Non-Invasive Methods for Host Certification

Patrick Traynor, Michael Chien, Scott Weaver, Boniface Hicks and Patrick McDaniel
Systems and Internet Infrastructure Security (SIIS) Lab
Department of Computer Science and Engineering
The Pennsylvania State University

Abstract— **Determining whether a user or system is exercising appropriate security practices is difficult in any context. Such difficulties are particularly pronounced when uncontrolled or unknown platforms join public networks. Commonly practiced techniques used to vet these hosts, such as system scans, have the potential to infringe upon the privacy of users. In this paper, we show that it is possible for clients to prove both the presence and proper functioning of security infrastructure without allowing unrestricted access to their system. We demonstrate this approach, specifically applied to anti-virus security, by requiring clients seeking admission to a network to positively identify the presence or absence of malcode in a series of puzzles. The implementation of this mechanism and its application to real networks are also explored. In so doing, we demonstrate that it is not necessary for an administrator to be invasive to determine whether a client implements good security practices.**

Keywords: Certification, Assurance, Network Security, Malware

I. Introduction

Access to and protection of many networks has traditionally been predicated on user authentication. Users providing the necessary credentials, typically in the form of a password, are given access to some authorized subset of resources within a domain. While separating "insiders" from unauthorized users, common network admission processes make no assessment of the safety of the connecting host. Because the machines of even the most trusted users are becoming the unwitting hosts of the current malware pandemic, user authentication alone is no longer a sufficient mechanism for providing protection. Instead, the security configuration of machines must be inspected to ensure that all hosts are appropriately protected.

Vetting hosts in this way is a problem of *certification*: untrusted hosts need to be evaluated to ensure they meet some predefined best practices for security. Once a host is determined to meets these minimal standards, it may be allowed access to the network or be subjected to further inspection or authentication. The set of practices one must adhere to is a critical and environment-specific element of a network's security policy.[1] However, the manner in which proof of adherence to these practices is obtained is a thorny issue. Because simply asserting an inherently untrusted platform's settings is insufficient, more invasive mechanisms are typically employed.

Given the available techniques, system scanning is the most often used form of host vetting. Before allowing machines to log on to a network, organizations such as the NSF require that they first be aggressively scanned for malware and security configuration by specialized software. Commercial software packages implementing system scans are widely available [9], [33], [30], [29]. While effective, this approach is neither desirable nor scalable. While the NSF may be trusted to scan laptops, requiring such exposure of personal or proprietary data may not be acceptable in more general settings.

Evolving access patterns only exacerbate the challenge of privacy-retaining certification. For example, wireless hotspots may require all hosts to be scanned for viruses prior to being permitted access. These and other procedures may legally indemnify a service provider from damages resulting from viruses or other malcode [15], [21]. Such requirements are totally appropriate and are likely to be observed more frequently in the future. However, allowing the proprietor of a local web cafe to run arbitrary scans and code on their customers' laptops, many of which likely contain sensitive financial information, is absolutely unacceptable. We argue that such tradeoffs need not be absolute. Specifically, the average user need not relinquish their privacy in the process of demonstrating correct adherence to security policy.

In this paper, we develop and evaluate a non-invasive host security certification procedure. As a means of demonstration, we introduce a protocol that allows a host to prove that it has properly configured, up-to-date, anti-virus software without any direct access to its internal state. Inspired by cryptographic zero-knowledge proofs [12], the network provides each client with a vector of randomly selected file blocks, which are either malcode or harmless placebos. The client is certified if it can identify which blocks contain malcode. Hence, the host proves the existence of correctly operating malcode detectors by demonstration. Any host that cannot differentiate malcode from placebos in the test vector is deemed unsafe and disallowed access. As is appropriate for the target environments, our model assumes that the user is not intentionally malicious, but may be an unwitting carrier of malware or may have outdated anti-virus software. We implemented our protocol using COTS malware detectors and tested it in a live network environment. The efficiency and provided assurance are evaluated, and we comment on the challenges and operation of our non-invasive, certification tools.

Note that our approach can also be generalized to handle many forms of security infrastructure—any security mechanism that is able to distinguish between normal and malicious behavior may be non-intrusively verified. The contribution of our work not only lies in the design of the protocol, but also in

[1] A formulation and definition of network access best practices is explicitly outside the scope of this work. Readers interested in guidelines in this area should consult CERT [5] or other professional security organizations.

the efficiency, flexibility, simplicity, and non-intrusive nature of its implementation.

The remainder of this paper is organized as follows: Section II offers a brief overview of the relevant related work; Section III defines adversary and protocol models; Section IV discusses the implementation of this system and examines the performance of this protocol; Section V discusses the results and the application of this protocol to real systems; Section VI provides concluding remarks and future avenues for this work.

II. Related Work

The authentication of users is often a necessary prerequisite for access in many public networks. For example, Needham and Schroeder presented a widely-used protocol to authenticate and initiate communications between two machines [20]. Kerberos improved upon this work by extending the protocol and building an authentication service upon it [28]. More recent research in authentication includes the use of smartcards [18], [26], [19] and biometrics [23]. Generally speaking, user authentication prevents unknown or unwanted entities from accessing a system. While this is necessary for some environments, it is not sufficient to determine the security of the joining host; authentication validates identity, but states nothing about the configuration or state of the authenticated party's host.

