number of novel features and properties that make it an attractive model for performing any sort of asymmetric cryptography-based authentication.

6.2.3.1 Private Key Secrecy

By splitting private key material between the user's host, their mobile device, and a cloud service, the CloudCard approach can greatly increase private key secrecy. An attacker must now compromise all three devices involved in the threshold signing operation in order to compromise the private key.

6.2.3.2 Flexible and Fast Revocation

By employing a threshold signature scheme, CloudCard allows for provable, fast revocation through the simple act of destroying one portion of the private key. In addition,

CloudCard also enables flexible revocation as the user can effectively revoke one of the two private key shares under their control. For example, if the user believes their host has been compromised, they can destroy the key share on their mobile device to ensure safety. And vice versa, if a user loses their mobile device, they can simply destroy the key share on their host. The key share stored in the cloud service may optionally be destroyable by a third-party such as the user's employer if deployed in an enterprise environment offering even more flexibility in revocation.

6.2.3.3 Out-of-Band Signature Confirmation

Having a mobile device participate in the threshold signature scheme also offers the benefit of out-of-band signature confirmation. Even if the user's host is completely compromised and the attacker attempts to initiate a signature request, the user will be prompted in real-time on their mobile device to approve or deny the signature request. Such functionality is analogous to the traditional `SSH_ASKPASS` confirmation invoked by `ssh-agent`, but presented to the user securely on their mobile device instead of on their host.

6.2.3.4 Compatibility via PKCS#11

A number of common client applications (e.g., web browsers, email clients, VPN clients, SSH client, etc.) offer a point of integration via a standard interface known as PKCS#11 (i.e. Public-Key Cryptography Standard #11). This interface is designed primarily for use with hardware tokens or smartcards. While PKCS#11 is intended primarily for use with hardware tokens, what it actually offers is a generic mechanism for offloading signature operations to custom code modules. As such, we can implement a PKCS#11 module which performs our CloudCard threshold signature scheme.

Leveraging this PKCS#11 interface, CloudCard can gain instant compatibility with a wide range of client applications. Therefore, CloudCard offers a notable model for practical deployments: no changes to client-side application or server-side services are necessary to use CloudCard.

6.3 Integration and Implementation

To put our CloudCard architecture into practice for real-world use and evaluation, we implemented CloudCard for SSH client login via pubkey authentication. While our approach doesn't require any modifications to the client (e.g., OpenSSH's `ssh` client) or the server (OpenSSH's `sshd` server), it does require implementation of a PKCS#11 interface module, a cloud service, a mobile application, and the actual threshold RSA library that each of the components uses. We describe each of these primary components and their implementations in this section.

6.3.1 TC-RSA Library

At the core of our implementation is the TC-RSA library, which is a quite literal translation of Shoup's paper into a C library [23]. The TC-RSA is present in all the components of our system: it is used by the cloud service and mobile application for individual signature generation and on the user's host to implement dealer functionality, key share generation, individual signature generation, and signature combination.

The TC-RSA library exposes a simple API: `TC_generate` to generate a dealer and key shares; `genIndSig` to generate individual signatures; and `TC_Combine_Sigs` to combine individual signatures into a composite signature. The entire TC-RSA library is only 1200 lines of C code and makes heavy use of OpenSSL's well-tested BIGNUM routines. While the C-based library provides performance characteristics important for a cryptographic component, it also is simple enough for easy auditing.

6.3.2 PKCS#11 Module

Integrating via the PKCS#11 interface [169] allows for compatibility with a wide range of applications on the user's host (such as web browsers, email clients, and VPN clients) without making any modifications to those applications. For example, a PKCS#11 provider module can be loaded into the Firefox web browser in the security preferences dialog and in the OpenVPN client via the `pkcs11-providers` configuration option.

For our target implementation of SSH client authentication, the OpenSSH client can simply be invoked with the `-I` option on the command line to specify the path to the shared library implementing the desired PKCS#11 provider. For example, `ssh -I cloudcard.so user@host` will cause the SSH client to invoke our PKCS#11 module for any RSA signing operations.

Despite PKCS#11 being a fairly complicated interface, there are only a handful of key functions that are necessary to implement for our purposes. Our PKCS#11 primarily acts as a simple shim between the TC-RSA routines and the integrating application (the OpenSSH client in our case), implementing minimal session support (e.g., `C_OpenSession`, `C_CloseSession`), basic slot and token management (e.g., `C_GetInfo`, `C_GetTokenInfo`, `C_GetMechanismInfo`), and core object and signing operations (e.g., `C_GetAttributeValue`, `C_FindObjects`, `C_Sign`).

The meat of our PKCS#11 module weighs in at under 1,000 lines of C code, with a large portion of that code from simply stubbing out all the unimplemented and unnecessary PKCS#11 interface functions (e.g., with `CKR_FUNCTION_NOT_SUPPORTED`). As this module represents the primary software customization to the user's host, we've strived to keep this component as lightweight and simple as possible.

6.3.3 ssh-keygen-tcrsa

Before a user can start using CloudCard for their SSH logins, they need to generate a public key and set of private key shares. In addition, the generated shares must also be distributed to the cloud service and the mobile device. Both the generation and distribution tasks are handled in our implementation by the `ssh-keygen-tcrsa` tool. Like the standard `ssh-keygen` tool, our `ssh-keygen-tcrsa` will generate new SSH keys on behalf of the user. Using the TC-RSA library, the tool generates the public and private key shares for the desired RSA key length.

The public key is output to the user's filesystem in `~/.ssh/id_tcrsa.pub`, so that they can later include that pubkey in the `authorized_keys` file of any SSH servers that they want to access using CloudCard.

Since our implementation uses a (3,3) threshold scheme, our TC-RSA dealer will generate three private key shares. One of the private key shares is stored locally on the user's filesystem in `~/.ssh/id_tcrsa`. Another private key share is transmitted by the `ssh-keygen-tcrsa` tool securely over the network to the cloud service. The last private key share is distributed to the mobile device. Instead of transmitting the mobile key share to the mobile device through some third-party (e.g., through the cloud service or through a SMS provider), we directly communicate the key share from the user's host to the mobile device via a QR code that is output on the screen of the user's host and scanned by our mobile application on the mobile device.

6.3.4 Cloud Service

While it may seem counter-intuitive, the cloud service is actually one of the most simple components in our implementation. While the cloud service does participate in the signing operation using one of the three private key shares, it primarily acts as a simple transport to pass signature requests from the PKCS#11 module to the user's mobile device and pass back the computed individual signatures.

The cloud service is implemented as a simple Python web service that presents a REST API for communication with the PKCS#11 module and the mobile applications installed on the user's mobile device.

