A Software Authentication System for the Prevention of Computer Viruses

Lein Harn* Hung-Yu Lin* Shoubao Yang**

* Computer Science Telecommunications Program
University of Missouri-Kansas City, Kansas City, MO 64110 USA
E-mail: harn@cstp.umkc.edu

** Department of Computer Science
University of Science and Technology of China, Hefei, Anhui 230026 PRC

Abstract

In the absence of systematic techniques to detect the existence of computer viruses, preventing suspicious software from entering the system at the initial point of entry appears to be the best method to protect computing resources against attacks of computer viruses. Currently, software is distributed primarily by diskettes instead of on-line transmission. Diskettes are more susceptible to modification and masquerading while on-line transmission usually follows proper user/message authentication. A software authentication system is proposed which does not require a mutually trusted center of both software vendors and users, or users' interaction with any key center. Vendors assume responsibility by signing released software and users verify the authenticity of received software before using it. Through such an authentication process, users eliminate the risk of running "unlicensed" or modified programs, thus eliminating the possibility of virus infections.

I. Introduction

In recent years, computer viruses have become major threats to the computing environment. Many anti-viral products to protect systems from infections are available on the market. These products do help users protect their computers from viral attacks to some extent by either monitoring running programs or by identifying certain patterns in infected programs.

However, a satisfactory solution to prevent virus attack has not yet been reached. Formal secure models [1, 2, 3] may help to reduce the possibility of a virus infection by logically isolating or limiting access to computing resources. However these methods are usually too complicated and expensive to implement. Also, they unnecessarily limit the sharing of computing resources.

Currently, the prevention of virus infection relies on the users' experiences. Vendors are not taking active roles in preventing their released programs from virus infections. Without vendor involvement, virus problem cannot be solved in any acceptable manner.

In this paper, we will review three different anti-viral approaches and their associated problems. Then, a software authentication system, which requires a software vendor to sign his released programs and provides users an easy way to verify the authenticity of the programs, is proposed along with the discussion of several practical considerations.

II. Review of Anti-viral Approaches

Recently, three different anti-viral approaches are well discussed in literatures. Here we review each of them together with their associated problems.

1. Pattern-matching

This is the most straightforward method and widely used in many commercial anti-viral products. It searches for code sequences (virus signatures, which may be those used by the virus itself to avoid reinfection) that are known to exist in programs infected by a specific strain of virus. Once a code pattern is matched, a certain virus is detected. However, it cannot detect any new/unknown virus. Whenever there is a new virus, the anti-viral products also need to be upgraded. Such upgrades result in the increase of program size and the downgrade of its performance. When the number of viruses increases beyond some point, this approach will become impractical.

2. Software integrity-checking

Based on the fact that detecting a virus (especially an unknown one) is difficult [4], and a program will always be modified after being infected by a virus, software integrity checks (more specifically, the checksum method [5]) turns out to be a promising and cost-effective approach in detecting potential virus.

The checksum algorithm is carried out by executing a checksum generator utility over the program resulting in a checksum value which is then stored along with the program for later reference. Any change in program content will be reflected in a change of the checksum value regenerated later. By comparing this regenerated checksum value with the pre-stored one, we can detect any modification of the program. However, one implicit but dangerous assumption has been made in the realization of this method. It assumes that all counterparts -- original program, checksum value and the checksum utility itself -- are virus-free. This dangerous assumption usually comes from our common belief that software purchased from well known vendors is trustworthy. Is this common belief justified?

Consider the case of software counterfeiting, where one sells a fake program claimed to be the one from a famous vendor with its checksum value computed according to the publicly known checksum generating algorithm of the vendor. The user will falsely believe that this program is "clean" after checksum verification, even when this fake program contains a Trojan horse or is already virus-infected. Another possible situation is that the virus-infected program is indeed written by a malicious employee of the vendor who carelessly distributes it. How can one make a clear distinction between these two cases, and who should be responsible for the damages caused to the user? Unfortunately, most solutions in this approach cannot answer this question because the released software contains insufficient information for the user to verify the authenticity of the software. Without proper authentication one can never be sure that a received program is indeed from the original licensed vendor and therefore it has no way to resolve the dispute, if any, between a software vendor and a user.

3. Software authentication

In order to overcome the problems associated with the previous approach, the software authentication method requires software vendors to sign their released programs and users to accept only genuine programs by verifying the signatures of received software.