To address this problem, some networks require that each new client be scanned by an agent in the network. This requires running foreign code on the machine—a practice that both violates privacy and is also insecure. Eustice et al. use available software packages to verify the state of a mobile computer user in the QED protocol suite [9]. Users are first isolated from the network and subjected to a system scan. Systems failing the scan are required to update with patches or prompted to turn on required software. While this approach does not require the use of specialized hardware or software on the user's computer, it does raise privacy concerns[2].

Attestations of software are actively being explored as a solution for non-invasive host certification [11], [4], [25]. In one instance, the Trusted Computing Group's Trusted Platform Module (TPM) [31] vouches for the current state of a system by cryptographically certifying the precise operating system and application software running on a system. There is hope that through future features, the TPM will offer non-forgeable verification of a platform's ongoing safety. However, such features will not solve the security requirements of legacy devices. Moreover, even if features were available, they can only attest to the software that is running. Hence, they say nothing about their internal configuration of the host, i.e., the configuration of security mechanisms, freshness of virus signature definitions, etc.

Inspired by the vast body of literature in zero-knowledge proofs [12], [1], we take a different approach than the historical authentication, scanning, or attestation methods. Zero-knowledge proofs allow a prover to demonstrate knowledge of an artifact (typically secret information such as a key) without actually exposing it. Such proofs are typically implemented using considerable cryptographic machinery, and are used as the basis for many valuable constructions. Using an analagous approach, we ensure that the host to be certified (prover) demonstrates, with a high degree of accuracy, that it is running recent and well configured anti-virus software. This mirrors a zero-knowledge proof in that the host proves it is executing the software without allowing the certifying party any direct access to the software's execution environment.

Our approach to remotely ensuring one aspect of the secure configuration of a machine is complementary to past certification methods. Note that all of these above methods have advantages to their implementation. Attestations allow certification of the running software. Scanning by a third party affords deeper inspection and hence stronger validation of the current state of infection but says nothing about the platform's ability to protect itself against future incursions. Each of these methods is appropriate for a set of environments. None of them is sufficient for the environment that is becoming rapidly most common: public access points to the web. This is an environment of mutual distrust, in which service providers cannot fully trust their clients and just let anyone log in, nor can clients merely allow service providers full access to all their files. This environment presents the particular challenge that it should be accessible to all users and yet it should also not compromise the privacy of those users. Our central contribution is providing an additional tool for host certification which is appropriate for these environments of mutual distrust: this tool can certify the security of foreign host clients while also not violating their privacy.

III. Protocol Definition

In this section, we present our procedure for certifying the proper functioning of anti-virus software on a client machine. In the subsections that follow, we describe the adversary, state our design goals and define the protocol itself. We complete this section with a discussion about the selection of specific malcode.

A. Defining the Adversary

The success of the most damaging worms and viruses sweeping across the Internet has depended largely on their ability to quickly infect a multitude of unprotected machines [27]. Indeed, the number of platforms unknowingly spreading a digital contagion is far greater than the number of people who are actively and consciouly attempting to further its diffusion. Accordingly, the most prevalent (and arguably the most dangerous) threat to any network is the presence of a susceptible or infected platform. *Therefore, the "adversary" for whom this work is designed is the authorized, but unaware user.* More specifically, we assume that the user of the host is not intentionally trying to infect others. We require that the user be able to prove, with a high degree of certainty, at the point of certification, that she has a properly functioning security infrastructure. That is, we desire to prevent an adversary from claiming that it has an up-to-date, well-configured infrastructure when, in fact, it does not.

[2]Eustice, et al., [9] declare, "In some environments, safety must take precedence over privacy".

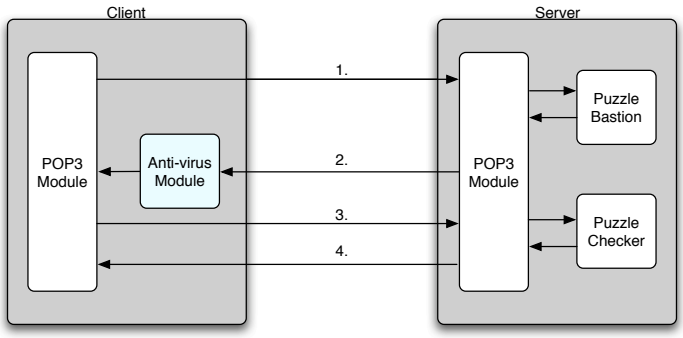

Fig. 1. The protocol for demonstrating a properly functioning anti-virus service begins with a client 1) sending a request to the server to join the network. The server 2) responds with a test vector, created by the test generator, which contains a random mixture of clean and infected files. It then awaits a response. After the client's anti-virus software intercepts and scans the test vector, the client 3) reports back to the server with a list of which files were clean and which were infected. The server 4) then admits or expels the client based on whether or not the client correctly identified the malcode in the test vector.

The threat model adopted in this work mirrors the standard model in this area. Much of the work conducted in this field has suggested the need for active scans of machines as they attempt to attach to a network [9], [33], [30], [29]. The goal of these scans is to detect and clean malcode from infected platforms. This is typically accomplished by examining the contents of all the files on a machine and comparing them against known malcode. Such measures only serve to protect the network from a machine at the time of certification – there are countless ways in which a truly malicious adversary could circumvent this defense. For example, the malcode intended for release can simply be encrypted or stored on removable media and therefore remain hidden from detection. Worse yet, there is nothing to prevent an adversary from successfully passing a full system scan and then downloading their malignant payload from a remote server. Full system scans by network administrators are simply unable to prevent attacks by a determined adversary.

