Experiments on the Impact of Computer Viruses on Modern Computer Networks

by Dr. Frederick B. Cohen
and Sanjay Mishra

Abstract

In this paper, we describe some initial results of experiments on the impact of computer viruses on two modern computer networks. We begin by describing a research environment designed to allow safe and repeatable experiments with computer viruses. Next, we describe exhaustive tests of protection settings, their effectiveness against viruses in two very common computing environments, and other vulnerabilities encountered in the process of these experiments. Finally, we summarize results, describe confirming experiments performed by other researchers, and propose further work in this area.

1 Introduction and Background

Computer viruses [Cohen-84] have been a problem in personal computers for some time, and significant work has been done in this area to indicate, among other things, that the amount of information sharing is directly and positively related to the rate of computer virus spread. [Gleissner-89][White-90] The recent and dramatic increase in the use of computer networks for computer to computer communications and file sharing has significantly impacted the spread of viruses in organizations, to the point that in one case, viruses infected a network containing tens of thousands of computers within a matter of days [Rochlis-89] and it is now common for viruses to infect hundreds of networked computers in a matter of hours.

To date, few significant results have been published on the impact of computer viruses on networks, and even fewer have been published on the effectiveness of network protection mechanisms against viruses. To a certain extent, this is because of the inherent complexity in analyzing and performing experiments on networks. For example, modeling load and user behavior is quite difficult, networks are used for a wide variety of applications, and complex interactions can have significant impact on viral spread. Another complicating factor is the limited availability of isolated networks for safe experimentation.

In July of 1992, we configured a small experimental network for the sole purpose of performing experiments on the impact of viruses on modern networking environments. We performed a number of experiments over the next 3 weeks, and reported initial results [Cohen-92]. A number of researchers were initially surprised at these results, many systems administrators expressed serious concerns about them, and several people with experience in managing these networks had difficulty believing that these results could be accurate. Because of the complexity of networks and the resulting possibility for errors in experimental technique and in interpretation of manuals, we requested that other research groups perform independent experiments to confirm our refute our results.

Since that time, two groups [1] have performed substantial experiments and reported confirmations back to us, several other groups have stated that they will be performing similar experiments [2], our virus research network has been dismantled, [3] and we returned to our eternal quest for components for further experiments. In the process, we have also received a substantial amount of additional information relating to our published results, and thus we present this 'new and improved' paper on the impact of computer viruses on modern

[1]Vesselin Bontchev of the The University of Hamburg's 'Virus Test Center' and Padget Pederson

[2]several of these groups are from government agencies who will almost certainly not reveal their results

[3]We borrowed most of the hardware and software on a temporary basis, and had to return the component parts to the original users.

computer networks.

For the purposes of communication, we used the F-Prot product (version 2.03a) for virus identification and scanning. When we identify the Jerusalem virus, we intend to indicate the virus identified by F-Prot as the 'Jerusalem virus (standard)', and when we identify the 1704 virus, we intend to indicate the virus identified by F-Prot as the 'Cascade 1704-A virus'. Similarly, when we specify 'Netware', we intend to indicate Novell's Netware product, version 3.11, when we specify 'Unix', we intend to indicate AT+T's Unix System 5, Version 3.0 running TCP/IP and NFS, also by AT+T for PC compatibles, and when we specify PC-NFS, we intend to indicate the product of that name produced by Sun Computer Systems. We note that these specific systems may not be indicative of all other similar systems, but based on the confirming experiments, we believe they are representative.

2 The Purpose of the Virus Research Network

The Virus Research Network (VRN) was designed to allow rapid, comprehensive, conclusive, and repeatable experiments on the impact of viruses on networks. This was achieved to a reasonable extent by the methods outlined below. We are also interested in finding ways to improve network protection, and when we identified an anomalies, we sought causes and report the results.

Rapid experimentation is key to getting enough useful results to justify the cost of such a facility, but in our case, we had only a small window of availability for the facility, so necessity led us to efficiency. We got rapid results by automating most aspects of reconfiguration, and by using a backup computer to rebuild configurations in relatively short order. Depending on the particular experiment, the entire clean and rebuild process took as little as 20 minutes. In the worst case, cleanup required about 8 hours, and was done overnight.

