Fault Tolerant Software for Computer Virus Defense

by Dr. Frederick B. Cohen ‡

Abstract

In this paper, we discuss the use of fault tolerant software techniques used in defense against computer viruses and other integrity corruptions in modern computer systems. We begin with a summary of research summaries on computer viruses, their potential for harm, and the extent to which this potential has been realized to date. We then examine major results on the application of fault tolerant software techniques for virus defense, including; the problems with conventional coding schemes in detecting intentional corruptions and the use of high performance cryptographic checksums for reliable detection; an optimal method for detecting viruses and preventing their further spread in untrusted computing environments; the use of redundancy and automated decision making for automatic and transparent repair of corruption and continuity of operation; and the use of fault avoidance techniques for limiting viral spread. Next we summarize several years of real-world results on the application of these techniques in DOS and UNIX computing environments, the implications of these results to other computing environments, and architectural issues in implementing hardware assisted virus defense based on the software fault tolerance techqniques alreading in widespread use.

‡ This research was funded by ASP, PO Box 81270, Pittsburgh, PA 15217, USA

1 Background

The "Computer Virus" problem was first introduced in 1983 [1], when the results of several experiments and substantial theoretical work showed that viruses could spread, essentially unhindered, even in the most 'secure' computer systems; that they could cause widespread and essentially unlimited damage with little effort on the part of the virus writer; that detection of viruses was undecidable; that many of the defenses that could be devised in relatively short order were ineffective; and that the only effective defenses were limited transitivity of information flow, limited function so that Turing capability was unavailable [2], and limited sharing.

In subsequent papers; it was shown that limited sharing, in the most general case, would cause the information flow in a system to form a partially ordered set of information domains [3]; it was proven that limiting of transitivity, functionality, and sharing were the only 'perfect' defenses [4]; and it was suggested that a complexity based defense against viruses might result in a practical defense [5]. It was also shown that viruses could 'evolve' into any result that a Turing machine could compute, thus introducing a severe problem in detection and correction, the connection between computer viruses and artificial life, and the possibillity that viruses could be a very powerful tool in parallel computing.

While initial laboratory results showed that viruses could attain all access to all information in a typical timesharing computer system with properly operating access controls in place in an average of only 30 minutes and that network spread would be very rapid and successful, experiments and anlytical results were severely limited by the unwillingness of the research community to even allow statistical data to be used in assessing the potential risks. Although substantial theoretical results indicated how quickly viruses might be expected to spread [6] given an accurate characterization of an environment, and an experiment at the University of Texas at El Passo showed that in a standard IBM PC network, a virus could spread to 60 computers in 30 seconds [7], thus the actual spread rate could not be determined accurately until real-world attacks took place.

Real-world viruses started to appear in large numbers in 1987, when viruses apparently created in Pakistan, Isreal, and Germany all independently spread throughout the world, causing thousands of computer systems to become unuseable for short periods of time, hundreds of thousands of computers to display spurrious messages, tens of thousands of users to experience denial of services, and several international networks to experience denial of services for short periods [8]. By 1988, there were about 20 well known and widely spread computer viruses, in early 1990, the IBM high integrity research laboratory reported over 125 unique viruses detected in the environment [9], and by March of 1991, over 200 unique

real-world viruses were known in the research community [10] [1], with a rate of more than one new virus introduced into the global computing environment per day.

In the period before viral attacks became widespread, there was little interest from the broader research community, and research results were considered of relatively little interest to funding agencies, and even though early results predicted many of the widespread implications we now see, very few organizations took any measures to defend themselves [11]. In the following years however, interest has sprung up throughout the world research community, and there are now international computer virus conferences more than once a month, hundreds of university researchers, and tens of books on the subject. For more complete summaries of the field, the reader is referred to summary reports [19,20]

Dispite the dramatically increased research effort and funding levels in recent years, relatively few researchers have developed basic research results or useful defensive techniques, and most of the useful techniques are based on basic results from fault-tolerant computing, with special consideration required to deal with defense against intentional attackers rather than random noise.

In the remainder of this paper we will look at how software based fault-tolerant computing techniques have been used to deal with the computer virus problem.

2 A Multitude of Broken Defenses

Many defensive ideas have been examined for their viability in virus defense. The vast majority of them have failed to pan out because there are generic attacks against them, they produce infinite numbers of false positives and false negatives, or they are too costly to be effectivly applied [7]. We now examine several of the well known ideas that are in widespread use even though we have known for some time about their vulnerabilities.