6.3.5 Mobile Application

In order for the user's mobile device to participate in the CloudCard architecture, we implemented a mobile application on the Android platform for the user to install. Although the app is implemented in Java, it calls out to our existing C-based TC-RSA library via JNI for its signing operations.

At the time of key generation on the user's host, the Android app is seeded with its private key share by scanning a QR code that is encoded with necessary key material. In addition, the seeded mobile app registers itself with the cloud service so that the cloud service will know how to reach the mobile device when a signing operation is requested.

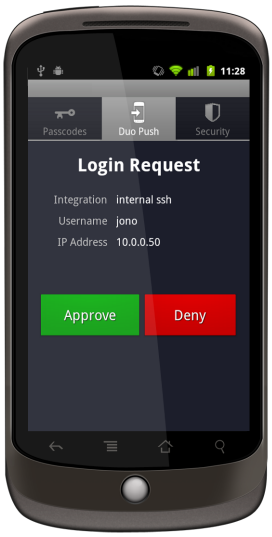


Figure 6.4: A screenshot of the CloudCard mobile application displaying a signature confirmation for approval.

Upon a signature request, the cloud service will ping the mobile app using Android's C2DM (Cloud to Device Messaging) service, which allows near real-time push notifications to be sent to the user's mobile device. Upon receiving a push notification, the mobile application prompts the user to confirm the signature request simply by tapping approve or deny on their mobile device. The PKCS#11 module can pass along contextual information such as the executed command line (e.g., `ssh -I cloudcard.so user@host`) so that the end user can see on their mobile device exactly which signing operation they are approving. This signature confirmation screen, seen in Figure 6.4, is analogous to the traditional `SSH_ASKPASS` confirmation invoked by `ssh-agent`.

6.3.6 Other Applications and Integrations

Integrating via the PKCS#11 interface is a huge win for instant compatibility with a large number of existing applications. While our particular implementation was centered around SSH client authentication, the CloudCard PKCS#11 module can be used for cryptographic signature operations in any range of applications including browsers, email clients, or VPN clients.

In addition to the PKCS#11 interface, other platform-specific interfaces exist that may

be useful to integrate with in future work. For example, CloudCard could easily be implemented as a Cryptographic Service Provider (CSP) as part of the Windows CryptoAPI. Similarly, on OS X, CloudCard could integrate via the native Common Data Security Architecture (CDSA) and Keychain APIs.

6.4 Deployment and Evaluation

To evaluate our CloudCard approach and implementation, we deployed our system in a real-world setting. As opposed to many experimental setups that may be deployed simply as proof of concepts to gather research data, the authors use the deployed CloudCard system on a daily basis to protect production servers and infrastructure.

For the experiments in this section, the following components were deployed and evaluated: the user's host is a modest Intel desktop connected to the Internet via a typical consumer broadband connection; the user's mobile phone is a Nexus One, running the latest version of Google's Android platform; the cloud service is a m1.small AWS EC2 instance hosted in the US-EAST-1 region; and the target service is a OpenSSH server hosted on a separate instance also in AWS EC2.

6.4.1 Key Generation

In a normal RSA scheme, no restrictions are placed on the primes p and q other than that they be sufficiently large [170, 98]. However, in our TC-RSA scheme, there is a requirement that p and q must be strong primes. Due to this requirement, the generation of the public key and the secret key shares in TC-RSA is more computationally expensive.

To measure the performance impact, we benchmarked key generation on a variety of RSA schemes, key sizes, and CPU types. In particular, we compared key generation of normal RSA, TC-RSA (2,2), and TC-RSA (3,3) with key sizes of 1024, 2048, 3072, and 4096 bits, running on both a modest desktop CPU (2.4 GHz Intel Core 2) and a mobile device CPU (1.0 GHz Qualcomm QSD 8250 Snapdragon ARM).

The results of this key generation performance evaluation are listed in Table 6.1. As

RSA Scheme	Key Size	Mobile CPU	Desktop CPU
Normal RSA	1024	0.841s	0.054s
	2048	5.71s	0.316s
	3072	9.66s	0.681s
	4096	27.0s	2.20s
TC-RSA (2,2)	1024	31.3s	1.86s
	2048	373s	25.1s
	3072	1621s	133s
	4096	4432s	376s
TC-RSA (3,3)	1024	21.6s	1.92s
	2048	295s	27.4s
	3072	1731s	141s
	4096	4939s	445s

Table 6.1: Timings of key generation across RSA schemes, key sizes, and CPU types.

seen in the results, key generation times for the TC-RSA schemes are consistently about two orders of magnitude greater than normal RSA on the desktop CPU. However, while TC-RSA key generation takes significantly longer than normal RSA, we observe that the required time is completely reasonable for real-world applications. As current recommendations on RSA key sizes cite 2048-bit RSA keys as being sufficient through 2030 and 3072-bit keys exceeding well beyond that [103], we believe that key generation time in the ballpark of a few minutes is acceptable.

In addition, we observe that key generation on the mobile CPU is at least an order of magnitude greater than the desktop CPU. While the dealer and its key generation does not take place on the mobile device in our current scheme, it's worth noting these performance numbers as mobile-generated key shares may be a viable, albeit computationally expensive, model.

6.4.2 Signature Generation

In a TC-RSA scheme, individual signatures are generated by each signer, then combined into one composite signature by the combiner. To evaluate the performance of our proposed scheme, we benchmark the individual signature generation on the same key sizes and devices as we did with our key generation benchmarks. However, in this experiment

RSA Scheme	Key Size	Mobile CPU	Desktop CPU
Normal RSA	1024	6.09ms	0.664ms
	2048	39.4ms	4.08ms
	3072	123ms	12.6ms
	4096	282ms	28.5ms
TC-RSA	1024	19.2ms	1.93ms
	2048	137ms	12.5ms
	3072	396ms	40.1ms
	4096	901ms	92.5ms

Table 6.2: Timings of signature generation across RSA schemes, key sizes (in bits), and CPU types.

there is no need to differentiate between the (2,2) and (3,3) TC-RSA schemes since we're only measuring the timing of a single individual signature operation and not the timing all the signatures and their combination.

The results of this signature generation performance evaluation are listed in Table 6.2. Across key sizes and CPUs, the signature generation timing is consistently a small constant factor greater for the TC-RSA scheme compared to normal RSA. Given that three such individual signatures are generated for each operation in our TC-RSA (3,3) scheme, it is vital that the TC-RSA operations are not significantly more computationally expensive than normal RSA.