Comprehensive experiments can often be performed through exhaustive testing. For example, in the experiments described later in this paper, we tested all possible protection settings for files and directories on the file server against some well known viruses. Given somewhat larger disks and more time, we could have performed still more comprehensive testing, but as we will see, this is not always required in order to get useful results.

Results were usually quite conclusive because we designed experiments with specific and attainable goals in mind, and had a facility which could be used for substantial run times without outside interference. Where possible, we tried to seek out more definitive experiments in favor of less definitive ones, and we used controls to assure that experimentation errors didn't cause false results.

We were relatively meticulous about documenting our experiments to assure that they were repeatable by outside researchers, and whenever reasonable, we used command scripts to automate operations. The use of automation in this case gives a high degree of repeatability. For example, one run may involve running several thousand programs in a particular order from a specific series of places in the network and with particular viruses active at particular times. It would be very difficult to perform or repeat such an experiment without automation.

In our first experimental run, these factors turned out to be quite critical, because the description of protection in the manuals of the network under test turned out to be inconsistent, and we had to repeat the entire run. Because of the automation, we were able to perform an identical 2 hour experiment involving over 2500 program runs under slightly different conditions with only 20 minutes of analysis, cleanup, and reconfiguration overhead.

As a part of the facility, we had a fairly substantial number of virus samples received by the Queensland University of Technology's Computer Virus Information Group (CVIG) as a part of its normal interaction with users. The selection of viruses for attacks was based on the prevalence of the virus in the environment, the techniques used by the virus for infection, and the defenses in place. We also created custom attacks where no virus in our collection was appropriate to the task. In the case of custom attacks, as a matter of policy we didn't provide copies of the attacks to others, but to date, this has not caused other researchers to report problems in generating confirming results.

3 The Virus Research Network

VRN consisted of 4 PC compatible computers in a physically isolated room. The computers were connected with Ethernet cable providing a relatively high speed communications backbone comparable to the most commonly used networking hardware today. The 'server' machine was a 80386 based computer with 4Mbytes of RAM and 100Mbytes of disk, required so that the commonly available network operating systems could run without further hardware. Two of the remaining computers were PC-AT class machines with 20Mbyte hard disks and 1Mbyte of memory required to run workstation software. The fourth machine had a tape drive which allowed it to perform rapid and reliable backup and restoration for the remainder of the network.

We pause to note that the 100Mbyte limitation on file server disk space restricted our ability to perform some exhaustive tests. For example, to test all file protections and directory protections under Unix requires 2^{18} files (2^9 different file protections in each of 2^9 different directories). To create the 262144 required files in 512 directories using only 1K per file requires over 250Mbytes. Since virus infections typically require more than another 1K of disk space, we could encounter conditions requiring over 500Mbytes of disk space. Many file systems also have difficulty handling that many files, and even more importantly, as we describe later, the complexity of updating protections in these circumstances makes accurate experiments difficult to perform.

In addition to physical isolation, all viruses used for experiments were kept on well marked floppy disks in a locked file cabinet, and all machines used in the network were clearly marked as contaminated. Machines were only allowed to leave the facility after the relatively thorough cleanup process described below, except in specific cases where well understood viruses were tested under very controlled circumstances. Floppy disks were not permitted to leave the facility without reformatting on a 'Clean'ed machine.

An additional machine which runs Coherent (a PC based Unix-like system) was used to create script files which automated configuration and operation of the network for experiments. This machine only produced disks for the network, and never took information out of the facility. Disks were transported back to this machine only after being formatted by the Backup machine and then again by the receiving machine. A typical run involved DOS batch files of over 50Kbytes, and could be created in a few minutes by the Coherent system with a minimum of effort.

VRN operated in three distinct operating modes:

1. **Run:** In the Run mode, the Backup machine was physically disconnected from the network so that it remained free from viruses. The server and the two workstations operated with viruses placed in the configuration as appropriate to the experiment.

