Computer Virus Infections: Is NSA Vulnerable?

[redacted]

(b)(3)-P.L. 86-36

~~This article is classified TOP SECRET in its entirety.~~

This paper is concerned with computer viruses – a potentially dangerous attack on computer systems. The virus is a special case of the trojan horse problem, distinguished by its ability to reproduce itself and infect previously healthy programs. [redacted]

INTRODUCTION

What is a computer virus? A computer virus is a self-propagating trojan horse.[1] A computer virus has three main parts: a mission component, a trigger mechanism, and a self-propagation component. The mission component is the executor of the deed the virus is designed to accomplish, e.g., erasure of all data on a computer system. The mission component lies dormant until activated by the trigger mechanism. The trigger mechanism tests one or more aspects of the system state such as the current date to determine whether to activate the mission component. For example, a possible virus may be of the form: if today's date is 10/01/85 then erase all accessible data on the computer system, otherwise propagate self. Indeed an actual simple virus is not much longer or complicated than this. The third part of the virus, the self-propagation component, allows the virus to quickly spread to other programs to which the virus is not already attached. I call this the process of "viral infection."

[redacted]

[1] A trojan horse, in the most general sense, is a computer program which, in addition to performing a desired function, causes a malicious side effect when run by an unsuspecting user. Even though the trojan horse problem is widely recognized, trojan horse identification is difficult.

(b)(1)
(b)(3)-P.L. 86-36

DISCUSSION

[redacted]

The question of whether or not an algorithm exists to decide whether a program is infected with a virus appears to be unresolved. Consultation with Dr. [redacted] and a number of colleagues within [redacted] has indicated that a formalization of the meaning of "infected" is required in order to make any rigorous statements about viruses. A theory of viral infection is required to characterize properties associated with viruses and ultimately to prove whether or not a decision algorithm exists.

Based on Rice's Theorem,[2] it is the author's intuition that a decision algorithm to determine whether or not a program is infected does not exist. In fact, even if it is proved that such an algorithm exists, there is no guarantee that the actual algorithm can be found. If the algorithm does not exist or cannot be found, it would not mean that the problem is hopeless. It would mean only that its general solution is not open to mathematically rigorous proof. This result would leave two approaches: (1) restrict the computer system specification so that a general solution is not required or (2) solve the problem heuristically, acknowledging that the solution is not rigorously complete. The second method seems to provide the cheapest and easiest approach without drastically changing the operational environment.

The thrust of the recommended actions proposed in this paper is to provide mechanisms to make the virus attack more difficult and expensive to a penetrator. This method is known as increasing the work factor – the amount of resources the attacker must expend to accomplish a successful penetration. The cost is measured in terms of both time and money. If the time required to mount a virus attack against a given system exceeds the life of the system, then the system is effectively secure. Similarly, if the cost is made high enough, the attacker will divert his resources to a more fruitful target. In either case, an effective solution is reached.

[2] Rice's Theorem states that any nontrivial property of the recursively enumerable sets is undecidable. A property is "trivial" if it is either true of all members in the set or of no members in the set. Since the set of all possible algorithms is a recursively enumerable set, it would seem to follow that the nontrivial property of being infected would be undecidable. For further reading on Rice's Theorem, see Hopcroft (cited in the bibliography).

(b)(1)
(b)(3)-P.L. 86-36

(b)(3)-P.L. 86-36

Attack Classes

The three major types of computer attacks are compromise, spoofing, and denial of services. They are discussed in detail in the following paragraphs.

a. Compromise – the unauthorized disclosure of classified data.

[redacted]

b. Spoofing – the unauthorized alteration of classified data.

[redacted]

Paradoxically, the type of program in which the virus lies can tell much about the system. Using a biological analogy, a human who finds himself in an alien environment knows a great deal about that environment by virtue of the fact that he is alive, e.g., there is enough oxygen to breath, the ambient temperature is within the human-tolerable range, etc. By the same token, a C language program, for example, "knows" with a high degree of certainty that it will be running in a UNIX-type environment. If the host program in which a virus hides will not run in a given computer system, there is no reason to ever import that program. If it is imported, it will not execute and presents no direct threat to the computer system. The following two scenarios exemplify the spoofing attack. The scenarios are not intended to be of sufficient detail to be beyond criticism, but to give a flavor for attacks that might be possible.

[redacted]

(b)(1)
(b)(3)-P.L. 86-36

[redacted]

c. Denial of Service – the unauthorized use of system resources to the exclusion of authorized users. Examples of denial-of-service attacks include "unfair" CPU utilization or "excessive" disk storage space usage by a user or process to a degree that negatively impacts the other users on the system. More concretely, if a user gets control of the CPU scheduling process, the computer can be directed to execute only his process to the exclusion of all others.