As observed in our results on key generation, signature generation on the mobile CPU is, similarly, an order of magnitude slower than the desktop CPU. Given that the mobile device does compute a partial individual signature in our CloudCard approach, it is important that this mobile signature generation is not prohibitively expensive. Thankfully, even in the most expensive case with a 4096-bit key size, mobile signature generation is a sub-second operation.

6.4.3 Signature Verification

Since the verifying service (which is the OpenSSH daemon in our deployment) performs the same operation to verify the generated signature, there is no significant measurable difference in the performance of signature verification between the various RSA

schemes.

6.4.4 End-to-End Performance

In a real-world deployment, the user-perceived delays involved when using CloudCard may exceed the importance of the individual cryptographic operations. After all, if the user has to wait an extended amount of time to log in to the target SSH server when using CloudCard, it may cause unnecessary frustration to the user and deter them from using such a system, depriving them of the potential security benefits.

To measure end-to-end performance, we consider the individual signature operations, the combination of individual signatures on the host, and the delays associated with all the network communication between the host, cloud, and mobile components in the CloudCard system. We conducted our measurements with the mobile device on WiFi, instantly responding to signature requests upon receiving them to remove any user-dependent latencies from the calculation (e.g., waiting for the user to tap the approve button on their mobile device). Also, for simplicity when calculating the signature generation time, we assume the cloud's CPU is equivalent to the host's CPU.

The results for our end-to-end timing for a full SSH login are listed in Table 6.3. The total time in the far right column of the table represents the time between when our PKCS#11 module first receives the signature request from the SSH client and when all of the partial signatures have been combined back together into a composite signature returned by the PKCS#11 module to the SSH client.

The results show that there is non-trivial computation involved across the participating devices in the threshold scheme. For example, all the cryptographic signing and combining operations requires approximately 50ms for 1024-bit key size, approximately 250ms for 2048-bit key size, approximately 750ms for 3072-bit key size, and approximately 1500ms for 4096-bit key size. It's worth noting that these signature operations would take place in parallel by the various participating devices rather than serially, so the 1500ms worth of cryptographic operations would not necessarily take 1500ms of wall time.

However, the total time is clearly dominated by communication overhead caused by

network latencies between the host, cloud, and mobile components in the CloudCard system. We consider this a positive result, as the chatty HTTPS communication between components in our current implementation is hardly optimized for latency, meaning there is significant room to improve the total end-to-end performance of the CloudCard implementation.

6.4.5 Revocation

For completeness, we also performed a simple demonstration of the revocation capabilities of our CloudCard approach by destroying the key share stored on the mobile device. Without the appropriate key material, the mobile device returns a dummy signature to the cloud service, causing the user's signature to be correctly rejected.

6.4.6 Limitations

While our CloudCard approach has significant advantages compared to other approaches, it is rare for such benefits to come without trade-offs and limitations. We view the current limitations of our system as motivation for incremental improvements rather than as fundamental weaknesses of the approach.

6.4.6.1 Availability of Signers

The most obvious trade-off of having multiple networked devices participating in a distributed system employing threshold cryptography is availability. That is, if a TC-RSA (n,n) scheme is deployed that requires all signers to be available, an unavailable signer will result in the system being unable to generate a valid combined signature. For example with our implementation, if the user's mobile device does not have data service available (via the cellular data network or a WiFi network), the user will not be able to log in via SSH.

The most problematic component with respect to availability is undoubtedly the mobile device. While mobile devices are becoming increasingly-connected over global data networks, availability and service coverage is certainly a concern. However, it may be possible for the host to securely communicate with the mobile device over mechanisms other than

a wide-area network. For example, a host could request signatures from the mobile device using a wireless network, Bluetooth, USB, or any similar local communication mechanism.

Alternately, backup key shares could be generated and stored securely offline in case of any sort of extended availability outages or other such extenuating circumstances. For example, a TC-RSA (3,4) scheme could be employed to generate four key shares, distribute the usual three to the host, cloud, and mobile, and store the extra fourth share offline on a USB stick. Obviously, the use of additional backup key shares has an impact on the security and revocation capabilities of the system.

In summary, while availability is certainly a valid concern, we believe the security advantages of the CloudCard approach outweigh the availability disadvantages.

6.4.6.2 Trustworthiness of Dealer

In our current scheme, the dealer generates the public key and private key shares on the user's host machine. If the host was compromised during the key generation phase, the key shares could obviously be stolen by a malicious party, compromising the integrity of the system. However, the same type of threat affects normal key generation on the host, so in this scenario CloudCard is no worse than the normal RSA model. This type of *Trust On First Use* (TOFU) model for bootstrapping trust has been shown to be acceptable and survivable under most practical threat models (including SSH's own host authentication mechanism).

Distributed key generation [28, 123, 62] is another area of research that may provide fruitful improvements. Instead of generating all the key shares on a single device which may represent a single point of compromise, several parties may be involved in the key generation protocol without any one of them possessing enough key material to forge a valid signature.

6.5 Related Work

Shoup's research on practical threshold signatures is the most relevant work [162] to ours as it acts as the reference for our threshold RSA (TC-RSA) implementation. How-

Key Size	Signature Generation	Signature Combination	Communication Overhead	Total Time
1024	1.93 + 1.93 + 19.2 = 23.1ms	16.2ms	865ms	904ms
2048	12.5 + 12.5 + 137 = 162ms	98.1ms	865ms	1125ms
3072	40.1 + 40.1 + 396 = 476ms	297ms	865ms	1638ms
4096	92.5 + 92.5 + 901 = 1086ms	670ms	865ms	2631ms

Table 6.3: Combined TC-RSA (3,3) timings for signature generation, signature combination, and the network communication overhead for a full CloudCard SSH login sequence in the pessimistic model of all operations occurring serially.

ever, while Shoup's threshold signature scheme introduced useful properties for a practical realization, prior research laid the groundwork for threshold cryptography both for RSA [35, 119, 89] and DSA [122, 81] algorithms. In addition, the benefits of *fast revocation* through the use of threshold cryptography was first pioneered by Boneh *et al.* [27, 26].

In general, most research in threshold cryptography has focused on expanding the state of the art from a mathematical perspective rather than one of practical application. Many researchers improved upon existing threshold cryptography schemes by strengthening assumptions of an adversary and tackling more challenging and exotic threat models [120, 99, 2, 43].

However, some practical applications of threshold cryptography have been explored in the past. For example, in [185], Wu *et al.* used threshold signatures to build an intrusion-tolerant web server and CA. Threshold cryptography has also been applied to distributed storage systems [153], email signing [67], and various Java frameworks [91, 33].

As mentioned in the evaluation section, related research has been performed in the area of distributed key generation which can be valuable for threshold cryptography schemes where a centralized dealer may not be appropriate [28, 123, 62].