2. **Clean:** In the Clean mode, each computer was rebooted from a write protected operating system disk using hardware reset, and all operations were performed from programs on that disk. Computers were scanned for known viruses using scanning software known to reliably detect the virus samples used for the experiments. These results were printed on a local printer for further analysis and as a record of operations. After analysis information was gathered, further experimentsm were sometimes performed by returning to the 'Run' mode, or the systems were prepared for rebuilding by performing a low level fixed disk format, preparing a known good partition table, and restoring basic operating system files from the floppy disk. After this procedure, the 'Rebuild' mode was entered.

3. **Rebuild:** In the Rebuild mode, the Backup computer was connected to the network, and configurations were restored to all of the other computers to prepare for the next experiment. After rebuilding was completed, the Backup computer was again disconnected from the network to allow further experiments to proceed. The backup

computer was always bootstrapped from a write protected floppy disk drive to further assure against contamination, and all systems were scanned for known viruses after cleanup as a control.

A copy of VRN can be configured for under US$6,000, and operating costs are predominantly driven by physical plant, people, and software costs. The development of experiments is also non-trivial, since issues arise about how to create and analyze accurate and efficient experiments, and experimental technique is of course critical. Nevertheless, we feel that any substantial organization with serious concerns about network protection could easily create and operate a similar facility.

4 Experiments

Our experiments were designed to test the efficacy of protection settings provided with network file servers against the most common computer viruses. We selected Netware and Unix for our experiments based on their availability in our laboratory and their widespread use in the current environment. We planned to perform similar experiments with other network configurations, with a variety of vendor-based defenses in place, and under a variety of usage profiles, but have been unable to get hardware, software, or support for further experiments as of this time.

Efficacy experiments are relatively easy to perform by creating an exhaustive test of protection settings for files and directories stored on the file server under the respective protection models. They are also useful because most environments operate without additional protection, and this permits us to provide accurate advice on prudent administrative measures appropriate to these environments. We thought at first that these experiments would not be particularly enlightening because if protection is properly implemented, the experiments will only reveal what we already know about the theoretical protection provided by the models. To a large extent, this was true, but as we will now see, the specific experiments and results indicate that implementation does not always meet expectations.

4.1 Netware

We installed a default Netware file server configuration per the instructions in the manuals that came with the system, and configured 256 subdirectories under the '\TEST' directory with names consisting of all possible combinations of the letters ‘Y’ and ‘N’ in the order shown below, and corresponding to the settings of the ‘Trustee Rights’ for a user ‘US1’ over those directories.

S	Supervisor
E	Erase
R	Read
W	Write
C	Create
A	Access Control
F	FileScan
M	Modify

Trustee Rights

As an example, the directory name ‘YNNYNNNY’ has ‘Supervisor’, ‘Write’, and ‘Modify’ rights enabled and all other rights disabled for user ‘US1’. We sometimes abbreviate by using exponention notation (e.g. NY^5NY is the same as NYYYYYNY).

Trustee rights were selected because they are supposed to dominate the ‘Inherited Rights’ implied by ANDing the parent directory’s effective rights with the ‘Inherited Rights Mask’, and therefore make analysis simpler without loss of generality. We did not test the assertion that Trustee Rights dominate Inherited Rights, and this should be confirmed by anyone interested in exhaustive testing of Netware protection.

Each of the resulting 256 directories were provided with 5 ‘EXE’ and 5 ‘COM’ executable files selected from standard DOS utilities such that they required relatively little disk space and run to completion producing only a syntax error when run from a command file with no command line arguments. This provides a method by which we can execute a large number of programs in a short period of time with a minimum of space overhead, and identifiable characteristics. The viruses selected for experiments were known to be able to infect these files and tested for this capability independently.

We initially turned on the ‘Copy Inhibit’, ‘Rename Inhibit’, ‘Delete Inhibit’, and ‘Archive’ file ‘Attributes’, setting one bit per file and leaving one file with no protection bits set as a control. These files were designated respectively ‘A-E.COM’ and ‘F-J.EXE’. In order to provide exhaustive testing of Attributes in combination with Trustee Rights, we would have to create all combinations of the 14 Netware Attributes with the 8 Trustee Rights shown above, leading to 2^{22} (4,194,304)) files in 512 directories, which would consume over 4 Gigabytes of storage and require over 48 days to test at 1 program execution per second. Instead, we selected the attributes which we thought to be of interest and tested them individually. A more thorough test would be appropriate for an organization with more resources.