To repeat, this protocol is designed to protect networks against well-intentioned users who *unknowingly* have machines that either are vulnerable to or already infected with malcode.

B. Design Goals

The design goals of this protocol are *preserving user privacy*, *flexibility*, *simplicity of use* and *efficiency*. In terms of privacy, this protocol allows for a client host to demonstrate that it exercises good security practices without requiring invasive searches and scans. Secondly, in order to impact the widest possible audience, such a solution must be flexible enough to remain agnostic to any specific anti-virus software suite. Thirdly, requiring active user participation in any protocol must be kept to a minimum in order for any solution to be accessible to the general public. This not only precludes requiring the user to respond to messages, but also must avoid mandating that a user install one specific software suite in order to be compliant. Finally, this protocol must perform efficiently.

The first two of these goals are achieved by the design of the protocol itself, which takes advantage of features available by default in the majority of available anti-virus programs. To support the remaining two goals, our software has been implemented as a Java applet. This applet appears on the terminal when the client attempts to log into a network via a web-based, access system and it hides all of the underlying functionality and messaging associated with the zero-knowledge client puzzle protocol. This applet displays the client's current status and alerts them of successful and failed attempts to attach to a network. We detail the cost and operation of this process below.

While we stress its applicability to web-based operations, this mechanism is not limited only to web-based systems. For example, in a network where authentication is not necessary, a DHCP server could run this process prior to assigning IP addresses. In a more controlled network, protocols such as 802.1X [6] could also be modified to include such a client puzzle access control mechanism prior to user authentication. Moreover, certification need not be used to regulate network access only; many providers of sensitive content may require some kind of certification before distributing data. Independent of the purpose or environment, the operation of the protocol will be quite similar.

C. Protocol Definition

We begin the discussion of this protocol by defining the notation used throughout.

Notation

- C, S are principals, specifically Client and Server.
- F_X is a file transmitted by S to C.
- $H(F_X)$ is the hash of the file F_X.
- I_N is the calculated Infection Index.
- N is a nonce.
- P is the number of files used per round.
- R is the number of rounds C must correctly pass to be admitted.
- $REQ_{J(C,S,N)}$ is a join request from C to S.
- $REP_{X(C,S,N)}$ is a join request response message from S to C, where X is either A for accept or F for failure.

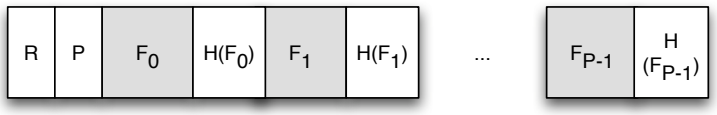

Fig. 2. A test vector message from the server to a request for network access. Each packet contains the number of successful rounds R needed for entry, the number of files P per test vector, and test files F_0 through F_{P-1} coupled with their corresponding hash values.

The certification protocol works as follows: A client attempting to attach to a network sends a join request to the server. The server responds by sending the client a test vector - a collection of files that must be scanned and categorized as infected or clean by the client. Note that in order to remain agnostic to any particular antivirus suite, we do not ask the client to report the name of the malware; rather, we use the repetition of detection as a means of assurance. The client responds to the server with a string representing the infected/clean pattern of the original test vector, which is compared against the pre-calculated answer at the server. If the two strings match, the client is admitted. If there is any deviation in the strings, the client is refused entry to the network. Figure 1 illustrates this protocol, which is described in further detail below.

A client wishing to attach to a network initiates communication by sending a join request, $REQ_{J(C,S,N)}$, to the server via the POP3[3] protocol. Because the protocol is run over POP3, the anti-virus module automatically scans all incoming packets. This is a feature available in all of the major free and commercially released anti-virus software suites. We take advantage of the fact that this mechanism is turned on by default in all of these products. We leverage this default setting to perform the necessary scanning of test vectors. (Methods of ensuring that the user maintain the proper security configuration during the entirety of the attachment period are discussed in Section V-B.)

As shown in Figure 2, the server responds to the request with a message containing the number of rounds R that must be passed for admission, the number of files P to scan per round and a test vector. The puzzle itself is assembled by the puzzle creator. Each vector consists of P files, F_0 through F_{P-1}, and their corresponding hash values, $H(F_0)$ through $H(F_{P-1})$. A bit string, I_N, representing the infection status of the files is then created and kept for the confirmation of the client's response. For example, a sample puzzle containing five files may be represented by 10110, where 1 and 0 represent infected and clean files, respectively. The test generator then batches the components mentioned above, minus I_N, into a MIME message and forwards the data to the client. We examine the performance of this protocol in Section IV.

When the client receives the response from the server, the anti-virus module is automatically triggered by the use of the POP3 protocol. Only after the test vector has been completely scanned by the anti-virus module (and declared safe) is it delivered to the client's POP3 module. Note that because the anti-virus steps in between the client and server while infected files are in use, there is no risk of contamination of the client's machine. Additionally, because the files themselves are never executed, the client's platform is at no risk of being infected. To prevent damage, in the case that anti-virus software is turned off, the buffers containing the files themselves are zeroed after each iteration. Because files that contained malware have been altered (cleaned or deleted) from their original state by the anti-virus module and therefore no longer match their original hash, determining the infection status of each of the files in a vector is as simple as comparing two hash values of the files before and after the anti-virus detection procedure is executed.