6.6 Summary

In this chapter, we have proposed a novel approach for employing threshold signatures known as CloudCard that addresses many of the limitations of a host-centric approach for cryptographic signatures and authentication. Through a unique architecture focused around the use of a mobile device in conjunction with a cloud-hosted service, CloudCard enables enhanced security, revocation, and usability, all while maintaining compatibility with ex-

isting PKCS#11-enabled applications and systems. Using a real-world implementation and deployment of CloudCard, we demonstrate its utility when applied to traditional SSH logins.

6.6.1 Leveraging the Cloud

Throughout this chapter, we've shown that CloudCard can provide robust security guarantees even in a threat model where we assume that the user's endpoint is compromised, which contrasts with the properties of traditional host-centric signature generation. In addition, we validated that the fundamental assumptions of this thesis around the benefits of cloud-centric security services are applicable to security realms beyond malicious software. In particular, the following properties of cloud computing proved invaluable in the design and implementation of the CloudCard service:

- **Availability:** High availability is an absolute necessity when considering an approach such as CloudCard. Cloud computing offers the ability to construct and maintain a highly-available cloud service that is a core participant in the threshold signature operation.

- **Host-proofing:** CloudCard's use of threshold cryptography demonstrates that cloud-hosted security services can operate securely even in the face of a compromised or malicious IaaS provider.

- **Empowerment:** Cloud computing empowers organizations and even individuals to spin up their own private CloudCard infrastructure with little difficulty, which would likely be infeasible in a traditional computing environment.

CHAPTER 7

Discussion and Conclusion

This thesis has demonstrated how leveraging the cloud can be used to improve software security services. While malicious threats have evolved considerably over the past decade, the emergence of cloud computing has presented a new perspective for addressing such threats. This thesis focuses on how cloud computing enables novel cloud-centric architectures that can provide improved efficacy against modern security threats. Through its five primary chapters, this thesis demonstrates how a cloud-centric model applies across a wide range of security applications such as detecting malicious software, protecting resource-constrained mobile devices, understanding the capabilities of malicious attackers, analyzing and classifying malware, and generating robust and resilient cryptographic signatures.

7.1 Summary of Contributions

This thesis began with a new architecture for deploying malware detection functionality. Our approach, CloudAV, ditches the host-centric model employed by traditional antivirus and instead advocates for a cloud-centric model where the actual detection engines are hosted in a cloud service. By maintaining a lightweight host agent to submit suspicious files to the cloud service, CloudAV achieves better detection of malicious software, enhanced forensic capabilities, retrospective detection, and improved deployability and management. We evaluated our CloudAV implementation using a large malware dataset and a production deployment of our system on a real-world campus network for a period of over six months.

Using the malware dataset, we showed how N-version protection provides substantial gains in detection coverage against recent threats. In addition, we demonstrated how our cloud-centric architecture enables advanced functionality, such as retrospective detection, which can greatly mitigate the impact of the large window of vulnerability presented by antivirus products. It is clear from our research and subsequent commercial activity that a cloud-based approach for malware detection indeed offers significant advantages over a host-centric approach.

While CloudAV brought benefits for malware detection in traditional computing environments, this thesis further explored the application of the CloudAV principles to resource-constrained mobile devices. We argued that it is possible to expend bandwidth resources to significantly reduce on-device CPU, memory, and power resources. We demonstrated how our cloud-centric model enhances mobile security and reduces on-device software complexity, while allowing for new services such as platform-specific behavioral analysis engines. Our benchmarks on Nokia's N800 and N95 mobile devices showed that our mobile agent consumes an order of magnitude less CPU and memory while also consuming less power in common scenarios compared to existing on-device antivirus software. With the skyrocketing adoption and sophistication of mobile devices, protecting these devices against malicious threats in a resource-efficient and scalable manner is increasingly important and can be effectively performed with a cloud-centric service.

Next, this thesis described a cloud-centric service, called PolyPack, that uses an array of packers and antivirus engines to select the packer that will result in the optimal evasion of the antivirus engines. Toward understanding the utility and efficacy of such a service, we constructed an implementation of PolyPack which employs 10 packers and 10 popular antivirus engines. Our evaluation showed that PolyPack provides 258% more effective evasion of antivirus engines than using an average packer and out-evades the best evaluated packer for over 40% of the binary samples. At its peak, PolyPack offered private packing services to over 50 personally-vetted researchers and penetration testers. Besides exposing the ease at which modern antivirus can be evaded in a comprehensive and automated fashion, PolyPack provided several important proof points that indicate the power of cloud-centric services. For one, PolyPack demonstrated that cloud services can be rapidly

prototyped and deployed (e.g., with the fusion of the packing service with the existing CloudAV service). More importantly, PolyPack demonstrated that the positive attributes of cloud services may also be abused to benefit attackers. Not surprisingly, since our initial research, a wide range of cloud-centric crimeware services, including those with PolyPack-like functionality, have cropped up in the wild and are in active use.

With thousands of new malware samples flooding in each day, scalable approaches to classification are an absolute necessity to gain a grasp of the ever-evolving capabilities of malware authors. This thesis advocated for a cloud-centric security service to provide such classification services at-scale. Deploying such a classification service not only allows us to elastically scale and leverage our existing virtualization platform for safely executing and tracing malware, but also offers the opportunity to open the service to other researchers, resulting in more effective classification through the network effect. Toward this goal, we developed and evaluated a dynamic analysis approach based on causal tracing of the operating system objects created due to malware's execution. The reduced collection of these user-visible system state changes (e.g., files written, processes created) was used to create a fingerprint of the malware's behavior and automatically clustered into groups that reflect similar classes of behaviors. The benefits of scale and isolation provided by our virtualized classification service were instrumental in our research.

Lastly, this thesis investigated a new cloud-centric architecture for generating cryptographic signatures using threshold cryptography. While the majority of the thesis has discussed malware-related services, our CloudCard research demonstrated that the benefits of cloud-centric services apply to other realms of security as well. CloudCard employed a threshold RSA signature scheme with private key material split across three parties: a user's host, a user's mobile device, and a cloud service. For a typical signing operation, partial signatures are individually computed by each of the parties, combined together on the user's host, and then typically passed along to a service for authentication purposes. The CloudCard approach has very tangible benefits including improved secrecy of private key material, flexible and fast revocation, and out-of-band signature confirmation while maintaining compatibility with existing applications and interfaces. To evaluate CloudCard, we implemented a PKCS#11 module for SSH pubkey authentication that cooperates with an

Android mobile application and our cloud service. In summary, CloudCard is ideal example of taking a security mechanism that was previously deployed in a purely host-centric model and adapting the functionality to a cloud-centric model.

