Server-Side Detection of Malware Infection

Markus Jakobsson
Palo Alto Research Center
Palo Alto, CA 94304
markus.jakobsson@parc.com

Ari Juels
RSA Laboratories
Cambridge, MA 02142
ajuels@rsa.com

ABSTRACT

We review the intertwined problems of malware and online fraud, and argue that the fact that service providers often are financially responsible for fraud causes a relative lack of incentives for clients to manage their own security well. This suggests the need for a server-side tool to determine the security posture of clients before letting them transact.

We introduce an exceedingly lightweight audit mechanism to address this need – permitting for post-mortem infection analysis – and prove its security properties based on standard cryptographic hardness assumptions. We describe a deployment architecture that aligns the incentives of participants in order to facilitate quick adoption and widespread use of the technology. Our approach is flexible enough to protect even low-end computing devices like mobile handsets, which future malware will target heavily, but whose power and bandwidth limitations result in poor effectiveness for traditional anti-virus solutions.

A contribution of independent potential value is the enabling of a centralized analysis of malware-related events, which promises to extend the power of detection in comparison to what today's decentralized paradigm allows.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems, Security and Protection; H.1.2 [**Information Systems**]: Models and Principles, User/Machine Systems

General Terms

Security

Keywords

anti-virus, audit, cell phone, detection, fraud, incentive compatible, infection, malware, mobile, post-mortem, retroactive

NSPW'09, September 8–11, 2009, Oxford, United Kingdom

1. INTRODUCTION

Currently, industry sources estimate that 0.25% of infected computers are directly involved in financial fraud [3]. But with malware penetrating 10-15% [15] of *all* computers connected to the Internet, converting these computers into tools for committing financial fraud (which can easily happen) would result in an instant fifty-fold increase of fraud. To put this in perspective, most businesses would fail or fold their Internet operations long before this happened. This situation urgently calls for a change in malware-combatting strategies, or the Internet will wither along with the economy that supports it.

As worrisome as this is, the problem is even more acute in the context of mobile malware. A problem first given attention in 2006 [20], it has recently been identified [35, 18] as a likely game changer within two to three years, at which time it is expected that there will be a comparable number of smart phones and laptops/desktops computers in use. The expected escalation of the problem is both due to the explosive adoption rates of smartphones, and due to their inherent limitations – such as power, memory, and bandwidth. The latter makes current Anti-Virus (AV) tools unsuitable once the rate of new or mutated malware instances reaches a threshold beyond which it is not reasonable to push updates. (Currently, there are close to forty thousand new instances of Windows malware *a day* and updates to AV filter rules several times an hour. This volume is not easy for a smartphone to manage.) Finally, phones may be more desirable platforms to attack, given the rich data they hold, and the fact that they inherently are payment platforms [22]. Even with today's relatively contained problem, consumers are wary [17].

Looking beyond the technical parameters of the problem, we see that this pending catastrophe owes much to poor alignment of incentives. Financial service providers commonly bear the full financial responsibility for fraud[1], while most of the available malware countermeasures are designed for client-side use. To come to terms with malware-fueled fraud, financial service providers need access to information allowing them to better evaluate a client's security posture. This type of information is sometimes available to network nodes – ISPs and backbone nodes – whose identification

[1]In the U.S., for example, federal law caps consumer liability for credit-card related fraud at $50 per card. Commonly, financial service providers waive this liability altogether, and consumers only have to invest their time to set things straight, and bear the emotional burden of having been victimized.

of corruption can be made using traffic analysis (observing patterns of communication) and sometimes by analysis of packet contents. However, network nodes are often not able to attribute traffic to a given originating machine, given that the network nodes cannot distinguish between different machines behind one IP address. Accurate information about a client's security posture is available to the *client itself*, though. The dilemma is that it is not trivial to extract this information in a reliable manner, in view of the fact that a corrupted machine will do anything it can to cover up the corruption.

Our contribution.

The main contribution of this paper is a new approach to malware management. We propose a system that allows a server to analyze the security posture of a client in order to detect (probable) malware corruption. More precisely, our system is a framework within which a client can compile an integrity-protected log of local security events, with no requirement for trusted hardware, and transmit this log to a trusted server for post-mortem detection of infection. By performing this audit in real-time, one obtains a preventive filtering functionality in addition to the retroactive protection. Here, we note that the log is malware-resistant: It either contains correct records up to the time of the audit, or else contains corruptions detectable to the server as signifying a malware attack.

We describe simple cryptographic tools in support of this framework. We also offer a rough taxonomy and some examples of the types of events that a client can log to facilitate server analysis. We do not offer detailed guidance around the type of events that should be logged, nor do we study resource requirements and malware-detection efficacy for event selection options from an empirical standpoint. Instead, we defer most log-formulation choices for future work.

Our proposals hinge on the simple insight that in order for a log to permit accurate post-mortem analysis, events that impact a client's security posture—e.g., installation of new application software—must be logged *before* they take effect. Otherwise, the event itself, if staged by a malicious entity, can subvert the log so as to conceal infection of the client. We note that the logs can be *inspected* on the central server at any point in time after the logging of an event is performed, or they would be of very limited value. The logs can be erased from the client as soon as an audit has completed and the server has obtained a copy of the logs. In principle, these logs would then be kept indefinitely on the server, in order to permit retroactive detection of infection. However, practically speaking, logs will have to be kept for much shorter periods of time – days, weeks, or maybe months.

Given the trustworthy log provided by our proposed methods, a server can take several different approaches to analyzing a client's security posture. It can adopt a whitelist approach to identify legitimate programs and/or a blacklist approach that permits detection of known dangerous events – whether characterized by code signatures, event origin, or circumstances of executable installation or use.

Our methods apply to a wide range of infection vectors, including drive-by malware installation, exploitation of vulnerabilities in legitimate software, and attacks in which a criminal coerces a user to install a legitimate program, only then to exploit a vulnerability in it to cause the execution of arbitrary code. We note that our tools – while intended to help control economically motivated crimes – can also help curb ideologically and politically motivated online crimes (see [11] for an overview of this problem). Our logging techniques may aid in law enforcement by providing forensic information or creating a digital chain of custody to prove that only accredited tools ran on a given machine, and, in general, as a way to offer assurance that a given machine is used in a manner that complies with some set of expectations. Finally, the techniques can be used to audit electronic voting equipment remotely, to identify attempts to commit electoral fraud.