[3]Note that our use of POP3 is tailored to current anti-virus tools. Other protocols, e.g., HTTP, SMTP, could be readily used for our purposes to the same effect, save that other anti-virus mechanisms would be used to vet the incoming data and identify malware.

1) $C \rightarrow S : REQ_{J(C,S,N)}$
2) $S \rightarrow C : R, P, N, F_0, H(F_0), F_1, H(F_1), \cdots, F_{P-1}, H(F_{P-1})$
3) $C \rightarrow S : REP_{C,S,N}, I'_N$
4) $REQ_{X(C,S,N)}$

Fig. 3. The challenge/response protocol used to determine the presence and proper functioning of anti-virus software on a client's platform.

As is done in the test generator, the client creates the corresponding Infection Index I'_N for the received files. The Infection Index is returned to the server, which passes the client's solution to the Answer Checking module. The module performs a lookup for the Infection Index calculated by the puzzle creator and compares the two values. A success or failure message is returned to the server's POP3 module. The module then either sends the next puzzle, if necessary, or informs the client of its correct response and allows it to enter the network ($REP_{A(C,S,N)}$) or reports the client's failure and ends the session ($REP_{F(C,S,N)}$).

D. Selecting Malcode

Without the proper selection of malcode for a specific environment, the value of this protocol is limited. A carefully chosen sampling of viruses, however, can be used to demonstrate that a system is protected against the most critical digital pathogens with a given assurance. The challenge in selection comes in ensuring that the files sent to a client are not only representative of the malcode most likely to affect a specific network, but also that the chosen malcode appears sufficiently random so as to make guessing the infected files extremely difficult.

We begin by first defining the term *assurance*. As the

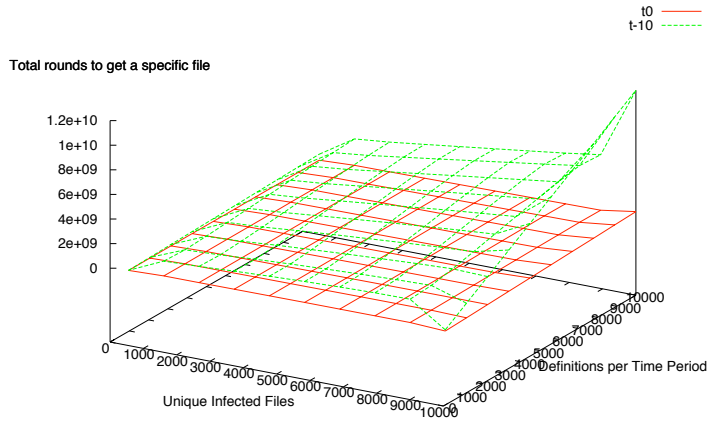

Fig. 4. Examination of the tradeoffs between the number of viruses per time period (based on a geometric distribution), the number of unique files in which each virus is manifest and the total number of rounds necessary to encounter a given file twice. In such a system, the frequency of replay is inversely proportional to the space allocated for storing test files.

number of files in a test vector increases, a client correctly classifying an entire test vector is probabilistically more likely to be running anti-virus software. The probability that a client is able to guess the status of infection for an entire test vector therefore decreases exponentially with the size of the test vector. For example, while a client presented with a single file can correctly guess whether or not the test file is infected with a probability of 0.5, a client receiving 10 files is only able to guess the correct Infection Index with a probability of $9.76 * 10^{-4}$.

Throughout the remainder of this text, we refer to the level of assurance according to the standard engineering metric of "nines". The example client above that correctly categorizes all 10 files in the test vector is said to be running anti-virus software with three-9's (0.999) probability. Assurance is simply $1 - Pr[GuessingCorrectly]$.

Using Equation 1 below, it is possible to determine the number of files P necessary to achieve a desired assurance. Assurance levels of three-, six-, and nine-9's are accordingly achievable through the use of 10, 20 and 30 files respectively.

$$P = \left\lceil \frac{\log\left(1 - AssuranceLevel\right)}{\log\left(.5^{R}\right)} \right\rceil \qquad (1)$$

We now explore various methodologies for selecting specific instances of malcode for use in test vectors. An important observation in selecting malcode is that the majority of machines connected to the Internet are eventually patched or upgraded. That is to say, malcode becomes less effective in infecting machines over time. The malcode that wreaked havoc a decade ago is highly unlikely to be able to establish a foothold in today's machines. While it is impossible to assume that these malicious programs ever completely vanish, the probability of being infected with a given piece of malcode decreases as a function of time. Additionally, because of the rapid spread of modern malcode [27], the probability that a machine is infected with one of any number of recently released digital pathogens is much more probable.

Time is therefore a natural metric for pathogen selection; newer viruses are more likely to be encountered in the wild, and hence it is desirable to test protections against them more frequently. From the discussion above, however, simply selecting the most recently released malcode or randomly selecting them from a uniform distribution is not sufficient. A more robust system achieves both recency and breadth by choosing malcode based upon a number of realistic models of decay. The most frequently used models for both digital and biological pathogen lifetimes utilize exponential [2] and geometric [10] decays.

In addition to the choice of distribution, the effects can also be tuned by adjusting the size of time periods. Most commercial anti-virus programs, for example, release weekly updates of malcode definitions. Accordingly, the smallest time period for time-based models in our system is one week. This level of granularity is helpful for ensuring that client platforms are updated to the most recent set of definitions. Because the number of viruses per update is limited, extending the length of time periods may instead prove more valuable as the frequency of replay (and therefore the ease of guessing) is decreased. The tradeoff between recency and replay must therefore be carefully balanced. Figure 4 illustrates that the tradeoffs between recency, breadth, resistance to guessing and storage can be set depending on the specific distribution used.