The most common virus defense is the so-called 'scanner', which examines computer files to detect known viruses. Scanners have several important problems that have a serious impact on their current and future viability as a defense, most notably; they only detect viruses known to the author; general purpose virus detection is undecidable, so looking for known viruses produces infinite numbers of false negatives; scanners tend to produce false positives as new programs enter the environment; they are not cost effective relative to other available techniques; scanners are ineffective against many types of evolutionary viruses; and they become less cost effective as time passes [7]. There are a number of variations

[1] some virus defense products claim to operate on over 600 known PC viruses

on scanners, most notably the so-called 'monitor', which is a variation on the 'integrity shell' technique described later in this paper [12,13], and which dramatically reduces the costs associated with detecting known viruses.

Another interesting idea was the use of built-in self-test for detecting and possibly correcting viruses in interpreted information [5]. It turns out that all such mechanisms are vulnerable to a generic attack which is independent of the particular mechanism [7], but the concept of using complexity to make attack very difficult remains one of the most practical techniques.

A third common technique is to 'vaccinate' a program against viruses by modifying the program so that the virus is fooled into 'thinking' that the program is already infected. This has several very important drawbacks, primarily that not all viruses have to check for previous infection, vaccinating against large numbers of viruses may require so many changes that the resulting program will not operate, and n-tuples of competing viruses may make vaccination impossible [7].

Multiversion programming has also been suggested as a solution to the virus problem [1], and recent improvements in this technique have made this more and more feasible from an operational standpoint, [26] but the costs associated with these techniques make them tollerable only in a very limited number of environments [7], and it is unclear whether they will be useful in avoiding the effects of computer viruses because they don't address the ability to discern between legitimate and illegitimate changes. An n-version virus could presumaby infect $(n+1)/2$ copies of the legitimate program, thus causing the voting technique to kick out the legitimate program in favor of the virus [1].

3 Coding Techniques

Although precise virus detection is undecidable [4], we may be willing to suffer an infinite number of false positives and a very low probability of false negatives in order to have an effective defense. This can be achieved through the use of coding techniques to reliably detect changes. For example, a simple checksum or CRC code could be used to detect changes in files. The problem with these methods is that they are easily forged, so that an attacker could easily make a modification while leaving the checksum or CRC information unchanged [1]. The reason for this vulnerability is that these coding techniques are designed to detect corruptions due to random noise with particular characteristics, but they are not designed to detect intentional changes designed to bypass them.

In the case of the simple checksum, forgery is trivial. You simply calculate a checksum,

modify the information, 'XOR' the current checksum with the original checksum, and append the result to the end of the information. Even with a checksum which incorporates the size of the information, forgery is straight forward for the vast majority of computerized information because data can be compressed before attack, leaving additional space that can be filled with null information. A compression virus if this sort has been demonstrated [1].

In the case of a CRC coded checksum, attack is quite similar. First you determine the coefficients of the CRC expression, and then you determine the series of bytes necessary to generate the difference between the original checksum and the altered checksum by solving the simultaneous equations. Even if you don't know the specific equation ahead of time, if you can access the CRC codes associated with a number of checksummed data sets sufficient to solve the simultaneous equations, you can determine the coefficients and forge CRC codes and/or make modifications such that the resulting CRC codes are identical to the originals. Several techniques have been devised to use multiple CRC codes with pseudo-randomly generated or user provided coefficients, but these appear to be simple to break as well.

An alternative method designed to withstand substantial attack by knowledgeable attackers is the cryptographic checksum. The basic principle is to use a secret key and a good but fast cryptosystem to encrypt a file and then perform a checksum on the encrypted contents. If the cryptosystem is good enough, the key is kept secret, and the process meets performance requirements, the result is a usable hard-to-forge cryptographic checksum [5,14,15]. In this case, we can store the checksums on-line and unprotected, and still have a high degree of assurance that an attacker will be unable to make a change to the stored information and/or the associated cryptographic checksum such that they match under the unknown key when transformed under the hard-to-forge cryptographic checksum.

Thus we can use this variation on standard coding techniques to reliably detect the changes associated with a computer virus even in the presence of a knowledgeable attacker, but we still have the problem of finding a way to efficiently apply the cryptographic checksum for virus detection.

4 Optimal Detection and Infection Limitation

We mentioned earlier that all built-in self-test techniques are vulnerable to a generic attack. The basis of this attack is that the virus could activate before the program being attacked, and forge an operating environment that, to the self defense technique, shows the altered information to be unaltered [7]. Since the introduction of this concept in short courses in 1988, several so-called 'stealth' viruses have appeared in the environment with the ability

to forge unmodified files when the DOS operating system is used to read files, thus making detection by self-examination untennable.