Our techniques are designed not just for traditional computational platforms, but also work particularly well on constrained computing devices that cannot run AV software effectively, such as mobile handsets. Audits in our system can be conducted periodically, e.g., when a mobile handset has sufficient supporting bandwidth. In contrast, traditional AV software is only effective when operating continuously, which may pose severe limitations on resource-constrained devices. Our techniques also support resource-adaptive approaches to audit, e.g., breaking a log into a small, high-priority segment for rapid audit and a larger, lower-priority segment for off-peak analysis.

It should be emphasized that our approach respects net neutrality: It is not necessary to tie the anti-virus mechanism to the infrastructure, but it is straightforward for third-party providers to perform data collection and auditing.

Outline.

We begin by reviewing the related work (section 2), followed by a basic review of what architectural approach is most suitable from the point of view of good system-wide security (section 3). We then detail what problem we need to solve to reach our goal (section 4). In section 5, we provide a high-level description of our proposed audit mechanism. This section also includes a review of privacy enhancements to our basic construction and a discussion of the benefits of collecting audit data on the network to augment the data already collected and reported by clients. Section 5 also describes how a central analysis of logs permits anomaly detection that cannot easily be performed in the traditional AV paradigm. Finally, in section 6, we describe our technical building blocks and provide a proof of robustness against corruption.

2. RELATED WORK

There are four principal approaches to combating malware-based fraud. The *first*, and currently most popular approach, is technical. This can be done as a server-side measure, typically as back-end anomaly detection (e.g., RSA FraudAction, 41st parameter and Mark Monitor). It can also be done as a client-side measure, as in the case of traditional anti-virus (AV) software. It is worth noting that – as simple as having an up-to-date AV filter may seem – it is believed [29] that less than 50 percent of networked computers in the developed world do. Consumer education (see, e.g., [25, 32]), which is a *second* principal approach, can help boost those numbers and suppress the rate of user-approved malware installations, but hardly eliminate the problem. Policy measures offer a *third* approach entailing

the creation of legislative and regulatory frameworks that favor stronger computer security. Policies may range from regulation to encourage ISPs to block traffic from apparently infected machines [1], to providing tax incentives for deployment of AV services, to legislation such as the CAN-SPAM Act in the U.S. A *fourth* principal approach is to step up law enforcement efforts to make online fraud less appealing to criminals by pursuing perpetrators. This includes efforts to infiltrate criminal networks and trace traffic to its sources. These four main approaches are symbiotic to a large extent, with technical measures playing a main and often enabling role.

It is starting to become commonly recognized that the traditional AV paradigm is poorly suited for mobile platforms, due to their resource constraints. The limitations in terms of battery power, memory resources and bandwidth make both signature-based technologies and behavioral technologies problematic for resource-constrained devices – at least when the detection software is run on the client machine. This is true even for any plausible or foreseeable advances in battery, memory and communications technology. One approach to address the problem could therefore be to run detection software on the *network* (see, e.g., [30]), using information indicating the security posture of the client device. If done right, this approach may mitigate the problems arising from end-device limitations. It is non-trivial to perform remote audits, though, as infected client machines can misrepresent their security posture, fabricating transcript evidence of having no infection.

Our solution lets a server efficiently audit the security posture of a client machine. Port-scanning is a common example of this general approach. Port-scanning can be used to detect the use of communication protocols that are likely to indicate vulnerability; for example, sending a DNS query to port 53 will tell whether the probed machine has a DNS server. Network Access Control (NAC) is a technique that allows a network to control access by verifying that connecting nodes satisfy certain minimum requirements, such as having AV software, an approved update level, and an appropriate configuration. This type of audit allows the endpoints to quarantine a connecting node that does not meet the requirements. Both of these techniques are commercially useful, but offer no guarantees against infection. In fact, it is possible for malware to communicate via commonly used ports (or simply not communicate while being audited), and compliance with requirements is in no way an assurance of security. Similarly, while so-called browser-recon [23] can be used to inspect the browser history of a target machine for signs of corruption, it is trivial for a malware agent to erase incriminating portions of the browser history once a machine has been corrupted.

Stronger assurance is obtainable in principle by the use of trusted hardware. Trusted Platform Modules (TPMs) are widely deployed hardware devices that generate remotely consumable attestations to the software stack executed by a client [34]. They can support rigorous enforcement of NAC policies requiring clients to run approved software configurations. Despite hundreds of millions of shipped units, however, TPMs have seen little application [36]. Implementation complexity, key-management challenges, and concerns about aggressive application to digital-rights management have stymied their use, but so too have more fundamental elements of their design. TPM attestations support only a whitelisting approach, and do not gracefully handle the dynamically changing nature of software configurations resulting from patches, driver updates, and so forth. Additionally, TPMs attest to the presence of an ostensibly trustworthy software configuration, but in their standard and intended modes of use provide no assurance against runtime attacks on software vulnerabilities—e.g., exploits of OS bugs.

We propose a solution in which a server can audit a client machine, avoiding the shortcomings of the approaches just described. Our technique combines client-side logging with server-side auditing. This general architecture was first proposed by Bellare and Yee [7, 8] to record important system events, such as crashes, resource exhaustions, and failed login attempts. Bellare and Yee do not mention detection of malware corruption among their motivations, though, nor is their solution geared toward this type of attack. In its basic instantiation, their solution is vulnerable to what we may term a limited "roll-back" attack, in which a malware agent erases the latest log entry (or entries) immediately upon infecting a client machine. This is the entry that provides evidence of corruption. The attack works due to the fact that the log remains open until explicitly closed – whether at the end of a time epoch or at the request of a server – and the latest log entries can be manipulated during this time period.

Independently of Bellare and Yee, Schneier and Kelsey [31] also proposed an audit mechanism, which is similar in flavor, but has privacy-preserving enhancements. Their solution is similarly vulnerable to roll-back attacks due to the need for explicit log closing, and also lacks formal security analysis. The same shortcomings exist in the schemes proposed by Jin, Myles, and Lotspiech [26, 27], who propose an audit structure aimed to improve software tamper resistance.