At first glance, the infection process itself may seem to represent a denial-of-service attack. To a small extent this is true; however, a viable infection must conceal itself by minimizing the time required to accomplish the infection process before executing the legitimate program. Specifically, the infection time required must be small compared to the time required to execute the legitimate program so that the user does not notice the delay. Indeed, once all of the programs have been infected, a process which can occur exponentially fast, the infection process consumes no more system resources until its mission component is activated.

The denial-of-service attack is similar to the spoofing attack but uses more brute force. Instead of providing false information during times of crises, programs are instructed to bring the system to a halt. [redacted]

(b)(1)
(b)(3)-P.L. 86-36

[redacted]

The threat of computer virus attack is very real. Fred Cohen's preliminary investigations reported in the paper cited in the bibliography, involving the actual production of working viruses on systems which included the Univac 110B, TOPS-20, VM/370, and VMS, demonstrates viral production times ranging from 6 to 30 hours. The average time to acquisition of full system privileges giving the virus unchallenged access to any data on the computer system was 30 minutes after virus introduction to the targeted computer.

Virus Uniqueness

What makes the computer virus problem different from the more general trojan horse problem? The difference is analogous to the difference between having one traitorous soldier in your ranks versus an infectious disease which converts your soldiers to enemy soldiers. The effect of one bad soldier is usually limited to his own group. The effect of the infectious disease is likely to be the loss of the entire war.

Current computer security research suggests that good security is accomplished by the separation of the computer system into small, isolated groups of related programs. Should a problem occur, this limits the damage to within that group. This is analogous to the bulkhead separation of compartments in ships and submarines to prevent uncontrolled flooding from a single leak.

The virus and the trojan horse, in any given partition, are indistinguishable in terms of the amount of damage they can cause. The difference is in the ability of the infections to escape the partition. The trojan horse is active only within the partition. The virus, on the other hand, has the potential to spread itself to other partitions as well. The virus quickly infects virtually all programs in the partition. The process is very simple and very fast. When the original infected program is run, it first finds an executable file, appends a copy of itself to the file, executes its mission component if the triggering event has occurred, and then executes the program body of the host program.

When a program runs in the user's space, it runs with the same access as the user himself. The algorithm for infection requires only reads, writes, and file renaming. For example, the algorithm could be to copy the virus part to a temporary file, append the reloaded executable program code to the virus code, delete the old program version, and then rename the temporary file to the name of

(b)(1)
(b)(3)-P.L. 86-36

the old program. At this point, there would be two infected programs, the original and the program the virus infected. The accesses required for these operations are almost universally allowed to the owner of the files and, hence, are available to the virus when run in the user's space. The collection of programs to which a virus has the required access to propagate when executed by a given user will be called a "partition."

Execution of either of the two infected programs can infect other programs in the partition. Given that programs in the partition are run with some regularity, the number of infected programs increases geometrically until all programs are infected. Furthermore, information flows must also occasionally take place across partitions by operational necessity. When upgrading system software facilities, software systems such as data base managers or editors developed on other computers must be loaded on the computer system. Programs often need to be copied from one partition to another in order to share the benefits of a program developed by users on the system. Since all programs within the virus-infected partition are potentially infected, the probability of transmission of the virus is greatly increased.

With the infectiousness of viruses established, I turn to the question of virulence. Even though the potential damage within a partition is equivalent between the virus and the trojan horse, the reliability and ease with which the damage can be done is greatly increased in the case of a virus. Given a fairly large number of programs within a partition, a virus infection obviously has many more traitorous agents doing its bidding. This could mean either a large number of agents (programs) attempting the exact same subversive task or possibly cooperating in subtle ways to accomplish a larger integrated task. The first case yields a high reliability of task success by simple redundancy. The second case is much more theoretical and sophisticated but provides the potential for more subtle tasks to be achieved.

The infectiousness and virulence unique to a virus arises from its ability to propagate itself. Solutions should address this particular feature in order to demote the virus to a trojan horse subject to the corresponding protection mechanisms, inadequate as they may be. Specific solutions are offered later in the paper.

Specific Vulnerabilities

[redacted]

(b)(1)
(b)(3)-P.L. 86-36

[redacted]

SOLUTIONS

The nature of the virus problem requires the simultaneous pursuit of several different solutions. First, both long- and short-term solutions should be sought. Immediate stopgap countermeasures should be taken to minimize the risk from this threat. Furthermore, some long-term, fundamental research is required to

(b)(1)
(b)(3)-P.L. 86-36

investigate the offensive potential of and defensive mechanisms for sophisticated viral attacks.