Time is a natural metric, but it is not the only means of calibration. For example, it may also make sense for an administrator to base their defenses around virulence. In this case, test vectors should include tests for more dangerous viruses with the greatest frequently. If the majority of malcode in recent virus signature updates do little damage to actual systems, it is difficult to justify that these pieces of malcode should be tested for more frequently. Accordingly, our tool could be tuned to take advantage of virulence classifications provided by many anti-virus software providers. Using this information, the proper statistical distributions could be formed and a network could be fine-tuned to prevent the most

TABLE I
MICROBENCHMARK RESULTS FOR ZERO-KNOWLEDGE CLIENT PUZZLES.

Operation	$\bar{x}$ Time (msec)	σ
Network	0.1	1.9
Anti-virus Scanning	425.2	48.2
MIME Parsing	1.0	3.5
Hashing	1.3	3.8
Total Time	431.5	58.9

catastrophic malcode with higher assurance. Continuing to maintain a broad sampling of test files case helps to maintain breadth as well.

Another approach that could be used is based on the protection of specific services. Networks providing particular services to customers may choose to protect against malcode specifically designed to interrupt that service. For example, a network composed primarily of Microsoft SQL Servers may be primarily concerned with blocking exploits designed to attack this application. Like the previous techniques, a system tuned in this manner should still provide breadth against other attack vectors, as the primary application running on these systems may not be the only means of exploiting vulnerabilities.

Of course, all of these approaches can also be used in concert to provide additional protections. A system based on time could subdivide each quanta into distributions based on virulence. In turn, a system built to protect primarily against a particular family of attacks could select these exploits most frequently, but fill in the remaining test vector slots based on a temporal method. These additional protections come at the additional cost of classification, setup and maintenance.

The tuning of resilience must be carefully considered and set by the administrator of any system to accurately protect against the most relevant adversary.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the efficiency of the certification process. We begin with some notes on the construction of the tools. The Java programming language was chosen to implement the protocol, allowing us to execute the entire process in a Java sandbox. Because the overwhelming majority of viruses are written for Windows-based systems, we believe that demonstrating the performance of this procedure on a Windows box is essential. Accordingly, the server runs on a 1.6 GHz Pentium 4 Windows XP Pro version 2002 SP1 with 256 MB RAM. The client runs on a 1.6 GHz Pentium 4 Windows XP Pro version 2002 SP1 with 256 MB RAM running Symantec Norton Anti-virus. We used a Netgear 10/100 Dual Speed Hub between the server and the client on a 100MB/sec network. To prevent the server's resident anti-virus software from overzealously cleansing the infected virus test files, the server reads the virus files off of a CD-ROM. The viruses themselves averaged between 2KB and 4KB in size. A test vector containing 30 "puzzles" can therefore easily be delivered using approximately 100KB - a relatively insignificant amount of bandwidth for the majority of access points [4].

[4]Wireless networks running 802.11b or g protocols can transmit at rates of multiple Mbps – beyond sufficient bandwidth to support our protocol

For completeness, additional tests of the client program were conducted on a 930 MHz Pentium 3 SUSE Linux Box with 256MB RAM running RAV AntiVirus. Because the results on both systems were similar, the results discussed below reflect only the experiments conducted on the Windows machine. The implementation of the protocols on the client and server occupied approximately 80KB of disk space. Notably, the programs required no additional software to be installed prior to joining a network and no caching between certifications was necessary. In accordance with our design goal of efficiency, the software had no persistent on-disk or memory footprint for the client and so required no sustained resources on the host. For increased safety, we could relax these requirements slightly and introduce tickets, as discussed in Section V-B.

A. Microbenchmarks

The results of our microbenchmark tests of this system are shown in Table I. Each of the 10,000 iterations included a single round consisting of a test vector containing 30 files. This test vector corresponds to nine-9's assurance that a client is running anti-virus software. The microbenchmarks measure the time spent on network transmission (*Network*), scanning (*Anti-Virus Scanning*), MIME encoding and decoding (*MIME Parsing*), and MD5 hashing of the received and potentially cleaned files (*Hashing*).

As expected, the dominant factor in the execution of this protocol was the scanning of incoming files. This activity, responsible for over 98.5% of the execution time over the average 431.548 milliseconds needed to execute this protocol, takes two orders of magnitude more time than all of the other operations combined. The observed variance can be attributed to the different file sizes in each puzzle and transient network and operating system events.

B. Macrobenchmarks

While the microbenchmarks tested a 30-file test vector in a *single* round, there are a number of compelling reasons for executing such a protocol across *multiple* rounds. For example, because it is significantly faster to transmit files than to scan them, it may be more efficient to issue puzzles with fewer individual files and test them in smaller batches over a larger number of rounds. In this way, we could maximize performance by exploiting the relative speeds of transmission and scanning to achieve a pipe-lining effect. Moreover, such a mechanism could decrease the total number of files transmitted because a failure would short-circuit the transmission of the remaining files. In testing this hypothesis, however, we discovered that this was not an effective approach.