We describe two solutions – one based on symmetric-key cryptography alone, and one based on public-key cryptography, both of them provably secure against roll-back attacks. Both are exceedingly simple – the symmetric-key solution authenticates pending events using a simple construction based on message authentication codes, and the public-key version – a simple but crucial modification of Bellare and Yee's scheme – uses a forward-secure signature scheme to authenticate pending events.

Our approach might be viewed loosely as an enabler of server-side execution of AV software [30]. Centralized analysis of events is not only a matter of moving the computational burden of detection from clients to servers, but also enables new pattern-based detection approaches [21]. For example, Bluetooth malware exhibits a strong geographic component in terms of how it spreads, whereas installation of a system patch is likely to instead depend on the local time of the day, and the time the patch was released.

Our solution allows for detection of corruption of typical user computers and mobile phones. This is in contrast to the more specialized solution proposed by Choi, Golle and Jakobsson [9, 10], in which deviation of the behavior of a client machine could be detected by a third-party auditor – but *only* if the audited client only performs publicly verifiable computation, such as the computation of a digital signature or the mixing of ciphertexts.

Durfee, Smetters and Balfanz's [13] recent work on posture-based data protection describes a mechanism by which a client machine can protect its contents using encryption using a key released by a third party – the trusted audit server

– after this server has verified that the client machine passes a security audit. Their solution addresses a different problem than we do; it does not describe the audit process, but treats that as an issue orthogonal to the question of how to protect the data on the machine against attack.

Although it is not explicitly discussed in the context of previous audit proposals (as listed above), it is evident that secure audit requires timely erasure of keys. Erasure is known to be a complicated problem – see e.g., [12, 16, 19]. However, we note that since our adversary is a software agent, it cannot access data on the physical layer, as a human adversary might. Therefore, it suffices to simply overwrite the old contents of the cells that were used to store the old secret key, along with intermediate values that were generated as the new secret key was computed. Thus, while the erasure component of the problem must be carefully attended to in general audit contexts, it is straightforwardly addressed in the context of malware detection.

3. ARCHITECTURE

We demonstrate that there exist both symmetric-key and public-key audit mechanisms that are robust against malware infection. The existence of a public-key version implies that it is possible to let *any* machine audit the security posture of *any other* machine. However, while this may at first seem beneficial, there are privacy reasons – and therefore also security reasons – not to do this. It is, for example, clearly undesirable for an untrustworthy server to get to learn that a given client has installed software that makes it vulnerable. To address these privacy concerns, one can imagine a solution in which a client records "scrubbed" data to avoid the privacy limitations associated with reporting detailed logs. However, this is an imperfect solution, as evidenced by the large gray area associated with executions and installations of programs that are benevolent but have some vulnerability. Furthermore, the *context* in which programs are called matters to the security of the system, but it is difficult to record the right amount of detail without sacrificing either the security or privacy of users. Worse still, it is *impossible* to perform retroactive audits relating to a program just found to have a vulnerability if only limited contextual data associated with this program has been recorded in the past. We therefore conclude that a model with one or a small number of trusted servers is better than a model in which any machine can query any other machine, since it limits the problem posed by rich audit data.

We assume that each end user establishes an audit relationship with an entity he or she trusts with his or her privacy, and that this entity can be queried by other machines and requested to return information about the security posture of the client in question – possibly according to one out of a small number of definitions of what it means to be secure. It can be argued that it is in the best interest of this entity not to abuse this trust, just as software-as-a-service (SaaS) providers today rely for commercial success on sustaining an appearance of respect for the privacy of their users. (In addition to simply relying on the trustworthiness of the selected servers, though, there are technical constructions that can limit the amount of data these servers need to handle. We discuss some of these possible extensions to our audit construction.)

It turns out that a centralized audit structure is also beneficial as regards deployment incentives. To succeed, a security technology has to be *used*. Recognizing the inadequate penetration rates of AV software, it is crucial to avoid this problem from reoccurring in the context of secure audit services. The source of the problem lies simply in poor incentive alignment: Consumers are not always convinced that they need AV software – or aware of what protection they have or do not have. At the same time, it is difficult for AV vendors to make financial service providers pay for services offered to end users (although this sometimes does happen, in the way of partnerships that produce end-user discounts for AV software). Further, as client-side software, AV systems must strike a difficult balance between effectiveness and performance. AV vendors often favor speed over thorough scans, as consumers are intolerant of security-induced performance degradation. To succeed, a well aligned deployment incentive structure is vital. A simple approach would be to ship all operating systems with audit software, and for financial service providers – or any service provider, for that matter – to carry the cost of the maintenance of the system by being required to pay to receive audit data from audit servers. This requirement again points in the direction of a structure with a low number of trusted audit servers who have established trust relationships with end users, and who are supported by those who would otherwise carry the direct cost of security failures: The financial service providers.

4. PROBLEM STATEMENT

We want to be able to remotely detect corruption of a machine. An adversary can corrupt a victim's machine (i.e., install software on it) either by coercing a user to approve the installation or exploiting a software vulnerability. There are actually *three* separate avenues of corruption: (1) the adversary coercing a user to install malware on the target machine; (2) the adversary exploiting a vulnerability in a piece of benevolent software running on the target machine; and (3) the adversary coercing a user to install benevolent software with a vulnerability, after which he exploits this vulnerability.

One can, in turn, classify software into three types: *malware*, benevolent software *known* to have vulnerabilities, and benevolent software *not known* (by the "good guys") to have vulnerabilities. Here, we choose to use the term "software" in a broad sense to include operating system code and routines, firmware, and other fairly static code, as well as traditional user-installed applications.

A given machine is exposed to a series of *events*. An event corresponds to any invocation or installation of software, and can be characterized by the code of the calling program, the location from which it was obtained, and the history of user actions leading up to the event.

Formal Statement.

We define a log Γ as a sequence of event entries and accompanying cryptographic data. We let $\Gamma(j)$ denote event entry j in a log with $\geq j$ entries. In an execution of our protocol, we let Γ_i denote the log generated in response to the i^{th} system event. We let K_i denote key state in epoch i. This may include both a key and supplementary data.

