Defense Against Man-in-the-middle Attack in Client-Server Systems*

D. N. Serpanos†
Dept. of Electrical and Computer Eng.
University of Patras
GR-26110 Rion, Patras
Greece

R. J. Lipton‡
Applied Research
Telcordia Technologies
Morristown, NJ 07960
USA

Abstract

The deployment of several client-server applications over the Internet and emerging networks requires the establishment of the client's integrity. This is necessary for the protection of copyright of distributed material and, in general, for protection from loss of "sensitive" (secret) information. Clients are vulnerable to powerful man-in-the-middle attacks through viruses, which are undetectable by conventional anti-virus technology. We describe such powerful viruses and show their ability to lead to compromised clients, that cannot protect copyrighted or "sensitive" information. We introduce a methodology based on simple hardware devices, called "spies", which enables servers to establish client integrity, and leads to a successful defense against viruses that use man-in-the-middle attacks.

1 Introduction

The explosive growth of the Internet and the advances of high-speed networks in the local and the wide area have resulted in the development of a wide range of applications and services in the enterprise and to the home. The increasing throughput to the user, either in the business or at home, have enabled demanding applications and services (e.g. multimedia) with more user-friendly interfaces, leading to increasing acceptance of the technology. Feasibility of application development and service provision to end users (clients) is clearly the first important step in the deployment of services to a wide population. Acceptance and extensive use of the services though will be achieved only when the quality of provided services is satisfactory to the users. One of the main parameters of such quality of service is *service security*.

Security of end systems has been a major concern, since the beginning of computing, especially for systems performing "sensitive" computations. The wide acceptance of personal computers and workstations has triggered a significant problem of security, mainly due to the development of computer viruses of increasing strength and ability. The increased connectivity through the Internet (and other networks) makes the problem even more acute.

One of the main concerns of many service providers is the security of client systems. When a client receives a service, it is common that the user obtains access to proprietary data that the user is supposed to use only once and/or not distribute to other clients. Typical examples include video-on-demand services (where a client should view the video data only once and not transmit them to others), electronic books (where a user should be able to read an electronic book only on the provided device), etc. So, from the point of view of the service provider, the client system should be secure enough to use the "sensitive" data appropriately.

Illegal use of the service data can be achieved only if the client system runs a "stealing program" that misuses the data. Such a program can be run either on purpose by the client himself, or by an illegal intruder (virus) to his/her system without his/her knowledge. The last case is actually feasible and quite dangerous considering that there exist viruses which can remain undetected in a client's system by conventional anti-virus programs; this can be achieved by a virus which becomes the man-in-the-middle between end-users or servers and the anti-virus protection programs, as we describe in this paper (Section 3).

In this paper, we introduce a method that achieves defense from man-in-the-middle attacks in environments where servers are secure. Thus, we address the problem that service providers face when they allow remote clients to obtain data that should be used in very specific ways directed by the service (and data) providers. We describe a method where each client is required to use a secure device, the *spy*, provided by the service provider on the client's system. The server, in co-operation with the spy, executes a

*Work supported by Telcordia Technologies.
†*Contact Author. E-mail:* serpanos@ee.upatras.gr
‡Also, with College of Computing, Georgia Tech.

protocol which establishes the integrity of the client, or, otherwise,it denies service provision to the client. It is important to emphasize that the proposed method is independent of the virus technology that may be used for the attack.

The paper is organized as follows. Section 2 describes the service environment and the assumptions we make as well as an overview of relative work. Section 3 describes the powerful "man-in-the-middle" attack that a virus can successfully launch against systems equipped with conventional anti-virus programs. Section 4 introduces a simple hardware-based configuration for the solution and Section 5 describes the protocols for the successful establishment of client integrity in various environments.