Figure 5 shows the correspondence between execution time and the number of rounds required for the delivery of 30 files in the puzzle (thereby giving nine-9's assurance). Each data point was collected from 1,000 iterations of the protocol. The observed increase in time with respect to the number of rounds is essentially linear. As demonstrated above, the dominating factor responsible for this behavior is the scanning of incoming test vectors. Further investigation revealed that the cost of

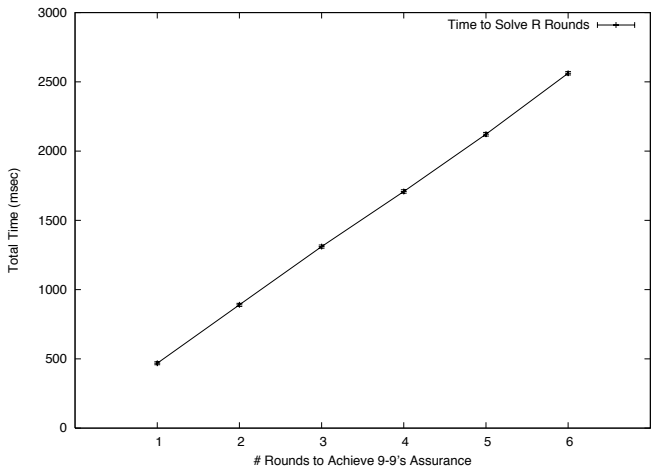

Fig. 5. The amount of time required to yield nine-9's assurance (30 files total) over a varying number of rounds. Note that even though the number of files scanned per round is inversely proportional to the number of rounds, the increase in rounds is highly linear.

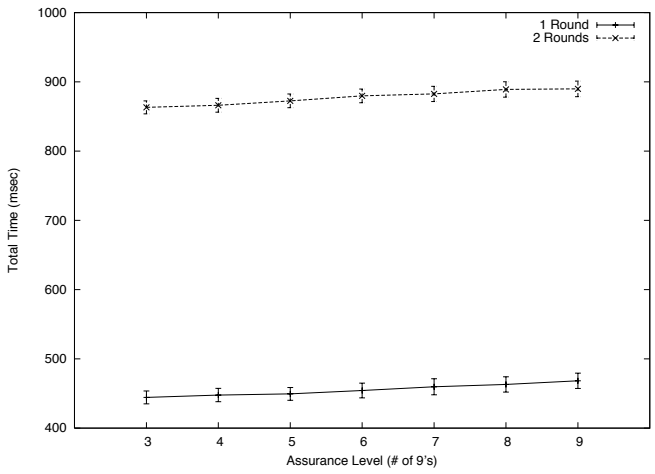

Fig. 6. Macrobenchmark performance results for varying levels of assurance. As is observable in Figure 5, the difference between a single versus multiple rounds of puzzle solving is dominated by the anti-virus scanning portion.

scanning files is dominated by the start-up cost of the scanner. That is, the scanner is not memory-resident and a new process is started each time a new batch of files is received.

Figure 6 demonstrates further evidence of the startup cost dominance. The number of files, and therefore the level of assurance, is varied between three and nine-9's. As the level of assurance increases, so too does the slope by an average of 3.997 milliseconds. The average slope in Figure 5, however, averages 418.769 milliseconds. The actual scanning of files therefore has little effect on the overall time required for the protocol to operate; rather, the running time is predominantly determined by the overhead associated with starting the scanner at the beginning of each round.

Although it is clearly less efficient, there may still be reasons that it desirable to use multiple rounds. For example, it may be undesirable to force the server to deplete resources in order to access more files than necessary. If a system can be proven insecure, it saves server resources to discover this with smaller test vectors. Fortunately, there are methods that can be used to decrease the reliance upon smaller test vectors while still preventing resource exhaustion (i.e. puzzle depletion) at the server. Waters, et al. [32], for example, study a similar problem for cryptographically-based client puzzles and are able to leverage the use of a "bastion" to generate a high number of puzzles on behalf of the server. However, the cost of assembling puzzles is likely to be sufficiently low that requiring off-line puzzle generating bastions may not be warranted in all but the most active areas.

While explicitly outside our threat model (see Section III-A), a malicious adversary may try to circumvent the system by extracting and storing test vectors[5]. Specifically, the ability to determine which guesses are correct and which are incorrect increases dramatically as the size of a given test vector decreases. For a test vector with a single file, for example, a client will *always* learn whether that file was malcode. This realization, combined with the results regarding efficiency gives compelling evidence that the number of rounds used should be kept low. At the same time, we leave our system flexible enough that this parameter could be adjusted according to the needs of a particular network.

Similarly, an adversary who promiscuously listens to the protocol exchanges of many joining hosts may create a dictionary of puzzles by looking at the server-provided puzzles and client-provided responses. Hence, that adversary would be able to produce correct answer with some probability dependent on the number and diversity of exchanges they eavesdropped. Again, such adversaries are specifically outside our threat model, but we note that any environment wishing to thwart such attacks need only use some transport level security protocol such as TLS/SSL [7], [14]. Such protocols are currently supported by most mail applications, and hence their integration into the process described above should be straight-forward.

V. Discussion

In the remaining subsections, we compare the performance of our implementation to the work of others in this area, discuss methods of increasing the security guarantees provided by such a protocol, examine the inherent safety issues to this particular implementation and conclude with a sampling of additional applications to which a similar approach could be applied.

A. Certification Performance

Comparing our approach with other certification processes is not straight-forward. For example, performing a full scan of a system as is suggested in Eustice, et al. [9] may take time on the order of tens of minutes and is highly dependent upon the amount and content of disk space occupied by a client. By contrast, our protocol takes a relatively short, fairly

[5]Such behavior may be exhibited by an adversary unwilling to purchase antivirus software.

constant amount of time to execute. Of course, the goals and level of assurance provided by each approach differ vastly, so any performance comparison would be of limited value.