7.2 Insights and Lessons

Before concluding, it is important to review several of the key insights and lessons learned from the research presented throughout this thesis.

- **An Evolution of Security Threats and Computing Capabilities:** Malicious threats have evolved past the capabilities of our current host-centric security mechanisms. Rather than propose incremental point solutions in an attempt to keep up with such sophisticated adversaries, we believe that innovative and fundamentally different architectures for security mechanisms are required to fight an asymmetric battle. We also observe that advances in cloud computing and high-speed, low-latency networking have drastically changed how software is deployed and consumed. This unique intersection of cloud computing and security offers the ability to develop and deploy novel security services in a cloud-centric architecture. We observe that this evolution in computing has removed many of the performance and usability constraints that would have previously hampered such security services. For example, transmitting files over the network to a centralized service running 10 different antivirus engines in parallel, like we advocate in CloudAV, would seem like an absurdly infeasible approach before the advent of utility computing, virtualization, and high-speed network interconnects. However, after our research results demonstrated its feasibility, it seems like a straightforward and obvious evolution of security technology. We theorize that as computing capabilities continue to evolve in the future, we'll observe the development of similar "enabling technologies" in the security space to address the continually-evolving malicious threats.

- **A New Perspective for Security Mechanisms:** While cloud-centric services offer a promising new approach for software security mechanisms, they do have similari-

ties with previous network-centric and host-centric mechanisms. When approaching a security service, one can separate the underlying mechanism (e.g., misuse detection, anomaly detection, access control) from the service itself (e.g., AV, NBADS, firewall). For example, host-centric antivirus and network-centric NIDS both employ signature-based mechanisms. However, the former operates on a host-based perspective with file-based granularity, while the latter operates on a network-based perspective with flow or packet-based granularity. Identical underlying mechanisms are also present in network-based and host-based firewalls and network-based and host-based anomaly detection systems. So while the core underlying mechanism remains unchanged in these examples, the security capabilities of these services can differ wildly based purely on their perspective and level of granularity that they're able to inspect. This is a subtle, yet powerful point: the perspective of a security mechanism can often have a more significant impact on its efficacy than the mechanism itself. We see this same principle applied to our proposed cloud-centric model in effect throughout the thesis (e.g., signature-based mechanism moved to cloud service in CloudAV, cryptographic signature computation moved to cloud service in CloudCard, etc). We've seen the cloud-centric model offer a powerful new perspective for such security mechanisms: the global scope of a network-centric perspective, yet the fine-granularity of a host-centric perspective.

- **A Blueprint for Constructing Cloud-Centric Services:** A common blueprint for building cloud-centric security services exists across the chapters of this thesis. While the benefits of our cloud-centric security service often appear to be derived from a hybrid of network-centric and host-centric models, the actual construction of such services is more of an evolution of the host-centric model. In several of the services presented in this thesis, a purely host-centric security mechanism was *segmented* and adapted to a cloud-centric model. This segmentation usually involves splitting up the host-centric functionality and moving a subset of it to a network-attached cloud service. For example, with CloudAV we segmented the detection engine functionality of antivirus and migrated it to a cloud service, while leaving behind the lightweight

agent on the end host. Similarly, with CloudCard we segmented the signature computation and migrated a portion of the operation to a cloud service and a mobile device. Many of the benefits of the security services presented in this thesis were derived from this process of adapting a host-centric security mechanism to a cloud-centric service. We feel this approach represents a logical and practical blueprint for approaching the construction of future cloud-centric software security services.

7.3 Future Work

This thesis has laid the foundation for future researchers to explore the benefits of cloud-centric security services. Our research has shown that existing security mechanisms can be vastly improved when deployed in a cloud-centric architecture. In addition, we've shown that completely novel services can be designed when leveraging the primitives provided by cloud computing. As cloud computing and high-speed networking continue to evolve, we anticipate that the benefits of our proposed approaches will be further amplified. In addition, we envision severals areas of future work that may extend the principles described in this thesis:

- **Exploring trade-offs of granularity and abstraction:** Interesting trade-offs may exist between the efficacy and performance of a cloud-centric security service based on the granularity of data collected and operated on. Too fine-grained information may have significant impact on end hosts, the network, and the cloud service and may be highly dependent on the deployment model and the number of hosts participating in the service. For example, the CloudAV implementation presented in this thesis operated primarily on a file-based granularity, sending suspicious files to the cloud for analysis when they were observed on an end host. However, one can envision a cloud security service that operates on finer-grained system activity or objects in order to detect and mitigate malicious attacks. For example, a cloud security service observing the control flow graphs of userland or kernel activity may be more effective in detecting in-memory attacks that never touch persistent storage devices. One could imagine even finer-grained approaches such as full-blown synchronously

or asynchronously replicated computing with a network-attached reference monitor. Thoroughly exploring these security and performance trade-offs may offer insight into which classes of security services can be efficiently deployed with our current and future computing capabilities.

- **Defending against malicious use of cloud services:** While this thesis has shown that one can design new architectures that offer asymmetric gains over existing approaches to address modern security threats, we must also realize that attackers can use the same beneficial properties of cloud computing to bolster their attacks, as we demonstrated with our PolyPack research. Future work may investigate whether there exist inherent properties in the way cloud services are architected or deployed that can be leveraged to make their use for malicious purposes more difficult. Alternately, it is possible that the availability of public cloud IaaS providers are a boon for fraudsters, offering scalable, bullet-proof platforms to host their malicious crimeware services. While this thesis focuses on the benefits of cloud services regardless of intent, this is will undoubtedly be an important research area as malicious actors continue to increase in organization and sophistication.

- **Designing trustworthy and host-proofed services:** As the adoption of cloud-based services is rapidly accelerating, the trustworthiness of the service provider is a major area of contention. Enterprises offloading security services to third-party cloud providers have serious and well-founded concerns about the confidentiality, integrity, and availability of their sensitive data. Therefore, it is vital that future work in the area of cloud-centric security services consider the threat model of a malicious or compromised cloud infrastructure provider and adapt architectural design decisions to be as resilient and trustworthy as possible. For example, techniques involving private information retrieval, tokenization, and homomorphic encryption have been proposed for ensuring data confidentiality in third-party cloud services. While designing completely host-proofed services is a non-trivial task, it is possible to achieve mathematically-provable security guarantees even in the face of a malicious or compromised cloud provider as we demonstrated with our CloudCard research.

- **Architecting the next generation of cloud-centric security services:** In addition to amplifying the benefits of our proposed approaches, we anticipate that the further evolution of cloud computing and high-speed networking will open up opportunities for the next generation of cloud-centric security services that may be infeasible or impractical in the current day. We believe that the lessons presented by this thesis will serve as a blueprint for the research and development of next generation cloud-centric security services.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Normalized compression distance for gene expression analysis. In *Workshop on Genomic Signal Processing and Statistics (GENSIPS)*, May 2005.