Although a public-key cryptosystem can provide digital signatures for the released software of vendors, its implementation is not so simple and straightforward as mentioned in a recent paper [6]. In that paper, a method was also proposed to realize software authentication in which it assumes the existence of a mutually trusted center of both software vendors and users. This assumption is hardly acceptable when there are many participants in a large commercial group [7].

Besides, if a mutually trusted center does exist, any software vendor needs to go through a registration process of this center to become a "licensed" vendor. In order to preserve the overall security, this mutually trusted center should be completely destroyed after issuing all secret keys to software vendors. Thus, any new software vendor cannot be registered after the center is destroyed. This situation is also totally unacceptable. Vendors are established on an almost daily basis. Therefore, this center can never be destroyed, and it leads to the risk that once the registration center is compromised, one can pretend to be a licensed software vendor to distribute virus-infected programs or Trojan horse. Under this situation, should software vendors be responsible for the damages caused by these programs? Thus, no vendor wants to risk their reputation by "trusting" someone who cannot be really trusted.

III. The Proposed Software Authentication System

In order to alleviate the worry of software vendors, the software authentication system should allow vendors to sign programs with their own secret keys. In order to make the verification of an acquired program as convenient as possible, the verification process should be performed locally by the user only without interacting with any registration or key center. With these considerations in mind, a software authentication system is proposed here.

1. System Components

In this system, there are four major components:

a. software vendors who sell software.
b. users who import software from software vendors.
c. the secret-generating machine(SGM) which generates one pair of public and secret keys and breaks the secret key up into pieces to be held by several certification centers.
d. certification centers(CC's) which are accredited organizations selected by the users and each provides a partial certificate for the signature system of a vendor.

The SGM and the certification centers do not have to be trusted by vendors and the SGM can be destroyed after distributing secret keys to certification centers.

2. System Initialization

a. Based on RSA cryptosystem [8], the SGM generates a small public key e, the corresponding secret key d, and the modulus n. Then it breaks d up into t shadows, d_j, j = 1 to t, such that

$$d_1 + d_2 + \ldots + d_t = d$$

and distributes d_j to certification center CC_j secretly. The SGM publishes e and n, and is then destroyed.

b. Each vendor i chooses a signature system and a pair of public and secret keys (a_i, b_i), and then has his $\{ID_i, S, a_i\}$ registered with the proper authorities. ID_i, S and a_i are the identification, signature function and public key of vendor i, respectively. The registered information will serve for the purpose of jurisdiction to resolve disputes, if any, between vendors and users. For convenience, we will say a vendor is licensed if he has done this registration, otherwise, the vendor is unlicensed.

The vendor also sends $\{ID_i, S, a_i\}$ to certification centers for certificating. After verifying that vendor i is licensed, each certification center will return back a partial certificate

$$c_j = \langle ID_i \| S \| a_i \rangle^{d_j} \bmod n$$

to vendor i, where $\|$ denotes concatenation and $< >$ is used as an integer value derived from the ASCII string. Now vendor i can compress all the partial certificates and results in a single certificate of his signature system, $cert_i$, by computing

$$cert_i = c_1 * c_2 * \ldots\ldots * c_t = \langle ID_i \| S \| a_i \rangle^{d} \bmod n.$$

Note that such certificate can be easily verified by using the public key e.

3. Distribution of software

Now if a program P is to be released by vendor i, it will consist of four parts:

$$\{ID_i, P, sign_i(P), cert_i\},$$

with

$$sign_i(P) = S_{b_i}(h(P))$$

where $sign_i(P)$ is vendor i's signature of P, S_{b_i} is the signature function S using vendor i's secret key b_i, and h is a publicly known one-way hash function, which maps programs to a smaller domain to speed up signature generation and verification.

4. Authentication of software

When a user acquires $\{ID_i, P', sign_i'(P''), cert_i\}$ from vendor i, he can derive vendor i's identification, signature function identifier, and public key through the system's public key e by computing

$$\langle ID_i \| S \| a_i \rangle = cert_i^{e} \bmod n.$$

If the result does not contain ID_i and a meaningful signature function identifier, the program P' should be discarded. If it does, the user will obtain the signature function, S, and the correct public key of vendor i, a_i. This process is called key certification. If $cert_i$ passes this process, the user can continue to compute the hash value of P', h(P') and verify the signature $sign_i'(P'')$. If $sign_i'(P'')$ has been verified (i.e. $h(P')=S_{a_i}(sign_i'(P''))$), program P' is indeed from vendor i without being modified. Otherwise, either P' or $sign_i(P'')$ has been modified. Again, this process is called signature verification.