Client-puzzles enforcing rate limitation on clients [17], [32] can be varied in difficulty and be made to last anywhere between minutes and seconds. Our protocol, too, can increase the difficulty required to gain access to a system; however, the orders of magnitude between rate limiting solutions and process verification are necessarily different. Attestation-based solutions, which may offer the most comparable solution, also cost up to tens of seconds [3], [24], even when implemented in hardware, e.g., TPM.

B. Safety

While the protocol presented thus far demonstrates, for a required level of assurance, that a client is running anti-virus software when logging on to a network, there is no guarantee that the client will continue to do so after the initial test. A user may disable the anti-virus software because of the observed, reduced performance some platforms experience while running anti-virus software. If clients only turn on their anti-virus software initially, they fail to protect themselves during the most critical period—when they are actually connected to the network. Similarly, if a client remains logged in, but never updates his anti-virus software, the prophylactic advantage of scanning will severely diminish over time.

User-initiated disabling of anti-virus software can be mitigated via a ticket-based extension to the proposed protocol. After successfully completing a set of test vectors, a client's platform could be issued a ticket as is done in authentication systems such as Kerberos [28]. At a time close to the expiration of this ticket, the client would signal the server responsible for vetting clients to resend a new set of test vectors. Should a ticket expire, a client would be denied access to the network until such time when it could demonstrate the continued presence of properly functioning anti-virus software.

The tests conducted during the benchmarking section of this paper give insight into the realistic granularity of ticket lifetimes. Given that it is possible to admit a machine into a network with 9-9's assurance in under half a second ($\bar{x} = 0.431$ sec), requiring attached clients to reaffirm that they are running current anti-virus software once every five minutes would amortize to approximately 0.144% of a machine's resources. Reaffirming the service once every minute would similarly require only 0.718% of the available system resources over time. Moreover, such assessment could be made totally transparent to the user. Depending on the user's tolerance for turning on and off their anti-virus software, the vast majority of users would be persuaded to constantly run such a program, thereby helping to better secure the network.

The use of the ticket-based mechanism also increases the effort that must be expended to bypass this system. The non-malicious adversary who is unwilling to install the requisite antivirus software would be forced to collect an enormous number of malware puzzles and their answers in order to *maintain* connectivity. Assuming a sufficiently large database of malcode, the cost associated with circumventing this protocol quickly eclipses the cost legitimately interacting with the system.

An additional method of increasing the safety of the network would be to redirect clients failing the admission test access to local repositories of anti-virus updates. Such default behavior would allow users the opportunity to bring their platform up to a required safety specification and then join the network. Such a model would be particularly beneficial to systems such as hot-spots that require users to log on before revenue can be collected.

C. Safety Issues

One of the largest issues facing the implementation and widespread distribution of this work is that of requiring average network administrators to maintain large databases of malcode. These repositories, while extremely valuable as a means of vetting protected users, could potentially be used as arsenals of weapons against known populations of unprepared machines. An adversary could request some number of puzzles and store all of the infected files received in the transaction. After amassing a sufficiently large cache of malware, the adversary could then attach to another network not implementing the same precautions and release its newly accumulated digital arsenal.

On reflection, however, we can see that this is not a serious threat. Dedicated adversaries do not need to use this system in order to obtain a cache of malcode. One could simply set up their own honeypot and catch a sufficiently large number of digital pathogens. This approach creates a potentially more dangerous stockpile as many of the viruses and worms found by honeypots are previously unseen. An attacker could also employ the same method that we exploited - use search engines and track down websites where malcode is distributed. Because malcode is so widespread and so easy to locate and because we take great care in our design to prevent client infection, the means through which we certify clients is no more dangerous than allowing them to connect to the Internet.

If this is still considered problematic, we note that there are also alternatives to real viruses. The EICAR test virus [8], for example, is a non-malicious piece of code used to demonstrate the correct scanning ability of anti-virus software without risking a real infection. The use of such test viruses has direct parallels to the use of vaccines in biological immune systems, in which case crippled or dead pathogens are introduced to the body so as to help build defenses without accidentally causing infection.

Very little research has been conducted into the creation of inoculated malcode. While some headway has been made in terms of creating realistic test viruses [13], many open problems remain. If, like in the world of immunobiology, the removal of the mechanism responsible for infection became simple, is seems likely that digital immunology could become a fruitful area of research.

D. Generalized Certification

The preceding sections have focused on a demonstration of the certification procedure that measures the presence

and correct operation of anti-virus software. This technique, however, is not limited strictly to this application. For example, some networks perform a similar certification process by periodically scanning for open ports using the `nmap` utility [16].

One could also use our approach to vet implementations of essential algorithms: determining that a machine is using robust implementations of cryptographic algorithms is possible by using a similar procedure. For example, a server could provide a client with a number of preimages with which to create keys, ciphertext, signatures, or HMACs. Signatures (in the non-cryptographic sense) of known, weak implementations, such as poor sources of randomness, could be observed depending on the client's response. Vulnerabilities in secure transmission suites such as SSH [22] and a variety of VPN clients could also be discovered in a similar fashion. In networks where the transmission of highly sensitive data is critical, such proofs of robustness would be extremely valuable.