We specify a forward-secure malware audit system as a function triple $\mathsf{FSMAS} = (\mathsf{setup}, \mathsf{log}, \mathsf{audit})$, defined as follows:

- $\mathsf{setup}(1^l) \rightarrow K_0$: Generates a shared client / server key with security parameter l.

- $\mathsf{log}(\Gamma_{i-1}, K_{i-1}, \epsilon_i) \rightarrow (\Gamma_i, K_i)$: Runs on the client; appends event ϵ_i to log Γ_i, outputting an updated log and key state (and deleting K_{i-1}); we use subscript i to denote log epochs.

- $\mathsf{audit}(\Gamma, K_0) \rightarrow \{0, 1\}$: Runs on the server; verifies the integrity of log Γ.

We define the security of FSMAS according to the following experiment run with adversary $\mathcal{A}$ whose running time is polynomially bounded in l:

Experiment $\mathbf{Exp}_{\mathcal{A},\mathsf{FSMAS}}[l, n]$
  $K_0 \leftarrow \mathsf{setup}(l)$; $\Gamma_0 = \phi$;
  for $i = 1$ to n;
    $\epsilon_i \leftarrow \mathcal{A}(\Gamma_{i-1}, \text{“test”})$;
    $(\Gamma_i, K_i) \leftarrow \mathsf{log}(\Gamma_{i-1}, K_{i-1}, \epsilon_i)$;
  $\Gamma' \leftarrow \mathcal{A}(\text{“forge”}, K_n)$;
  if $\mathsf{audit}(\Gamma', K_0) = 1$ and
      $\exists j \in [1 \ldots n-1]$ s.t. $\Gamma'(j) \neq \Gamma_i(j)$ then
    output ‘1’;
  else
    output ‘0’;

We say that FSMAS is *robust* if for all $\mathcal{A}$ and values of n polynomial in l, we have $\mathsf{pr}[\mathbf{Exp}_{\mathcal{A},\mathsf{FSMAS}}[l, n] = 1]$ negligible in l. We say that FSMAS *detects* event $\mathcal{M}$ provided that $\mathsf{audit}(\Gamma, K_0) = 1$ iff $\Gamma(j) = \mathcal{M}$ for any j. (Of course, this definition can be relaxed to allow false positives or false negatives.)

In the next section, we describe a robust solution that can detect any event that can be identified by a combination of a whitelist, a blacklist and heuristics.

5. LOGGING AND AUDITING

This section begins with a high-level description of the processes associated with audits. We then describe the policies used to identify what to log. (This is an area where we believe substantial improvements are possible, especially in terms of heuristic mechanisms to detect low-volume attacks that are designed to go unnoticed.) We then discuss user privacy, and how to trade off audit accuracy against improved privacy. This is followed by a description of why – and how – to collect additional audit information on the network.

The Basic Components.

Our solution requires client-side installation of the logging software and the selection of one (or more) audit servers. The latter involves running (on either the client or server) the setup routine in which a key is selected and communicated between the client and server machines.

Before the client machine $\mathcal{C}$ allows an event to take place, it is recorded by calling the routine log. The order here is crucial: If the event were allowed to take place prior to being logged – and the event involved infection – then the audit mechanism could be subverted – see figure 1.

The audit mechanism, finally, is performed by calling the routine audit. This may be done with a certain, fixed frequency, or whenever the client device has sufficient resources, such as bandwidth and power. It is possible to consider a collection of logs, each one of which contains different types of events (potentially of different urgency or associated volume), and to audit different logs under different circumstances.

Various implementations of these three routines are possible; we describe two sets of routines below – one based on symmetric key cryptography, the other on public key cryptography.

Evaluating the Client Security Posture.

The audit function provides a server with highly probable indication of client-side infection in cases where it detects log corruption. Scanning an intact log for evidence of malware is much more challenging, as suggested in our discussion of event selection. Broadly speaking, there are three different approaches to server identification of client infection:

1. **Whitelisting.** Whitelisting can be useful in paring down a log to enable more focused analysis of potentially suspect events, and to identify the use of benevolent programs with known vulnerabilities. In a whitelisting approach, the server refers to a list of executables / client behaviors believed to be secure. The whitelist policy of course dictates what type of event data should be logged by a client. Various programs may require different data to be logged – especially in light of the fact that some benevolent programs may be known or suspected to have vulnerabilities, and it is vital to collect information that allows the auditor to determine whether the invocation of a piece of software on the whitelist corresponds to an infection or not. In a pure whitelisting approach, a server would only classify a client posture as secure if its log contains exclusively whitelisted events.

2. **Blacklisting.** In a blacklisting approach, the server refers to a list of executables / client behaviors known to induce (or suspected of inducing) client-side vulnerabilities. In a pure blacklisting approach, the server would only classify a client posture as secure if its log contains no blacklisted events. Examples of blacklists present on clients today include the signature file of an AV filter and the list of offending websites present in some browsers.

3. **Heuristics.** Whitelists and blacklists both have their limitations. As the scope of logged events grows, many events may be ambiguous or only subject to probabilistic analysis, or analysis in a larger context. Heuristics may be used to analyze events that are highly correlated with malware infection, even if they cannot be identified as installation or execution. For example, a user that frequently visits online gambling sites may be at higher risk for infection—a fact useful for a server to know in its assessment of client posture. But visiting a class of sites is not a clear indication that a client actually *has* been infected with malware. Such heuristic analysis is similar to the behavioral model in a traditional AV filter.

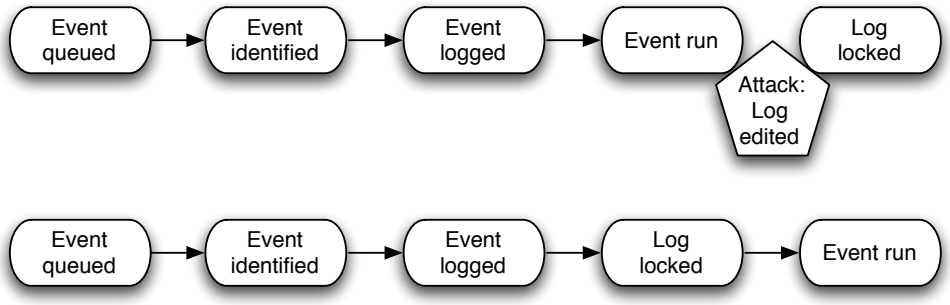

Figure 1: The top sequence corresponds to previously proposed audit mechanisms in which the log is not locked until after the execution of the event. We note that this is not a problem outside the context of malware related events, as the event itself would not pose a threat to the audit mechanism then. The lower sequence shows the simple conceptual modification embodied by our proposed solution.