[2] M. Abdalla, S. Miner, and C. Namprempre. Forward-secure threshold signature schemes. *Topics in Cryptology–CT-RSA 2001*, pages 441–456, 2001.

[3] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 2–13. ACM, 2006.

[4] Debin Gao andDesiree Beck, Julie Connolly" Michael K. Reiter, and Dawn Xiaodong Song. Behavioral distance measurement using hidden markov models. In *RAID*, pages 19–40, 2006.

[5] Avast Antivirus. Avast. `http://www.avast.com/en-us/index`.

[6] AVG Antivirus. Avg. `http://www.avg.com/us-en/homepage`.

[7] BitDefender Antivirus. Bitdefender. `http://www.bitdefender.com`.

[8] F-Prot Antivirus. F-prot. `http://www.f-prot.com/`.

[9] F-Secure Antivirus. F-secure. `http://www.f-secure.com/en/web/home_us/home`.

[10] Kaspersky Antivirus. Kaspersky. `http://usa.kaspersky.com/`.

[11] McAfee Antivirus. mcafee. `http://www.mcafee.com/us/`.

[12] Symantec Antivirus. Symantec. `http://www.symantec.com/index.jsp`.

[13] Trend Micro Antivirus. Trend micro. `http://us.trendmicro.com/us/home/index.html`.

[14] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

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

[16] 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.

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

[18] Piotr Bania. Generic Unpacking of Self-modifying, Aggressive, Packed Binary Programs. `http://piotrbania.com/all/articles/pbania-dbi-unpacking2009.pdf`, 2009.

[19] P. Barford and V. Yegneswaran. An inside look at botnets. *Malware Detection*, pages 171–191, 2007.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[21] bart, xt. Fast Small Good (FSG). `http://www.woodmann.com/collaborative/tools/index.php/FSG`, 2009.

[22] Desiree Beck and Julie Connolly. The Common Malware Enumeration Initiative. In *Virus Bulletin Conference*, October 2006.

[23] Abhilasha Bhargav, Rahim Sewani, and Sarvjeet Singh. Intrusion fault-tolerance using threshold cryptography. *Purdue University, Threshold Cryptography*, 2004.

[24] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 31–38. IEEE, 2010.

[25] D. Bolzoni, E. Zambon, S. Etalle, and P. Hartel. Poseidon: A 2-tier anomaly-based intrusion detection system. *Arxiv preprint cs/0511043*, 2005.

[26] D. Boneh, X. Ding, and G. Tsudik. Fine-grained control of security capabilities. *ACM Transactions on Internet Technology (TOIT)*, 4(1):60–82, 2004.

[27] D. Boneh, X. Ding, G. Tsudik, and C.M. Wong. A method for fast revocation of public key certificates and security capabilities. In *Proceedings of the 10th conference on USENIX Security Symposium-Volume 10*, pages 22–22. USENIX Association, 2001.

[28] D. Boneh and M. Franklin. Efficient generation of shared rsa keys. *Advances in Cryptology–CRYPTO'97*, pages 425–439, 1997.

[29] M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel. Attacking and fixing pkcs# 11 security tokens. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 260–269. ACM, 2010.

[30] C. Boyd. Digital multisignatures. *Cryptography and coding*, 1986.

[31] Death by Captcha. Death by captcha: Fastest discount captcha solvers. `http://www.deathbycaptcha.com`.

[32] BypassCaptcha.com. Bypass captcha. `http://www.bypasscaptcha.com/`.

[33] G.T. Byrd, F. Gong, C. Sargor, and T.J. Smith. Yalta: A secure collaborative space for dynamic coalitions. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 30–37. Citeseer, 2001.

[34] Danilo Bzdok. Yoda's Crypter. `http://yodap.sourceforge.net`, 2009.

[35] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology–CRYPTO'99*, pages 78–78. Springer, 1999.

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

[37] Brian Caswell and Marty Roesch. Snort: The open source network intrusion detection system, June 2004.

[38] P.M. Chen and B.D. Noble. When virtual is better than real. *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS)*, pages 133–138, 2001.

[39] T.M. Chen. Stuxnet, the real start of cyber warfare?[editor's note]. *Network, IEEE*, 24(6):2–3, 2010.

[40] Y. Chen, V. Paxson, and R.H. Katz. Whats new about cloud computing security? *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, 20(2010):2010–5, 2010.

[41] WetStone Chet Hosmer. Polymorphic and metamorphic malware. `http://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf`.

[42] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[43] S. Chow, C. Boyd, and J. Nieto. Security-mediated certificateless cryptography. *Public Key Cryptography-PKC 2006*, pages 508–524, 2006.

[44] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, and R.E. Bryant. Semantics-aware malware detection. In *Ieee symposium on security and privacy*, pages 32–46. Citeseer, 2005.

[45] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)*, pages 32–46, Oakland, CA, USA, May 2005. ACM Press.

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

[47] J. Clulow. On the security of pkcs# 11. *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 411–425, 2003.

[48] Fred Cohen. *A Short Course on Computer Viruses*. John Wiley & Sons, 2nd edition, April 1994.

[49] E. Cooke, M. Bailey, Z.M. Mao, D. Watson, F. Jahanian, and D. McPherson. Toward understanding distributed blackhole placement. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 54–64. ACM, 2004.

[50] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX SRUTI Workshop*, pages 39–44, 2005.

[51] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet (SRUTI 2005 Workshop)*, Cambridge, MA, July 2005.

[52] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[53] F-Secure Corporation. F-secure mobile anti-virus. `http://mobile.f-secure.com/`, 2008.

[54] Nokia Corporation. Maemo sdk. `http://maemo.org/`, 2008.

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

[56] Symantec Corporation. Symantec mobile antivirus for windows mobile. `http://www.symantec.com/norton/products/overview.jsp?pcid=pf&pvid=smavwm`, 2008.

[57] The MITRE Corporation. Cve-2006-4926. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4926`, 2006.

[58] The MITRE Corporation. Cve-2006-4927. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4927`, 2006.

[59] The MITRE Corporation. Cve-2010-0098. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0098, 2010.

[60] Aldo Cortesi. Host-proof applications. http://corte.si/posts/security/hostproof.html.

[61] L.P. Cox and P.M. Chen. Pocket Hypervisors: Opportunities and Challenges. *Proceedings of HotMobile*, 2007.