An alternative to built-in self-test is the use of a system-wide test capability that uses cryptographic checksums to detect changes in information. The question remains of how to apply this technique in an efficient and reliable manner. It turns out that an optimal technique for performing cryptogrpahic checksums called an 'integrity shell' has been found [12,13].

An integrity shell is a testing method that performs tests of information that is about to be interpreted just before the information is interpreted. This technique is optimal in that it detects all primary infection, [2] prevents all secondary infection, [3] and performs only necessary checks when they must be performed in order to fulfill the previous requirements. Cost analysis has also shown that this technique is more cost effective in actual use than any of the less optimal techniques explored to date, including less reliable methods such as scanners [16]. There are even viruses designed to bypass less optimal cryptographic checksum based defenses by only infecting information that has been recently changed. A similar cryptographic checksum method has been proposed for multi-level trusted systems [17], and finer grained hardware based detection at load time has been subsequently proposed [18].

5 Automated Repair

Automated repair has been implemented with two techniques; for known viruses, it is sometimes feasible to remove the virus and repair the original data set with a custom repair routine; while general purpose repair is accomplished through on-line backups.

Although custom repair has some appeal, it is possible to write viruses that make this an NP-complete problem or worse through the use of evolution [4,7]. In several cases, customized repair has also produced undesired side effects, but this is primarily because of errors in identification of viruses or because certain side effects cause by viruses are not reversible from the information remaining in the data set.

On-line backups are used in an integrity shell to replace a data set with an image of the data stored when it was last trusted. This succeeds in all but the rarest cases, but has the undesirable side effect of doubling the space requirements for each covered data set. This problem can be reduced by 50% or more in cases where original data sets have sufficient

[2] infection by information that contains a virus but has been trusted nonetheless

[3] infection by information infected through primary infection

redundancy for compression to be effective, but the overhead can still be unacceptable in many cases. On-line backups are also vulnerable to arbitrary modification unless they are protected by some other mechanism. Two such mechanisms have been devised; one cryptogrphically transforms the name and/or contents of the redundant data set so as to make systematic corruption difficult, and thus force an attacker to destroy a multitude of information leaving strong indicators of the presence of the attack; and the other is to protect the on-line backups with a speciual protection mechanisms so that they can only be modified and/or read when the integrity shell is active in performing updates and/or repairs. Both of these mechanisms have been quite effective.

6 Fault Avoidance Techniques

In almost all cases where viruses modify files, they exploit the operating system calls for file access rather than attempting to perform direct disk access. In systems with operating system protection, this is necessary in order to make viruses operate, while in unprotected systems, it is too complex to implement the necessary portions of all versions of the operating system inside the virus, and it makes the virus less portable to hinge its operation on non-standard interface details that may not apply to all device types or configurations. An effective fault avoidance technique is to use enhanced operating system protection to prevent viruses from modifying some portion of the system's data sets. This may sound simple, but it turns out it is non-trivial.

It turns out that because viruses spread transitively, you have to limit the transitive closure of information flow in order to have an effective defense [4]. In the vast majority of existing computer systems, the access control scheme is based on the subject/object model of protection [21], in which it has been shown undecidable to determine whether or not a given access will be granted over time. As we pointed out earlier, this problem can only be resolved in an information system with transitive information flow, sharing, and Turing capability through the implementation of a partially ordered set [3].

To date, only one such system has been implemented [22], and preliminary operating experience shows that it is operationally more efficient and easier to manage than previous protection systems, primarily because it uses coarse grained controls which require less time and space than the fine grained controls of previous systems, and because it has automated management tools to facilitate protection management. It has also proven effective against the transitive spread of viruses.

Although covert channels [23] still provide a method for exploitation by domains near

the INF of the partially ordered set just as Bell-LaPadula based systems [24] are vulnerable to the same sort of attack by the least trusted user [1], with partially ordered sets, there needn't be a single INF, and thus even the impact of attacks exploiting covert channels can be effectively limited by this technique.

A wide variety of other fault avoidance techniques have been implemented, including the automated rollback of system state at bootup to avoid the effects of corruptions in the bootstrapping process [25], testing of all disks entering an area for known viruses using a scanner [7], physical isolation from external systems [1], and change control [7]. Although all of these techniques provide limited coverage against many current attacks, they have serious and fundamental cost and effectiveness problems that make them less desirable than more sound and cost effective techniques [7].