Useful heuristics include receipt of email from a person not in the list of contacts; any connection attempt from an external source; visits to URLs that the user did not navigate to (e.g., iframes); browser redirection with an invalid or missing REFERER field; any event following the installation of software, and within the same session[2]; any installation following (within the same session) a refusal by the user to install a piece of software; and any *uncommon* event[3].

The most useful heuristic analysis may very well be one that takes into consideration several concurrent dimensions to describe an installation or execution; we refer to [21] for a discussion of this approach.

Event Selection: What to Log.

The most critical decision in the design of our system is what events should be placed in the client log in support of the whitelisting, blacklisting, and heuristic posture-assessment approaches outlined above. Our goal is to record a broad enough range of event types to ensure capture of indicators of malware infection for a substantial fraction—ideally, a large majority—of vectors of attack. At odds with this goal is the desire to ensure that the logging process is efficient, both in terms of computational overhead and the size of the resulting log. Among those with the highest ratio of detection efficacy to logging cost are:

- **Installed executables:** A common vector of malware infection is direct installation by a deceived user–an event subject to capture via logging of installed executables. The challenge with this approach is how most effectively to fingerprint an executable.
  - A file hash is effective for whitelisting approaches to vetting executables, but is not meaningful in the context of blacklisting, since packers will easily defy such measures.
  - An approach reminiscent of the the current "signature" approach for file fingerprinting, as used in traditional AV filtering, allows us to model blacklisting to some limited extent. However, it is unsuitable for whitelisting, since signature collisions are trivial to obtain.[4] File fingerprinting is potentially ineffective for blacklisting, given the threat of polymorphic malware.
  - More strongly invariant and effective for blacklisting approaches are the URL and IP address of origin of the executable[5]. In special cases, it may be relevant to report the entire binary to the trusted authority, although this must be avoided as far as possible.
- **Opened attachments:** Another frequent vector of infection are executables in attachments. Here, however, blacklisting approaches based on executable-file contents or sender names are unworkable. Attackers can readily modify executables and sender names for individual users. The routing history of the message is more likely a better data source for blacklisting—although still challenging to use effectively in the face of botnet-based email distribution.
- **Browsed URLs / IP addresses:** Drive-by attacks [14] are a growing vector for malware installation. Browsing history provides potentially important coverage of this infection vector.
- **Wireless connections.** Local connections, whether WiFi or Bluetooth, pose a potential threat in terms

[2]Here, we define session in the context of a browser, with any process being forked from a triggering process being considered part of the session. This case corresponds to a sequence of events starting with the installation of benevolent software with a vulnerability in.

[3]With "uncommon", we mean less common on the target machine than on an average machine, whether relating to the type of application involved or the technical building block used. This is a particularly useful rule, as it is local, and therefore cannot be anticipated by an attacker wishing to test an infection strategy beforehand.

[4]It is important that executable descriptors be "collision-free" – that it is not possible to construct a piece of malware that has the same descriptor as a piece of software on the whitelist.

[5]Given that attackers use dynamic addresses for malware distribution, and that URLs can be subverted via attacks such as DNS poisoning, additionally valuable may be the referring source.

of malware infection – both as it comes to delivery of payload and of corruption of routing information. These connections may cause slow but insistent spread of malware, given the difficulty in monitoring them from the backbone. However, detailed traffic analysis information for connections may help identify patterns indicative of epidemics. Therefore, simply reporting back that a connection was made, along with location information or other identifying data, could help identify infection.

- **Data for anomaly detection.** As described in [21], it may be helpful to record data that does not directly allow for identification of infection attempts – such as approximate geographic location, and whether an attachment was received from a contact from the address book – in order to use anomaly detection as an early-warning system. It is clear that some of this data might negatively affect the privacy of the user, and care has to be taken to strike an appropriate balance between privacy and security. This issue will be discussed briefly next.

Privacy Enhancing Constructions.

Whether a symmetric or asymmetric building block is used in the audit construction, it is possible to enhance the privacy of users by limiting the expressiveness of the data reported back during an audit. This can be done by replacing the event to be recorded by one or more ciphertexts describing the event in various degrees of detail, and a more general classification of the event in plaintext format. The server could then decrypt different collections of event descriptors – potentially based on the current threat structure. The access to the decryption key may be controlled using traditional threshold cryptography. Depending on the expressiveness of the various classifications (a matter which can be a function of the type of application, and of user preferences), different trade-offs between privacy and security can be made.

Another privacy enhancing approach involves a tiered audit, whether hierarchical or circular. An example of a hierarchic structure is a tree-based solution in which parent nodes audit children; an example of a circular structure involves two peers (such as a smartphone and the computer it synchronizes with) to audit each other. In a tiered audit, the auditing node would simply maintain records indicating the audit outcome, using an independent (second-level) audit log. *This* is the log that then will be audited by a third party – we refer to it as the *second-order* log. In figure 2, we see an example of a circular audit in which two peer nodes audit each other, and the resulting audit logs are then audited by a third party, the so-called audit server. Tiered audit can be structured in a way that enhances privacy, as the second-order log could contain only a filtered set of entries.

Augmenting the Infrastructure.

To improve resistance against malware infection and online fraud still further, it is beneficial to collect and report information on the network. Some of this information can closely mimic the information logged on the client device; for example, it is possible for a wireless access point to record URLs of visited sites, and other notable information. Just as a client computer can be audited by a trusted server, so can an access point. This can be straightforwardly done using the client machine as a proxy that downloads and forwards the audit logs to the verifying server; it can even be done as part of a regular audit of the client[6].