[redacted]

Before I recommend specific solutions, I must preface my remarks with some cautions. Persons using the computers should carefully evaluate these suggestions, along with any others made as a result of the virus problem, in terms of operational impact. Knee-jerk reactions can cause more problems than they solve. Perfect computer security can be achieved by hermetically sealing all computers, but they could then do no useful work. Clumsy, complicated procedures and policies are more likely to be ignored than followed.

The cost and benefit of each suggestion should be compared and properly weighed and, in turn, compared to the risk. I suggest that formal techniques of risk analysis be applied to the problem to establish a procedure of measuring this trade-off.

Considering the above mentioned specific vulnerabilities, the steps towards preventing trojan horse importation are as follows:

[redacted]

(b) (1)
(b) (3)-P.L. 86-36

[redacted]

Virus-specific Countermeasures

[redacted]

(b)(1)
(b)(3)-P.L. 86-36

[redacted]

Operational Ramifications

This section may be more appropriately labeled, "What does this all mean to me?" This paper should have an immediate effect on operation as well as research areas. The suggested solutions which can be implemented immediately [redacted] are strongly urged.

This paper is essentially a call to arms for all computer systems research and support groups to focus attention on this very real problem. The solutions proposed are in various stages of development. Each should be analyzed, implemented, and tested. New ideas should be generated. More resources should be dedicated to the problem to find viable solutions for both the long and short term.

(b)(1)
(b)(3)-P.L. 86-36

[redacted]

Implications to Computer Security Criteria

[redacted]

All right then, how about adding Biba's[3] integrity extensions to the mandatory model requirements in the Criteria. The addition of integrity levels to the mandatory access control mechanisms is certainly a step in the right direction. This additional control, however, is not a panacea; in fact, it is only another measure to increase the work factor of viral penetrations.

The integrity dual model suggests the segregation of all of the programs on a computer system based on the degree of trust that the program does exactly what it is designed to accomplish and nothing more. For example, if the designing software engineers were all Top Secret cleared, the software was formally specified and verified correct, and a large panel of experts reviewed the final code,

[3] Biba suggested the addition of the integrity dual of simple security and the *-property proposed by Bell and LaPadula. In sum the model requires (1) no writing "up" in integrity (simple integrity) and (2) no reading "down" in integrity (integrity *-property). Note that here, read and execute may be considered equivalent accesses.

(b)(1)
(b)(3)-P.L. 86-36

such a program might be placed in the class of "high integrity" programs. Conversely, if a program's origin is no longer known and the source code is not available for inspection, then such a program might be placed in the "low integrity" class of programs. All programs would be labeled as to which class they belong. Now, if the system prevents all "low integrity" programs from accessing any "high integrity" programs, then there is some measure of protection against the spread of viral infection from lower integrity levels to higher integrity levels.

The establishment of a hierarchy of integrity levels requires some way of determining the relative degree of reliability. With respect to the virus problem, this corresponds to determining the probability of an algorithm being infected or its susceptibility of infection. The method of such a determination is unclear and may itself be unreliable. If the method were implemented as an algorithm on the computer system, it too would be susceptible to the very same viral attacks as the other programs.

There is no way of guaranteeing that the routines labeled as "highest integrity" are not infected if a decision algorithm to detect viruses does not exist or cannot be found. Infection of the highest integrity routines could then eventually lead to a system-wide infection. This would make the whole integrity structure useless and could give a false sense of assurance. Therefore, the addition of integrity levels into mandatory access can only be a part of an integrated strategy to combat the virus attack.

CONCLUSION

[redacted]

It appears that a large variety of inexpensive measures can be taken to counteract a large percentage of the potential viral attacks. Furthermore, other countermeasures can be adopted to increase the work factor of any virus attempting system penetration.

[redacted]

How to increase work factors to the extent of making this attack infeasible is a matter for more research. I suspect the solution will be heuristic in nature, and the final protection system will probably come to resemble the human immunological system in approach. In general, I believe pattern recognition and artificial intelligence will play a key role in long-term research into this problem.

(b)(1)
(b)(3)-P.L. 86-36

~~(FOUO)~~

[redacted]

BIBLIOGRAPHY

Bell, D. E. and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997 Rev. 1, MITRE Corporation, Bedford, Massachusetts, March 1976.

Biba, K. J. "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, Bedford, Massachusetts, April 1977.

Cohen, Fred. "Computer Viruses: Theory and Experiments," *7th DOD/NBS Computer Security Conference Proceedings*, 1984.

Department of Defense Trusted Computer System Evaluation Criteria, DOD Computer Security Center, Fort Meade, Maryland, 15 August 1983.

Hopcroft, John E. and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979, pp. 177-213.

(b)(3)-P.L. 86-36
(b)(6)