[62] I. Damgård and M. Koprowski. Practical threshold rsa signatures without a trusted dealer. *Advances in Cryptology–EUROCRYPT 2001*, pages 152–165, 2001.

[63] DeCaptcher. Decaptcher. http://decaptcher.com/client/.

[64] S. Delaune, S. Kremer, and G. Steel. Formal security analysis of pkcs# 11 and proprietary extensions. *Journal of Computer Security*, 18(6):1211–1245, 2010.

[65] Y. Deswarte, L. Blain, and J.C. Fabre. Intrusion tolerance in distributed computing systems. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 110–121. IEEE, 1991.

[66] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[67] X. Ding and G. Tsudik. Simple identity-based cryptography with mediated rsa. In *Proceedings of the 2003 RSA conference on The cryptographers' track*, pages 193–210. Springer-Verlag, 2003.

[68] 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.

[69] Dwing. UPack. http://dwing.cjb.net, 2009.

[70] Dan Ellis, John Aiken, Kira Attwood, and Scott Tenaglia. A Behavioral Approach to Worm Detection. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM04)*, October 2004.

[71] C. Ellison and B. Schneier. Ten risks of pki: What you're not being told about public key infrastructure. *Comput Secur J*, 16(1):1–7, 2000.

[72] N. Falliere, L.O. Murchu, and E. Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[73] Jason Flinn, Dushyanth Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 61–66, Schloss Elmau, Germany, May 2001.

[74] International Organization for Standardization. Iso/iec 7816. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=29257.

[75] Internet Engineering Task Force. Rfc 2560. http://www.ietf.org/rfc/rfc2560.txt.

[76] Internet Engineering Task Force. Rfc 3280. http://www.ietf.org/rfc/rfc3280.txt.

[77] M. Fossi, G. Egan, K. Haley, E. Johnson, T. Mack, T. Adams, J. Blackbird, M.K. Low, D. Mazurek, D. McKinney, et al. Symantec internet security threat report trends for 2010. *Volume XVI*, 2011.

[78] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[79] A. Fox, S.D. Gribble, E.A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. *ACM SIGPLAN Notices*, 31(9):160–170, 1996.

[80] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[81] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *Advances in Cryptology–EUROCRYPT'96*, pages 354–371. Springer, 1996.

[82] Bernard Golden. Cloud computing: Two kinds of agility. http://www.cio.com/article/599626/Cloud_Computing_Two_Kinds_of_Agility.

[83] Google. Android - an open handset alliance project. http://code.google.com/android/, 2008.

[84] Google. Google safe browsing. http://code.google.com/apis/safebrowsing/, 2008.

[85] G. Gu, R. Perdisci, J. Zhang, W. Lee, et al. BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Usenix Security Symposium*, 2008.

[86] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium*, pages 167–182, 2007.

[87] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[88] C. Herley and D. Florêncio. Nobody sells gold for the price of silver: Dishonesty, uncertainty and the underground economy. *Economics of Information Security and Privacy*, pages 33–53, 2010.

[89] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 100–110. ACM, 1997.

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

[91] Y. Huang, D. Rine, and X. Wang. A jca-based implementation framework for threshold cryptography. In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pages 85–91. IEEE, 2001.

[92] N. Ianelli and A. Hackworth. Botnets as a vehicle for online crime. *CERT Coordination Center*, pages 1–28, 2005.

[93] IBM/ISS. Proventia network intrusion prevention system. `http://www-935.ibm.com/services/us/index.wss/offerfamily/iss/a1030570`.

[94] Inc Immunity. CANVAS. `http://www.immunitysec.com/products-canvas.shtml`, 2009.

[95] Cisco Systems Inc. Netflow services and applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`, 2002.

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

[97] 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.

[98] American National Standards Institute. Public key cryptography using reversible algorithms for the financial services industry (rdsa). *ANSI X9.31-1998*.

[99] S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptosystems without erasures. In *Eurocrypt'00*, pages 221–242, 2000.

[100] M. Jensen, J. Schwenk, N. Gruschka, and L.L. Iacono. On technical security issues in cloud computing. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 109–116. Ieee, 2009.

[101] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, page 138. ACM, 2007.

[102] Jibz, Qwerton, snaker, xineohP, BoB. PEiD. `http://www.peid.info`, 2009.

[103] Burt Kaliski. Twirl and rsa key size. *RSA Laboratories*.

[104] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. Renovo: a hidden code extractor for packed executables. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malcode*, 2007.

[105] L.M. Kaufman. Data security in the world of cloud computing. *Security & Privacy, IEEE*, 7(4):61–64, 2009.

[106] 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.

[107] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 223–236, Bolton Landing, NY, USA, October 2003. ACM.

[108] J. Zico Kolter and Marcus A. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research*, 2007.

[109] Eleftherios Koutsofios and Stephen C. North. Drawing graphs with *dot*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 8 October 1993.

[110] 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.

[111] Thomas Kunz and Sali Omar. A mobile code toolkit for adaptive mobile applications. In *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, pages 51–59, Monterey, CA, December 2000.

[112] Kaspersky Lab. Kaspersky mobile security. `http://usa.kaspersky.com/products_services/mobile-security.php`, 2008.

[113] N. Leavitt. Is cloud computing really ready for prime time? *Computer*, 42(1):15–20, 2009.

[114] Tony Lee and Jigar J. Mody. Behavioral classification. In *Proceedings of EICAR 2006*, April 2006.

[115] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitányi. The similarity metric. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 863–872, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[116] Z. Li, Q. Liao, and A. Striegel. Botnet economics: uncertainty matters. *Managing Information Risk and the Economics of Security*, pages 245–267, 2009.

[117] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *Proc. of IEEE Symposium on Security and Privacy*, 2006.

[118] M. Locasto, J. Parekh, S. Stolfo, A. Keromytis, T. Malkin, and V. Misra. Collaborative distributed intrusion detection. *Dept. Computer Science, Columbia Univ., Tech. Rep. CUCS-012-04*, 2004.

[119] A. Lysyanskaya. Efficient threshold and proactive cryptography secure against the adaptive adversary. 1999.

[120] A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. *Advances in Cryptology–ASIACRYPT 2001*, pages 331–350, 2001.

[121] Justin Ma, John Dunagan, Helen Wang, Stefan Savage, and Geoffrey Voelker. Finding Diversity in Remote Code Injection Exploits. *Proceedings of the USENIX/ACM Internet Measurement Conference*, October 2006.

[122] P. MacKenzie and M. Reiter. Two-party generation of dsa signatures. In *Advances in Cryptology–CRYPTO 2001*, pages 137–154. Springer, 2001.