2 The Service Environment

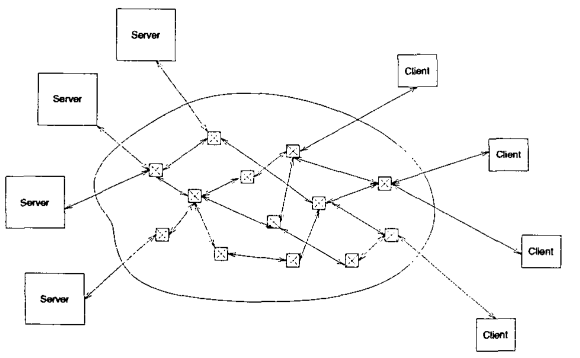

Figure 1. Client-server environment

We consider an environment like the Internet, where services are provided by a set of servers to a set of clients interconnected over a network, as shown in Figure 1. We assume that clients make requests to servers for objects, e.g. video streams, electronic books, programs, data files, etc., which are used for real-time or non-real-time applications. So, the server provides data to clients for one-time (in case of real-time applications) or exclusive (in case of download applications) access. In the following, for simplicity, we consider only one server in the system. The results apply to and are easily extended to the case of multiple servers.

In regard to security, we assume that the servers and the network in the system are secure, but the client is susceptible to software attacks and may be compromised; furthermore, we assume that there is no hardware tampering of the client. So, we consider the cases where the client is himself an enemy, or the client can be attacked by an external enemy with a virus-like attack.

The problem is to avoid violation of the one-time or exclusive-user (single-user) access to the delivered objects. Violation occurs if the objects sent to the client by the server are copied (by the client or compromised client), and thus can be used for unauthorized access. A characteristic example is a video-on-demand service where the server delivers a video stream to a client for real-time display. There are various threats against such a service with conventional technology, even for data that are delivered encrypted. For example, there exist 'screen-scrapper" programs, which copy the information displayed on the screen (after it has been decrypted); data can be continuously copied from the screen and written in a file or transmitted over the network to unauthorized (illegal) users. Many applications and systems, e.g. electronic books for exclusive use, software updating, etc. are vulnerable to the "screen-scrapper" type of attack.

The worst scenario under which this violation occurs is when a "virus" takes over control of the client, i.e. the virus becomes the man-in-the-middle between the server and the client, as descibed in Section 3. The virus can run the original machine as a subroutine, since it controls the client's system, and thus obtain all necessary information for communication with the servers (and the external world in general). This scenario is the worst case one that can appear, because the virus can deceive the server and pretend that it is the authorized client, which it has under its control.

This security problem is significant to a wide range of applications and services ranging from electronic commerce to video distribution and personal services. Due to the deployment of an increasing number of such services, significant effort has been spent and is being spent on defense mechanisms for such attacks.

Man-in-the-middle attacks are well-known in security environments, and have drawn significant attention. Most of the proposed defense methods focus on the development of protocols that enable secure key exchange or generation for secure communication [4]. Typical examples include protocols that use personalized information, certificates, or one-way hash functions. No special attention is paid to establish the integrity of the communicating parties (especially the client in service environments), as our method does. This approach of secure key exchange is reasonable in systems where the clients are considered secure, as for example in the case of smart-cards; it is vulnerable though in environments where the clients may be compromised.

Our methodology ensures integrity of the client through the use of the "squeeze" protocol, and relates to computer virus protection (anti-virus) methodologies, because it ensures integrity of the client, before starting delivery of the service. All conventional anti-virus technologies follow different approaches though. The typical methods use virus detection programs that examine (system) programs stored or in use through fingerprints, signatures, etc., or by observing behavior for "inappropriate" patterns [3]. Various additional methodologies have been suggested that measure system parameters, such as free main memory size, or boot the system from a safe boot sector and establish the integrity of the remaining system through secure programs. All these methodologies and their combinations are vulnerable to man-in-the-middle virus attacks, especially after the system becomes operational. A successful attacker can change parameters, or measurements, and present seemingly correct information to anti-virus programs.

3 Man-in-the-middle Attack with a Virus

Viruses[1] are computer programs with two main goals: to hide (in order to survive and replicate inconspicuously) and to perform some action (undesired, typically, by the legitimate users and operators). Computer viruses are increasingly becoming threatening to computer systems, because they create problems that frequently result to loss of computing power and/or data. Their development and deployment have triggered the development of anti-virus technology, resulting to several efforts and commercial products.