Moreover, network nodes at a greater distance from client machines may be used to collect and report on traffic data. Whether this is done using a secure audit mechanism or not, this information may be of a more aggregate form and used to identify failed attempts to record events indicative of infection. Namely, if network traffic indicative of a spam bot can be traced to a small set of network endpoints, none of which has reported an infection, then this inconsistency is a sign of failure – and infection. Awareness of failures is important for the maintenance of client policy information.

A Note on Machine Dependencies.

Our exposition so far has dealt mostly with how to protect a client desktop or laptop computer, and has focused on typical user-initiated actions, such as those associated with common web browsing activities. However, the type of policies one would use depends on the type of machine to be protected. Let us consider a few examples to illustrate this:

1. Unlike a laptop or desktop machine, a wireless access point (WAP) would not necessarily have access to the packet contents (due to use of end-to-end encryption), but would have access to information regarding the visited URLs and IP addresses, and could record these. (There are limitations: E.g., for HTTPS traffic, the WAP sees only the hostname.) Due to its near-constant connectivity, it is easy to update the WAP's policies quickly. Some of the policies may reside on the network instead of on the endpoint WAP node, and require interaction to evaluate. This has the benefit of making reverse engineering and adversarial penetration testing harder.

2. As cell phones increasingly become used to perform payments, their value will increase to attackers. At the same time, as the number of applications – and therefore the number of potential vulnerabilities – increase, the policies will become increasingly complex. This is undesirable given the limitations on computation (due to power conservation issues) and storage. Therefore, and just like for the WAPs, it may become desirable to manage some of the policies on the network, in particular those that would not require excessive amounts of data to be communicated in order to be evaluated.

3. The infotainment system[7] of a car would mostly communicate with an associated cell phone, and using a very limited interface. The type of information exchanged – now and in the relatively near future – may correspond to uploading of directions and coordinates, synching of address books, transmission of user-specified content markups, and use of the hands-free

[6]For access points serving multiple clients, the MAC address or other identifier of the requesting machine may be recorded to allow for the clients to be distinguished, and the forwarding of the audit logs done by a principal client only.

[7]While it is unlikely that infotainment systems will allow user-guided installations of arbitrary applications, it is clear that any of the approved applications may be vulnerable to attacks, just like any applications on traditional computers.

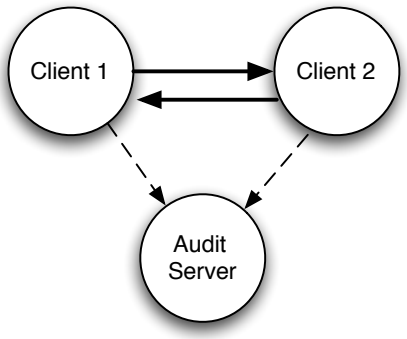

Figure 2: In a circular audit, only very limited audit data is transmitted to the main audit server, resulting in privacy benefits for the client nodes. Assuming a relative independence between the peer devices, the attacker's best approach is to infect one device first, and then attack the other via the audit process. Given the relative simplicity of this process, corruption is difficult.

phone features. Due to the high losses associated with successful corruption (largely due to the PR impact on the brand) and the relative lack of storage limitations, it may be desirable to record the entire transcripts in the audit log.

Note that several types of audits are possible: To begin with, an infrastructure node can audit the vehicular system's interaction with the phone, to identify infections of the infotainment system. It is also possible for phones and infotainment systems to audit each other.

We note that it is possible to quickly roll out machine-specific policies to protect against epidemics on the rise. It is not only the target machines that would receive these updates. For example, assume that there are signs indicating that a given cell phone infection is becoming common. Apart from updating the policies on vulnerable devices, one may also update them on any machines[8] that are set up to perform peer-audits of the vulnerable machines – whether these are traditional computers, vehicular infotainment systems, or infrastructure nodes such as cell phone base stations. This way, these associated devices can assist in maintaining the security of the target device: They would have information about what patterns indicate corruption.

6. BUILDING BLOCKS

We present two constructions for the building block underlying the secure audit process. One is based on symmetric-key cryptography, and can be proved secure based on common assumptions on the underlying hash function. The other is based on public key cryptography, and can be proved secure based on number-theoretic assumptions. The two constructions share the same principal structure, as is illustrated schematically in figure 3.

[8]The main auditing server would know what types of devices a given client machine is set up to audit, and could distribute targeted updates of the audit policies to machines that have been set up to audit any client type of concern.

A Symmetric Construction.

In the following, we present a construction for $FSMAS$. We require a cryptographic hash function $h : Z_* \to \{0,1\}^l$ and a message authentication function MAC. (The latter can be constructed from a hash function [5].)

Our construction is largely straightforward, but it is worth pointing out the role of the value c_i, which is a new construction. Namely, c_i is a key-validation value, i.e., a value that proves that the key k_i is current, but without revealing the value. This is part of the protection against roll-back attacks.[9]

- **Setup.** The function $\mathsf{setup}(l) \to K_0$ computes $K_0 = (c_0 = \perp, k_0)$, where $k_0 \stackrel{R}{\leftarrow} \{0,1\}^l$. It is executed by the server $\mathcal{S}$ or client $\mathcal{C}$ and K_0 securely transmitted to the other party. The client sets counter $i \leftarrow 0$.
- **Logging.** To log event ϵ_i, the function $\mathsf{log}(\Gamma_{i-1}, K_{i-1}, \epsilon_i) \to (\Gamma_i, K_i)$ computes $\mu_i = MAC_{k_i}(0 \parallel \epsilon_i)$, sets $\Gamma_i \leftarrow \Gamma_{i-1} \parallel (\epsilon_i, \mu_i)$ and $K_i = (c_i, k_i)$, where $k_i \leftarrow h(k_{i-1})$ and $c_i \leftarrow MAC_{k_i}(1)$. It then erases K_{i-1} by overwriting the corresponding cells[10]. The client increments its counter value i.
- **Audit.** When the server $\mathcal{S}$ wishes to audit $\mathcal{C}$, it requests (Γ, c), the current log and key-verification value c from the client $\mathcal{C}$. These must be sent over a secure channel, namely one that ensures session freshness (to prevent rollback attacks). The function $\mathsf{audit}(\Gamma, K_0) \to \{0,1\}$ extracts a counter value i from Γ by counting the number of entries. It computes k_j for all $j \leq i$ and verifies all corresponding MACs in Γ. It also verifies that $c = h(1, k_i)$. If all verifications are valid, then audit outputs '1'; otherwise, it outputs '0.'