[123] M. Malkin, T. Wu, and D. Boneh. Experimenting with shared generation of rsa keys. In *1999 Symposium on Network and Distributed System Security (SNDSS)*, pages 43–56. Citeseer, 1999.

[124] L. Martignoni, M. Christodorescu, and S. Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2007.

[125] McAfee. W32/Sdbot.worm. `http://vil.nai.com/vil/content/v_100454.htm`, April 2003.

[126] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[127] Microsoft. Microsoft security intelligence report: January-june 2006. `http://www.microsoft.com/technet/security/default.mspx`, October 2006.

[128] Microsoft. Microsoft security intelligence report. `http://www.microsoft.com/security/portal/Threat/SIR.aspx`, 2009.

[129] Y. Miretskiy, A. Das, C.P. Wright, and E. Zadok. Avfs: An on-access anti-virus file system. In *Proceedings of the 13th USENIX Security Symposium (Security 2004)*, pages 73–88, 2004.

[130] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *Security & Privacy, IEEE*, 1(4):33–39, 2003.

[131] D. Moore, C. Shannon, et al. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284. ACM, 2002.

[132] Alex Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A Crawler-based Study of Spyware in the Web. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2006.

[133] 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.

[134] M. Myers. Revocatoin: Options and challenges. In *Financial Cryptography*, pages 165–171. Springer, 1998.

[135] 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.

[136] Arbor Networks. Arbor malware library (AML). `http://www.arbornetworks.com`, 2009.

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

[138] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. *Proceedings 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 8–11, 2005*, 2005.

[139] P. Ning, Y. Cui, D.S. Reeves, and D. Xu. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):318, 2004.

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

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

[142] Inc. Nullriver. iphone installer.app. `http://iphone.nullriver.com/`, 2008.

[143] Inc Nullsoft. NSIS. `http://nsis.sourceforge.net`, 2009.

[144] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[145] Markus Oberhumer. UPX. `http://upx.sourceforge.net/`, 2009.

[146] Federal Bureau of Investigation. Cyber attacks: Net jam. http://web.archive.org/web/20070326115414/http://www.fbi.gov/libref/factsfigure/factsfiguresapri2003.htm.

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

[148] OpenBSD. ssh-add - adds private key identities to the authentication agent. http://www.openbsd.org/cgi-bin/man.cgi?query=ssh-add.

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

[150] D. Quist and Valsmith. Covert Debugging: Circumventing Software Armoring. In *Proc. of Black Hat USA*, 2007.

[151] M.A. Rajab, F. Monrose, and A. Terzis. On the effectiveness of distributed worm monitoring. In *Proceedings of the 14th conference on USENIX Security Symposium-Volume 14*, pages 15–15. USENIX Association, 2005.

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

[153] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, 2003.

[154] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[155] 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.

[156] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

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

[158] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. The Remote Processing Framework for portable computer power saving. In *Proceedings of the ACM Symposium on Applied Computing*, San Antonio, TX, February 1999.

[159] R. Sandhu, M. Bellare, and R. Ganesan. Password-enabled pki: Virtual smartcards versus virtual soft tokens. In *Proceedings of the 1st Annual PKI Research Workshop*. Citeseer, 2002.

[160] C. Shannon and D. Moore. The spread of the witty worm. *Security & Privacy, IEEE*, 2(4):46–50, 2004.

[161] Monirul Sharif, Andrea Lanzi, Jonathon Giffin, and Wenke Lee. Automatic Reverse Engineering of Malware Emulators. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland '09)*, 2009.

[162] V. Shoup. Practical threshold signatures. In *Advances in Cryptology–EUROCRYPT 2000*, pages 207–220. Springer, 2000.

[163] 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.

[164] 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.

[165] Matt "skape" Miller. Using dual-mappings to evade automated unpackers. In *Uninformed Journal Vol 10*, 2008.

[166] North Star Software. NsPack. `http://www.nsdsn.com/eng/index.htm`, 2009.

[167] D. Solo, R. Housley, and W. Ford. Internet x. 509 public key infrastructure certificate and crl profile. 1999.

[168] Inc. Sourcefire. Clamav antivirus. `http://www.clamav.net/`, 2008.

[169] R.S.A.C. Standard. Pkcs #11 cryptographic token interface standard. *RSA Laboratories*.

[170] R.S.A.C. Standard. Pkcs #7: Cryptographic message syntax standard. *RSA Laboratories*.

[171] StarForce. ASpack. `http://www.aspack.com/`, 2009.

[172] B. T OGRAPH and Y.R. MORGENS. Cloud computing. *Communications of the ACM*, 51(7), 2008.

[173] 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.

[174] Toni Koivunen / Teamfurry.com. SigBuster. `http://www.teamfurry.com`, 2009.

[175] Bitsum Technologies. PECompact. `http://www.bitsum.com/pecompact.php`, 2009.

[176] Oreans Technology. Themida. `http://www.oreans.com/`, 2009.

[177] tHE EGOiSTE/TMG. tElock. `http://programmerstools.org/node/164`, 2009.

[178] K. Thomas and D.M. Nicol. The koobface botnet and the rise of social malware. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 63–70. IEEE, 2010.

[179] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[180] Kaushik Veeraraghavan, Ed Nightingale, Jason Flinn, and Brian Noble. qufiles: a unifying abstraction for mobile data management. In *The Ninth Workshop on Mobile Computing Systems and Applications (HotMobile 2008)*, February 2008.

[181] Brian Walters. VMware virtual platform. *j-LINUX-J*, 63, Jul. 1999.

[182] Y.M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 35–49, 2006.

[183] Stephanie Wehner. Analyzing worms and network traffic using compression. Technical report, CWI, Amsterdam, 2005.

[184] P. Wohlmacher. Digital certificates: a survey of revocation methods. In *Proceedings of the 2000 ACM workshops on Multimedia*, pages 111–114. ACM, 2000.

[185] T. Wu, M. Malkin, and D. Boneh. Building intrusion tolerant applications. In *Proceedings of the 8th conference on USENIX Security Symposium-Volume 8*, pages 7–7. USENIX Association, 1999.

[186] 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.

[187] Vinod Yegneswaran, Jonathon T. Giffin, Paul Barford, and Somesh Jha. An Architecture for Generating Semantics-Aware Signatures. In *Proceedings of the 14th USENIX Security Symposium*, pages 97–112, Baltimore, MD, USA, August 2005.

[188] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127. ACM New York, NY, USA, 2007.

[189] B. Zenel. A general purpose proxy filtering mechanism applied to the mobile environment. *Wireless Networks*, 5(5):391–409, 1999.

[190] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[191] P. Zheng. Tradeoffs in certificate revocation schemes. *ACM SIGCOMM Computer Communication Review*, 33(2):103–112, 2003.