The paradigm used by the computer community, in regard to viruses, is simple: virus programs are *"criminal"* ("undesirable"), following the "hide-and-hit" approach, while anti-virus programs are *policing*, i.e. "chaising" the viruses: they try to identify them, destroy them and repair the damage they have caused (if possible). Based on this paradigm, the authors of viruses focus on developing new "hiding" techniques for new viruses as well as on creating successful mutations of previously detected and recognized viruses. Authors of anti-virus programs collect information on many viruses and improve techniques for identifying (precisely) viruses, either through their behavior, activity or their structure in data, program files or stand-alone.

There are some assumptions in this paradigm, which are driving the success of the antivirus methods up to date:

1. for any virus V(), there can be a program D(), which detects V();
2. anti-virus programs are secure and in power to perform any activity required for the defense of the system.

Athough there exists theoretical work that shows that universal detection of viruses is an undecidable problem [2], it has always been the case that the appearance of a new virus is accompanied by the availability of programs that defend systems from the new intruder. Thus, although the theoretical work indicates that there may exist viruses that are undetectable, reality up to date has shown that the first assumption holds. Furthermore, except the consideration that a copy of an anti-virus program may itself be infected, it is generally considered that a "clean" anti-virus program(s) can always defend the system from hostile intruders.

[1]We use the term *virus* for any program that intrudes a system without the operator's knowledge, and performs some undesirable action (in our example, stealing data); the program may or may not reproduce and infect other systems independently. In this respect, we use the term *virus* to describe a larger class of intruding programs than typically assumed; this class is a superset of the common viruses.

However, the assumptions are far from true, and the paradigm followed up to date renders anti-virus programs easy to work around. As we show, it is possible for an intruder to overcome any defense in the system, if the virus follows the man-in-the-middle attack and infects the anti-virus programs that reside in the system. Analogously to the biological HIV virus, such a computer virus attacks the defense system of the victim. The results of such an attack can be disastrous, since the intruder will control all defense mechanisms and thus can control the infected system completely: the system will become unable to detect and/or react to any undesirable activity.

3.1 Typical Virus and Anti-virus Technology

Viruses are programs that enter a computer system without the users' and/or operator's knowledge; then, in general, they perform some action and replicate and transfer to other systems. They are conceptually composed of 3 main parts:

- **hiding code:** the code responsible for entering a target system inconspicuously, and for hiding so that the virus remains undetected;
- **replicating code:** the code responsible for virus replication and migration to other systems;
- **action code:** responsible to perform the action of the virus on the local system.

Viruses hide in code segments, since they need to execute in order to perform their goal. A typical classification of viruses is based on the placement of their code (i.e., on the function of their hiding code, which places the virus code in difficult to identify areas): *boot sector viruses* hide in system code – specifically, either on the boot sector of hard disks or diskettes –, *file appending viruses* hide in executables, and *macro viruses* hide in macros of data files.

Anti-virus protection is achieved through execution of programs that detect the presence of a virus, identify and destroy the virus and reverse (if possible) the damage it has caused. Typically, detection and identification of a virus is based either on behavior or structure, or both. Behavior-based antivirus programs detect viruses through specific operations or "illegal" activity, while structure-based ones detect them through identification of their code. Both methods require non-trivial functionality in order to avoid false identification of legitimate programs as viruses; furthermore, they need to be able to identify old as well as newly developed viruses. The classification of viruses, presented above, allows anti-virus engineers to focus on specific behavior and/or structural patterns of viruses, but the danger of the approach is that any new virus that diverges significantly in either behavior or structure may prove a great challenge for conventional anti-virus technology. This is the case of the "HIV"-type virus described in this article.