IV. Discussion

We would like to discuss this software authentication method from three different aspects.

1. Security

From the properties of public-key cryptosystems, only the vendor himself can either generate the signature or modify a program without being detected. Therefore, no one else can forge a licensed vendor's program and no vendor can evade the responsibility for his released programs.

From a vendor's viewpoint, he trusts his own signature system and nothing else. In other words, a licensed vendor does not have to be responsible for programs which are not released by himself, even when all of the certification centers are compromised, since one can never pretend to be a licensed vendor to sign programs without knowing the secret key b_i, which is only known to the vendor.

Although an unlicensed vendor may create a consistent pair of P' and $sign_k'(P'')$ with a chosen signature system and related public and secret keys, he still cannot pass the key certification process because he won't get the proper certificates from certification centers. By employing multiple certification centers instead of only one, it is impossible for any unlicensed vendor to generate a valid certificate, even when only one certification center remains confidential and the other t-1 ones are compromised. From the user's viewpoint, the system is secure unless all of the certification centers are compromised.

Even though all certification centers could be compromised, nevertheless, users can still choose a new set of certification centers with newly generated shadows of the system's secret key d before such disaster really happens. This work can be done by constructing a simple circuit which accepts old d_j's from each certification center and outputs a new set of shadows to the newly chosen certification centers.

2. Flexibility

By separating key certification and signature verification in the authentication process, software vendors can change their signature systems for security considerations without introducing any change in current key certification process. The only thing vendors need to do is to get new certificates from the certification centers.

For the same reason, users can choose another set of certification centers if necessary without changing the vendors' current certificates as mentioned in the above subsection.

3. Performance

In this proposed system, software authentication consists of two processes: key certification and signature verification. No third party is involved in these two processes and the interaction with certification centers happens only when the certificate is requested by a new vendor.

For any vendor, before releasing any program, all he needs to do is to sign this program without having to recompute his certificate. For users, the time required for the key certification will not be long because the system's public key can be made as small as possible (e.g. e=3). Again, if vendors' own signature systems are RSA-based, their public keys can also be made as small as possible to reduce the time required for signature verification. Therefore, the authentication of software in this system is quite efficient. In fact, software authentication is required only when a new software is acquired from foreign sources, so the speed of verification will not become a critical issue.

V. Conclusion

A software authentication system is proposed in this paper, in which users can verify the signatures of a program when first acquired from a software vendor. Through such authentication, suspicious programs can be filtered out and the risk of virus infections can be reduced. For software vendors, this system is secure because they can choose their own signature schemes and keys without having to trust any third party. For users, this system is secure, since one can be deceived with programs distributed by unlicensed vendors unless all certification centers are compromised. For both software vendors and users, this system is acceptable because it provides a mechanism to resolve disputes, if any, between the two parties.

References

[1] D.E. Bell and L.J. LaPadula, Secure Computer System:Unified Exposition and Multics Interpretation, *MITRE Tech. Report MTR- 2997*, Mitre Corp., Bedford, Mass., March, 1976.

[2] K. J. Biba, Integrity Consideration for Secure Computer System, *MITRE Tech. Report, MTR-3153*, June, 1975.

[3] D. D. Clark and C. T. Wilson, A Comparison of Commercial and Military Computer Security Policies, *Proc. 1987, IEEE Symp. Security and Privacy*, Oakland, CA, April, 1987, pp. 184-194.

[4] Fred Cohen, Computer Viruses: Theory and Experiments, *IFIP-TC11 Computers and Security, Vol 6, No. 1*, 1987, pp. 23-35.

[5] Y.J. Huang and F. Cohen, Some Weak Points on One Fast Cryptographic Checksum Algorithm and its Improvement, *IFIP TC-11 Computers and Security, Vol. 8, No. 1*, 1989.

[6] E. Okamoto and H. Masumoto, ID-based Authentication System for Computer Virus Detection, *Electronics Letters, Vol. 26, No.15*, July, 1990, pp. 1169-1170.

[7] I. Ingemarsson and G. J. Simmons, A protocol to set up shared secret schemes without the assistance of a mutually trusted party, *Proc. of Eurocrypt '90*, May 21-24, 1990.

[8] R.L. Rivest, A. Shamir, and L. Addleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystem, *Communication of ACM, Vol. 21, No. 2*, Feb. 1978, pp. 120-126.