[9]We note that it is possible to eliminate the use of c_i by logging the request to perform an audit as one particular event, along with a nonce or a counter. This slightly reduces the amount of computation and storage required by the client, but at the cost of the need for keeping more state across machines.

[10]It is not enough to free up the space, as that makes the old value still accessible to a malware agent with a non-negligible probability.

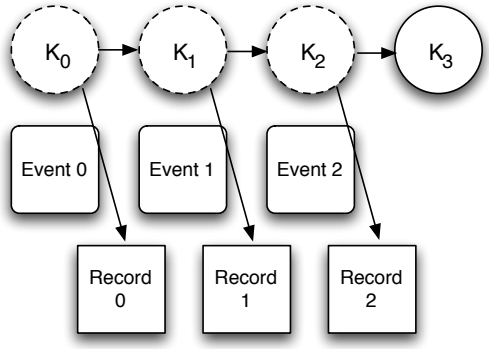

Figure 3: When an event occurs, a new log record is computed from a key and the event descriptor. Then, a new key is computed, and the old key is erased – as indicated by the dotted circles. Since events are logged before they are allowed to take place, attackers cannot modify the logged records: The key needed to compute this record has already been erased then.

Remarks: We note that any forward-secure PRF can alternatively be used to compute keys in the above. We offer the conceptually simplest option here. Note also that it is possible to reduce the amount of necessary computation by interpreting c_i as the 64 first bits of k_i; only these bits of what otherwise is k_i will be transmitted to the server during an audit. The server verifies that the log has not been truncated by verifying the correctness of these bits – recall that old instances of k_i, and therefore also c_i, will be unavailable to a malware agent that infected during time interval i. Also, note in the context of the audit operation that once an audit on Γ_i has been performed, to minimize client storage and later communication, $\mathcal{S}$ can retain Γ_i, while the client can erase it and in later audits send only $\Gamma_j - \Gamma_i$ for $j > i$.

CLAIM 1. *The proposed system* FSMAS *is robust.*

Proof [Sketch].

For simplicity, we make the (strong) assumption that it is infeasible for any poly-time algorithm to distinguish between $\{MAC_r(M_j)\}_{j=1}^n$ and $\{r_j\}_{j=1}^n$ for any random key r and set of random values pair $\{r_j\}_{j=1}^n$, i.e., that a set of MACs with a secret, random key on chosen messages are indistinguishable from random. We model h as a random oracle.

We then construct a sequence of experiments $\mathsf{FSMAS} = S_1, S_2, \ldots S_{n-1}$, as follows. In S_1 (and all later simulations in the sequence), the simulator substitutes uniformly random values for the triple (k_1, c_1, μ_1). In S_2, it substitutes uniformly random values for the triple (k_2, c_2, μ_2) for μ_2 and k_2, etc. Now suppose that $\mathsf{pr}[\mathbf{Exp}_{\mathcal{A},S_2}[l,n] = 1] - \mathsf{pr}[\mathbf{Exp}_{\mathcal{A},\mathsf{FSMAS}}[l,n] = 1]$ is non-negligible. Then either the adversary queries the random oracle for h on k_1 with non-negligible probability—which is not possible, given its knowledge only of images of k_1—or it can distinguish between $(MAC_r(0 \parallel m), MAC_r(1))$ and (r_1, r_2), for random (r, r_1, r_2) and chosen m, violating our assumption. A similar argument holds if $\mathsf{pr}[\mathbf{Exp}_{\mathcal{A},S_j}[l,n] = 1] - \mathsf{pr}[\mathbf{Exp}_{\mathcal{A},S_{j-1}}[l,n] = 1]$ is non-negligible for any j in the sequence.

Therefore, as $\mathcal{A}$ cannot distinguish with non-negligible probability between a true and random distribution on the audit transcript for epochs through $n-1$ with non-negligible probability. It is straightforward then to show that $\mathsf{pr}[\mathbf{Exp}_{\mathcal{A},\mathsf{FSMAS}}[l,n] = 1]$ is negligible in l, and hence that FSMAS is robust.

An Asymmetric Construction.

An asymmetric construction for the building block underlying our audit mechanism can be immediately obtained from a secure forward-secure signature (FSS) scheme, such as [6]. The setup of the building block is the same as the setup of the FSS scheme; the logging operation corresponds to producing a signature and evolving the secret key. The key-validation value c can be represented as one-way function of the secret key from the previous epoch, or similar to our stream-lined approach above, as a portion of the current secret key[11]. As for the symmetric version, old instances of c are erased along with old instances of the secret key.

The audit operation involves the verification of an uninterrupted series of signatures from the FSS scheme, starting with the signature associated with the first time period of the FSS scheme, and ending with the signature associated with the key-validation value. The security of this construction depends directly on that of the underlying FSS scheme, and on the fact that a queued event is not run until it has been logged. The proof of security at that level of abstraction is analogous to the proof of the security of our symmetric construction.

7. CONCLUSION

We believe that we are close to a tipping point at which malware costs will upset the fragile balance of attack and defense within which Internet commerce remains sustainable today. Malware also threatens to erupt as a dominant threat on mobile handsets as these devices grow more open and sophisticated. We do not believe that the current anti-virus paradigm can control this alarming situation much longer: Mobile handsets are too limited to run AV software, and mutating malware is blunting the power of AV even on more powerful computing platforms.

In this paper, we have proposed a new paradigm that uses a server- or cloud-based post-mortem detection of client in-

[11]We note that the value c is never revealed to an attacker, as all audits are performed over encrypted channels.

fection. Having argued that our proposal is both sound and practical, we hope to see other challenges to and improvements of traditional malware defenses. In our view, they are long overdue.