3.2 CIV: Computer Immuno-deficiency Virus

Conventional virus and anti-virus authors follow a simple paradigm: a system has a detection program D() which detects a virus V(), when it enters the system. The goal of V() is to escape detection through use of "hiding" techniques, while D() uses detection techniques that identify the existence of any V(), if possible. We pose the question: *is it possible for a virus V() to fully avoid detection by any D()?*

The answer to the question is *yes*. This can be achieved by having the virus follow an aggressive attack to the system, i.e. by "attacking" the anti-virus program D(). This attack follows the biological paradigm of the HIV virus attack on an organism's defense system. We introduce this paradigm using a simple version of the attack first and later we introduce more sophisticated viruses, which can circumvent all conventional defenses of an anti-virus D(). Due to their analogy to HIV, we name such computer viruses CIV (Computer Immuno-deficiency Virus).

3.2.1 A Simple CIV Computer Virus

Consider a system equipped with the anti-virus detector D(). A virus V() enters the system and performs the following two simple steps:

1. V() creates a new program D'() with two properties:
   (a) D'() provides the same interface to users as D();
   (b) D'() does not really detect.
2. V() replaces D() with D'().

This attack renders V() completely undetectable, because:

a) property (1) makes D'() undetectable from other application programs or the system's human user(s);

b) D'() does not detect (at least not V()).

The attack has an important property: *the simple CIV virus is effective, independently of the anti-virus technology used by program D()*. Despite this effectiveness though, there is functionality that can be added to D() to make this virus attack difficult:

1. D() can "hide";
2. D() can be constructed so that it communicates periodically with a remote server (servers), so that it (they) can verify the integrity of D().

These counter-measures can be circumvented though by a more sophisticated virus V() as we show below.

3.2.2 Sophisticated CIV Computer Viruses

If D() "hides", then it uses some "hiding" technique similar to the one used by typical viruses, and is thus susceptible to detection in the same fashion as viruses. Any detection technique developed for anti-virus technology can be used by V() to detect and identify D().

If D() communicates periodically with a server in order to have the server ensure the integrity of D(), then V() can circumvent this defense with the following attack: V() becomes the man-in-the-middle between D() and its users (e.g., programs running on remote servers). All requests, whether encrypted or not, directed to D() will arrive to D'() –we consider it as part of V()– and then D'() will run D() as a subroutine and return the results exactly as D() returns them. The only way for this attack scenario to fail is if D() realizes that it is being run as a subroutine, i.e. that some V() simulates D(); this would lead D() to the conclusion that it is compromised, and then to report it in its results. However, this problem is undecidable, making it infeasible for D() to identify, if indeed it is used for simulation.

Although D() cannot identify functionally if V() is simulating its behavior, it seems possible for one to use performance measures for such identification: simulation results in longer delays. The remote server can measure the execution delay of its request on a system in question and conclude that the system is compromised (and take appropriate measures), if the delay is higher than expected. Again, one can consider defeating scenarios though, where D'() can execute D() remotely on faster systems, etc. Thus, it becomes clear that, if an aggressive virus V() attacks D() and becomes man-in-the-middle in an infected system, then it can become undetectable and control the infected system fully.

There are many practical issues that need to be addressed in order to create such a virus V() –or rather CIV()– which will defeat all possible antivirus solutions. However, the development of "stealth" viral technology [5] demontrates that it is feasible to produce viruses that behave as man-in-the-middle; in stealth viruses up to date, the goal of the introduced man-in-the-middle is to stand between the hard disk and the processor and present an image of an un-infected hard disk to any program reading the state of the disk.

CIV viruses are feasible and can be implemented relatively easily. Actually, a primitive CIV virus (which is effectively the first CIV-type virus) has already appeared, although its goal has not been any attack as described above. The virus, named *rescue* or *Pro-aLife*, is a memory resident virus that targets a wide range of antivirus products [1].

Despite the practical issues to develop effective CIV viruses, it is clear that the new attack paradigm needs to be addressed in conventional antivirus technology, which seems fully unprepared for such a paradigm shift in viral activity (conventional antivirus programs provide quite an "easy" target to any program interested in attacking them).