In essence, any program providing security services that provides a demonstrable result can be polled and fingerprinted through this technique. This work serves as a blueprint for implementing these tools. In the hopes of encouraging others to build similar infrastructure, we provide source-code and documentation for our certification framework via the adress below:

`http://siis.cse.psu.edu/tools/av-tools.html`

VI. Conclusion

This work has considered the problem of non-invasive host certification. In so doing, we have asked, "How do we ensure that a host joining a network is following the proper security practices?" Determining whether a user or system is exercising appropriate security practices is difficult in any context. Commonly practiced techniques used to vet hosts, such as system scans, have the potential to infringe upon user privacy and do not necessarily indicate the user's ability to protect themselves. Other certification approaches such as attestations provide limited insight into software state—hence, they do not enable an appropriate level of certification of host configurations.

We have shown that it is possible for clients to prove the presence, proper functioning, and configuration of security infrastructure without allowing unrestricted access to their system. We apply this approach to certify that hosts are properly using up-to-date anti-virus software. Users are given a vector of small files that may or may not contain malware. A host is certified if it can correctly identify the presence of malware in the test vectors. We have described our implementation and provide and demonstrate the feasibility of this work through performance analysis.

Our future work will seek to examine other means of employing this approach. In particular, we will explore how our zero-knowledge proof inspired protocols can be used to certify more diverse security infrastructure. In the end, when combined with other techniques, such mechanisms may provide the kinds of certification desperately needed by contemporary networks.

Acknowledgements

We wish to acknowledge the support of CISCO through the University Research Program (URP). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of this sponsor.

References

[1] Hannu A. Aronsson. Zero Knowledge Protocols and Small Systems. `http://www.tml.hut.fi/Opinnot/Tik-110.501/1995/zeroknowledge`, 1995.

[2] M. Bailey, E. Cooke, F. Jahanian, D. Watson, and J Nazario. The Blaster Worm: Then and Now. July 2005.

[3] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, New York, NY, USA, 2004. ACM Press.

[4] A. Buldas, P. Laud, and H. Lipmaa. Eliminating counterevidence with applications to accountable certicate management. *Jounal of Computer Security*, 10(3):273–296, 2002.

[5] Computer Emergency Response Team (CERT). `http://www.cert.org`.

[6] P. Congdon. RFC 3580 - IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines, 2003.

[7] T. Dierks and C. Allen. The TLS Protocol Version 1.0. *Internet Engineering Task Force*, January 1999. RFC 2246.

[8] European Institute for Computer Anti-Virus Research. eicar - Anti-Virus test file. `http://www.eicar.org/anti_virus_test_file.htm`, 2003.

[9] Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald Popek, V. Ramakrishna, and Peter Reiher. Securing Nomads: The Case for Quarantine, Examination, and Decontamination. In *NSPW '03: Proceedings of the 2003 workshop on New security paradigms*, pages 123–128. ACM Press, 2003.

[10] M. Garetto, W. Gong, and D. Towsley. Modeling Malware Spreading Dynamics. In *Proceedings of IEEE INFOCOM*, 2003.

[11] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 193–206, 2003.

[12] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, New York, NY, USA, 1985. ACM Press.

[13] Sarah Gordon. Is a good virus simulator still a bad idea? `http://www.research.ibm.com/antivirus/SciPapers/Gordon/Simulators.html`, 1995.

[14] The OpenSSL Group. OpenSLL, May 2000. `http://www.openssl.org/`.

[15] N. Harris. Securing network will help business owner mitigate legal liabilities. `http://www.bizjournals.com/houston/stories/2004/01/19/focus5.html`, 2004.

[16] Insecure.org. Nmap - Free Security Scanner For Network Exploration & Security Audits, November 2005. `http://www.insecure.org/nmap/`.

[17] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, 1999.

[18] T. Kato, T. Tsunehiro, M. Tsunoda, and J. Miyake. A secure flash card solution for remote access for mobile workforce. *Consumer Electronics, IEEE Transactions on*, 49:561–566, August 2003.

[19] T. Kawase, A. Watanabe, and I. Sasase. Proposal of Secure Remote Access Using Encryption. *Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration. IEEE*, 2:868–873, November 1998.

[20] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21:993–999, 1978.

[21] F. Olsen. The Growing Vulnerability of Campus Networks. `http://chronicle.com/free/v48/i27/27a03501.htm`, 2002.

[22] OpenSSH. `http://www.openssh.com`.

[23] M. Rahman and P. Bhattacharya. Remote Access and Networked Appliance Control Using Biometrics Features. *Consumer Electronics, IEEE Transactions on*, 49:348–353, May 2003.

[24] Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. Attestation-based policy enforcement for remote access. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 308–317. ACM Press, 2004.

[25] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 223–238, 2004.

[26] D. Scheuermann. The smartcard as a mobile security device. *Electronics and Communication Engineering Journal*, pages 205–210, October 2002.

[27] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the USENIX Security Symposium*, 2002.

[28] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: An Authentication Service for Open Network Systems. *USENIX Winter*, 1998.

[29] Sygate Website. Sygate Secure Enterprise. `http://www.sygate.com/products/sygate-secure-enterprise.htm`.

[30] Symantec. Symantec Client Security. `http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=154`.

[31] Trusted Computing Group. `http://www.trustedcomputinggroup.org`.

[32] B. Waters, A. Juels, J.A. Halderman, and E. Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2004.

[33] Zone Labs. Zone Labs Integrity SecureClient. `http://www.zonelabs.com/store/content/company/corpsales/secureClientOverview.jsp`.