7 Experimental and Real-World Experience

In the laboratory, numerous viruses have been tested against a wide variety of software-based defensive techniques. Although most experiments indicate very little because their results are easily predicted, occasionally we find a surprising result and have to improve our models of what has to be covered and how to effectively cover it.

8 Architectural Implications

9 References

1. 1 - F. Cohen, "Computer Viruses - Theory and Experiments", originally appearing in IFIP-sec 84, also appearing in DOD/NBS 7th Conference on Computer Security, and IFIP-TC11 "Computers and Security", V6(1987), pp22-35 and other publications in several languages.
2. 2 - A. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", London Math Soc Ser 2, 1936.

3. 3 - F. Cohen, “Protection and Administration of Information Networks with Partial Orderings”, IFIP-TC11, “Computers and Security”, V6#2 (April 1987) pp 118-128.
4. 4 - F. Cohen, “Computer Viruses”, Dissertation at the University of Southern California, 1986.
5. 5 - F. Cohen, “A Complexity Based Integrity Maintenance Mechanism”, Conference on Information Sciences and Systems, Princeton University, March 1986.
6. 6 - W. Gleissner, “A Mathematical Theory for the Spread of Computer Viruses”, “Computers and Security”, IFIP TC-11, V8#1, Jan. 1989 pp35-41.
7. 7 - F. Cohen, “A Short Course on Computer Viruses”, ASP Press, PO Box 81270, Pittsburgh, PA 15217, 1990.
8. 8 - H. Highland, “Computer Virus Handbook”, Elsevier, 1990.
9. 9 - S. White, “A Status Report on IBM Computer Virus Research”, Italian Computer Virus Conference, 1990.
10. 10 - K. Brunnstein, “The Computer Virus Catalog”, DPMA, IEEE, ACM 4th Computer Virus and Security Conference, 1991 D. Lefkon ed.
11. 11 - F. Cohen, “Current Trends in Computer Virus Research”, 2nd Annual Invited Symposium on Computer Viruses - Keynote Address, Oct. 10, 1988. New York, NY
12. 12 - F. Cohen, “Models of Practical Defenses Against Computer Viruses”, IFIP-TC11, “Computers and Security”, V7#6, December, 1988.
13. 13 - M. Cohen, “A New Integrity Based Model for Limited Protection Against Computer Viruses”, Masters Thesis, The Pennsylvania State University, College Park, PA 1988.
14. 14 - F. Cohen, “A Cryptographic Checksum for Integrity Protection”, IFIP-TC11 “Computers and Security”, V6#6 (Dec. 1987), pp 505-810.
15. 15 - Y. Huang and F. Cohen, “Some Weak Points of One Fast Cryptographic Checksum Algorithm and its Improvement”, IFIP-TC11 “Computers and Security”, V8#1, February, 1989
16. 16 - F. Cohen, “A Cost Analysis of Typical Computer Viruses and Defenses”, IFIP-SEC “Computers and Security” (accepted, awaiting publication, also appearing in 4th DPMA, IEEE, ACM Computer Virus and Security Conference, 1991)
17. 17 - M. Pozzo and T. Gray, “An Approach to Containing Computer Viruses”, Computers and Security V6#4, Aug. 1987, pp 321-331
18. 18 - J. Page, “An Assured Pipeline Integrity Scheme for Virus Protection”, 12th National computer Security conference, Oct. 1989, pp 369-377
19. 19 - M. Bishop, “An Overview of Computer Viruses in a Research Environment”, 4th DPMA, IEEE, ACM Computer Virus and Security Conference, 1991
20. 20 - F. Cohen, “A Summary of Results on Computer Viruses and Defenses”, 1990 NIST/DOD Conference on Computer Security.

21. 21 - M. Harrison, W. Ruzzo, and J. Ullman, "Protection in Operating Systems", CACM V19#8, Aug 1976, pp461-471.
22. 22 - F. Cohen, "A DOS Based POset Implementation", IFIP-SEC TC11 "Computers and Security" (accepted, awaiting publication, 1991)
23. 23 - B. W. Lampson, "A note on the Confinement Problem", Communications of the ACM V16(10) pp613-615, Oct, 1973.
24. 24 - D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", The Mitre Corporation, 1973
25. 25 - F. Cohen, "A Note On High Integrity PC Bootstrapping", IFIP-SEC "Computers and Security" (accepted, awaiting publication, 1991)
26. 26 - M. Joseph and A. Avizienis "A Fault Tolerant Approach to Computer Viruses", Proceedings of the 1988 IEEE Symposium on Security and Privacy, 1989