4 A Solution based on Hardware

Defense against CIV-type viruses is feasible. The proposed solution is composed of two parts:

1. we attach trusted devices, called **spies**, to the clients and establish secure communication between the server and the spies;

2. we ensure that the application is executed in a "safe" environment at the client, where either no offending program (i.e. a virus) resides in the client's memory, or it is detected as soon as it attempts to steal data.

The spy is a device with 3 main characteristics:

1. it is a passive I/O device that is not placed in a critical path of the client system;

2. it can detect I/O activity, such as disk accesses and network transmissions;

3. it has some computational power and memory, so that it can perform cryptographic computations (e.g., public-key cryptography, etc.). Importantly, computation speed is not critical.

The spy can be a simple PCMCIA card, attached to the client I/O bus, with an embedded processor and memory to perform the necessary cryptographic computations. Since the spy needs to identify data loss due to disk file accesses or network transmissions, it needs to be attached to the client system so that all disk and network device transactions are visible to it[2]. It is important to emphasize that, *the spy is a simple, passive, low-cost device*, in contrast to alternative methods to secure hardware which require much more complex and intrusive changes, with a higher cost.

5 Use of the "Spy"

We implement spy-server secure communication through authentication of spies and through exchange of encrypted messages for data exchange; so, the virus (man-in-the-middle) mentioned above can have no influence on the communication between server and spy.

5.1 A Simple "Spy" Protocol

We present a simple protocol using the spy, for a video-on-demand service in the environment of Figure 1. The client is equipped with a spy, which "observes" the client system, while it executes the application, and reports to the server the "symptoms" that indicate that a virus has entered the client and is "stealing" data; these symptoms include disk accesses and network packet transmissions.

As the video stream arrives at the client, data are stored in main memory and displayed on screen. If a virus exists in the system and copies the data of the video stream in any fashion, then, eventually, it needs to either copy them to a file on disk or transmit them over the network to another system. When the spy detects such a disk or network transaction at the client, it informs the server, which reacts appropriately (e.g., stopping the service)[3].

This approach clearly detects a virus, but it allows loss of some data, i.e. it has a *data leak*. This occurs, because the spy will identify the existence of a virus after an "inappropriate" (illegal) transaction occurs, i.e. a file write or network transmission; so, the data that participate in this transaction may be stolen. There is a bound on this data leak though, since the maximum size of lost data is equal to the client's main memory size. In certain applications however, this may be acceptable, as for example in the case of video-on-demand (movie) applications, where loss of such an amount of data accounts for a very small fraction of the whole movie and is useless to the thief.

It is possible to avoid data leaks with advanced, sophisticated protocols, as the one we introduce below.

5.2 Advanced Spy Protocol

The concept of the approach is that the spy can always bring the client system to a "clean" state, i.e. a secure state where the non-existence of a virus in main memory is guaranteed, in order to start an application. Subsequently, the spy can "observe" the client system, while it executes the application, and report to the server the "symptoms" that indicate that a virus may have entered the system and is "stealing" data (e.g., disk accesses, network packet transmissions, etc., as mentioned above). The spy reports these symptoms to the server, which in turn re-acts according to the policy adopted for such situations.

The client system can also be checked periodically, in order to identify if there is a virus in the system, which has not been detected yet, due to absence of the observed symptoms. In such a protocol, the spy or the server can occassionally initiate a "challenge" to the client system which will force the client to present a "symptom", if a virus is present. This is done with the method described below, which is based on the property that when a virus exists and is active in the system, its code is residing in main memory. Such challenges are needed only in case there is activity in the client that may allow a virus to enter the system or become main memory resident for execution (e.g., initiation

[2]Thus, if a technology is used where a disk or a network device is attached to the client memory bus and not the I/O bus, then the spy needs to be implemented in such a way so that the necessary transactions are visible

[3]We assume that no other application is running on the system; this can be enforced, if necessary, as shown below with the advanced spy protocol.

of execution of a new program, etc.).