Acknowledgments

We wish to thank Jan Feyereisl, Sean Peisert and Sid Stamm for helpful feedback on an earlier version of the manuscript, and Michael Barrett, Eric Davis, Hampus Jakobsson, Karl-Anders Johansson, Werner Johansson, Christopher Kruegel, Allyn Romanow, and Matt Williamson for stimulating discussions and helpful suggestions. Moreover, we wish to thank the anonymous NSPW reviewers for their bountiful and helpful feedback. Finally, we wish to thank the participants of the 2008 Santa Fe workshop on Malware for inspiring discussions leading up to the formulation of the need for the proposed technology, and the Santa Fe Institute for hosting the workshop.

8. REFERENCES

[1] The Australian Internet Security Initiative. www.acma.gov.au/WEB/STANDARD/pc=PC_310317.

[2] M. Barrett and D. Levy, "A Practical Approach to Managing Phishing," April 2008.

[3] M. Barrett, "Cybercrime – and What We will Have to do if We Want to Get it Under Control," July, 2008. publius.cc/cybercrime_and_what_we_will_have_do_if_we_want_get_it_under_control.pdf.

[4] Michael Barrett, PayPal CSO, personal communication.

[5] M. Bellare, R. Canetti and H. Krawczyk, "Keying Hash Functions for Message Authentication," Advances in Cryptology - Crypto 96 Proceedings, 1996.

[6] M. Bellare and S.K. Miner, "A Forward-Secure Digital Signature Scheme," Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, p.431-448, 1999.

[7] M. Bellare and B. Yee, "Forward Integrity for Secure Audit Logs," Technical Report, University of California at San Diego, 23 November 1997.

[8] M. Bellare and B. Yee, "Forward-Security in Private-Key Cryptography," Topics in Cryptology - CT-RSA 03, pp. 1-18, 2003.

[9] J.Y. Choi, P. Golle and M. Jakobsson. "Auditable Privacy: On Tamper-Evident Mix Networks." Financial Cryptography, 2006.

[10] J.Y. Choi, P. Golle, and M. Jakobsson, "Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware," DACS, 2006.

[11] K.-K. R. Choo, "Organised crime groups in cyberspace: a typology," Trends in Organized Crime, DOI 10.1007/s12117-008-9038-9, 2008.

[12] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson, "How To Forget a Secret," In STACS. LNCS 1563. Springer-Verlag, pp. 500-509, 1999.

[13] G. Durfee, D.K. Smetters, and D. Balfanz, "Posture-Based Data Protection," PARC Technical Report 06-11; September 2006. www.parc.com/publication/2302/posture-based-data-protection.html

[14] M. Egele, E. Kirda, and C. Kruegel. Mitigating Drive-by Download Attacks: Challenges and Open Problems. In iNetSec, 2009.

[15] Georgia Tech Information Security Center, "Emerging Cyber Threats Report for 2009," October, 2008. www.gtisc.gatech.edu/pdf/CyberThreatsReport2009.pdf.

[16] A. J. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," Commun. ACM, pp. 91-98, 2009.

[17] Harris Interactive Public Relations Research, "A Study about Mobile Device Users," June 2009. Abstract available at www.cloudmark.com/en/company/mobile-survey.html, full version at www.cloudmark.com/releases/docs/harris_poll_stats.pdf.

[18] S. Havlin, "Phone Infections," Science, Vol. 324. no. 5930, pp. 1023 - 1024, 22 May 2009.

[19] G. Hughes and T. Coughlin, "Tutorial on Disk Drive Data Sanitization," cmrr.ucsd.edu/people/Hughes/DataSanitizationTutorial.pdf.

[20] M. Hypponen, "Malware Goes Mobile," Scientific American Magazine, pp. 70-77, November 2006.

[21] M. Jakobsson, "A Central Nervous System for Automatically Detecting Malware," blogs.parc.com/blog/2009/09/a-central-nervous-system-for-automatically-detecting-malware/, September 9, 2009

[22] M. Jakobsson, "Will mobile payments usher in a new era of crime?," blogs.parc.com/blog/2009/10/will-mobile-payments-usher-in-a-new-era-of-crime/, October 13, 2009

[23] M. Jakobsson, T. Jagatic, and S. Stamm, "Phishing for Clues: Inferring Context Using Cascading Style Sheets and Browser History," www.browser-recon.info.

[24] Javelin Strategy and Research, "2008 Identity Fraud Survey Report", February 2008. www.javelinstrategy.com/products/F59339/97/delivery.pdf.

[25] P. Kumaraguru, S. Sheng, A. Acquisti, L.F. Cranor, and J.I. Hong, "PhishGuru: Lessons From a Real World Evaluation of Anti-Phishing Training," e-Crime Researchers Summit, Anti-Phishing Working Group, October 15 - 16, 2008.

[26] H. Jin, G. Myles and J. Lotspiech, "Towards better software tamper resistance," Information Security Conference, Springer, 2005.

[27] H. Jin, G. Myles and J. Lotspiech, "Key evolution-based tamper resistance: A subgroup extension," Association for Computing Machinery (ACM) Symposium on Information, Computer and Communications Security (ASIACCS), 2007.

[28] Kaspersky Lab forecasts ten-fold increase in new malware for 2008. www.kaspersky.com/news?id=207575629.

[29] E. Mills, "Microsoft to offer free consumer security suite," November, 2008. news.cnet.com/8301-1009_3-10101582-83.html.

[30] J. Oberheide, E. Cooke and F. Jahanian, "CloudAV: N-Version Antivirus in the Network Cloud," Proceedings of the 17th USENIX Security Symposium (Security'08), 2008.

[31] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines," ACM TISSEC, 2(2):159Ñ176, 1999.

[32] S. Srikwan and M. Jakobsson, "Using Cartoons to Teach Internet Security," Cryptologia, vol. 32, no. 2, 2008.

[33] Symantec Report on the Underground Economy, Published November 2008, `eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_underground_economy_report_11-2008-14525717.en-us.pdf`.

[34] Trusted Computing Group. Trusted platform module main specification. Version 1.2, Revision 103, July 2007.

[35] P. Wang, M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding the Spreading Patterns of Mobile Phone Viruses," Science, 2 April 2009.

[36] J. Wiens. A Tipping Point For The Trusted Platform Module? *InformationWeek*. 28 June 2008. `www.informationweek.com/news/security/encryption/showArticle.jhtml?articleID=208800939`.