5.2.1 Identification of Virus Existence

The most basic operation needed in our environment is a *Safe_Exec(P)* routine, where P is a program. *Safe_Exec(P)* executes 5 operations to assure that P is executed safely:

1. assures that P is signed, i.e. that it is safe to execute;
2. assures that no other program runs;
3. delivers secrets (e.g., keys) to P from the spy, if necessary (if assurances 1 and 2 -above- are given);
4. executes P;
5. cleans up, when P finishes execution[4].

All the above operations are clear to understand, except the operation where *Safe_Exec()* assures that no other program runs (operation 2). We achieve this with the following program, which is based on the property that the virus program needs to reside in memory in order to execute:

```
Id_Virus()
{
  for (i= 1; i< N; i++)
  {
    block_write(from_spy, to_mm, all_mm_but_this_code);
    block_write(from_mm, to_spy, all_mm_but_this_code);
  }
  jump addr_0;
}
```

The concept behind the method to detect a virus in the system is simple: the virus needs to reside in some space in main memory, and our protocol "squeezes" the virus out of its storage area. This "squeeze" protocol works as follows.

Assume that the main memory where programs reside has a size M and that a resident virus has a size L (both M and L are measured in bits). Furthermore, let us assume that the running program ($Id\_Virus()$),which ascertains that no other program is running in the system, i.e. checks the existence of a virus, has a size L_{Id}. The program identifies the existence of resident code (virus) by writing and reading random bits in the memory that should not have any other code, i.e. in the $M - L_{Id}$ remaining bits. In this fashion, if a virus exists in memory, it needs to "guess" the random bits that would be written in its place; this is necessary because the spy will read the random bits after writing all memory, and the virus will have to return a response. If the virus attempts to store these random bits on disk, it will be identified through the disk access. So, the only solution for the virus is to guess the answer, i.e. the random bits. The probability of making a correct guess is: 2^{-L}, where L is the size of the virus in bits. To avoid the probability of a small-size virus which can make a correct guess, we propose that this process of reading and writing the memory is performed many times (N in the program), so that the probability that a virus makes a correct guess is brought to any desired level.

Although this approach seems non-problematic, it has a pitfall because there is still an area where the virus can hide: the virus could replace the program that performs this test with another one which, when optimized for space, is smaller and thus allows the virus to fill the available space. So, if the original program has a size L_p and the virus can replace it with a smaller one with size $L_{p'}$, then the available space for the virus is: $L_v = L_p - L_{p'}$; considering the small size of this program as well as the availability of optimizing compilers, we believe that L_v cannot be practically large enough to accomodate a complete virus program; furthermore, the case $L_v = 0$ seems possible.

It should be noted here that, *limited memory of a known size is key to the cryptographic protocol presented.* Furthermore, *the presented approach is effective independently of the virus technology.*

The advanced spy protocol leads to a method that can achieve *no data leaks.* Importantly it can be used, not only for on-line applications, such as the video-on-demand service mentioned previously, but for off-line applications as well. In such off-line applications, the server can provide the client with encrypted data (off-line) and then the server can provide the spy with the decryption key, which can be delivered to the application while the *Safe_Exec()* is executed. This ability increases the spectrum of possible applications significantly.

6 Conclusions

We have described a powerful class of computer viruses which implement the man-in-the-middle attack. Conventional anti-virus defenses are not capable to protect systems against this attack. We have introduced an innovative defense with the use of a device that implements a special protocol, the "squeeze" protocol, to detect the existence of a virus.

References

[1] D. F. (analyzed by Mikko Hypponen). F-secure virus information pages. http://www.data-fellows.com/v-descs/proalife.htm.

[2] F. Cohen. *A Short Course on Computer Viruses.* Wiley and Sons, 1994.

[3] J. Kephart, G. Sorkin, D. Chess, and S. White. Fighting Computer Viruses. *Scientific American,* November 1997.

[4] B. Shneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Wiley and Sons, 1995.

[5] R. Slade. *Guide to Computer Viruses.* Springer-Verlag, 1996.

[4]This is imperative and must be complete, since there may be secrets of the spy stored in memory.