In RUN 1, following the description in the manual, we enabled all rights for the '\TEST' directory. According to the manuals, subdirectory rights are supposed to 'take precedence' over parent rights (see P345 of the Netware 'Utilities Reference' manual and P207 of the Netware 'Concepts' manual). We then logged into the network as user 'US1', and for each directory, changed to that directory, executed the Jerusalem virus from the hard-disk on the local workstation (stored in the file 'C:\VIRUS.EXE'), executed each of the programs 'A-J' in order, executed 'VIRUS.EXE' again, and changed back to the '\TEST' directory. This was done in sequence starting at the Y^8 directory and continuing in standard binary counting sequence through to the N^8 directory. The run took about 2 hours to complete.

After completing the run, we rebooted the workstations from their write-protected floppy boot-disks, logged into the network as Supervisor, and scanned for known viruses on the network. This procedure assures that the workstations are not infected with the virus under test, that the scan is valid, and that the scan does not create infections where none existed before. To our surprise, we found that all of the files under test were infected, even those in directories to which the user 'US1' had no access! Upon deeper examination, we found that the Netware 'Supervisor' right dominates all other rights except when inherited rights cancel it's effects. Fortunately, our controls picked up this problem early on, and we were able to track down a resolution relatively quickly.

This problem is indicative of one of the major difficulties we encountered in examining Netware and many other protection systems. The manual attempts to explain protection in several different places, and as an aide to the less experienced user, makes several attempts to put protection in simple terms. Unfortunately, protection in Netware is not simple to understand or to manage, and the simplifications lead to serious misunderstandings. It was only the specific examples in one of the manuals in conjunction with our experiments that led us to our current understanding of Netware protection.

The problem of determining 'Effective Rights' (the Rights users actually have over files or directories) turns out to be quite complex in Netware, and we feel that this is a serious impediment to effective protection management. As an example of the complexity, each user's effective rights to a given file are dictated by 14 file attributes, 8 file trustee rights, 8 inheritance rights, and 16 rights (8 inherited and 8 trustee) per directory for each directory up to the root of the file system. At 4 directory levels down from the root, each file's accessibility is determined by about 100 bits of protection information. Similarly, changing any one protection bit may impact substantial portions of the file structure, and determining what is affected is a non-trivial problem.

The problem of determining effective rights is apparently so difficult that Netware can't immediately determine proper effective rights, which introduces yet another problem. In some cases, these settings don't take effect for some time, which leaves substantial windows of vulnerability and makes it somewhat complex to be certain that test results are accurate and repeatable. We did not thoroughly verify the inheritance of rights, and we encountered several anomalies that disappeared over time scales of a few minutes. By following the advice of one of the people who read about our initial results in the 'Risks' computer forum, we logged out and back in whenever we changed effective rights, and this seemed to seriously reduce the anomaly rate.

Now consider the problem of performing the exhaustive experiment with over 4 million files described earlier. This would mean that over 400 Million protection bits would have to be checked over the process of the experiment, and that the space consumed by protection bits alone comes to over 12Mbytes! How long might it take to update protection for 4 million files when it took several minutes for 2,500 files? If the process is linear with the number of files, we would expect a delay on the order of 5,000 minutes, or about 100 hours! Fortunately, the mechanism that causes this problem appears to be related to the cache mechanism used in Unix networks that we will describe later, and thus we don't expect a serious impact for larger file systems, but we have not performed tests to verify this, and suggest it as an open area for future experimentation.

This time variance also means that reliable programming in Netware is somewhat more complex than it would be in a more static situation, and many DOS programs make assumptions about the static nature of accessibility that will certainly make them unreliable in a non-static protection environment of this sort. This is not necessarily a problem to be resolved by Netware, since DOS programmers tend to make too many assumptions about file accessibility, and in any networking environment there will be changes, even if they are more predictable over time as Netware.

In RUN 2, we set the directory rights of '\TEST' to the Netware recommended values of 'Read' and 'FileScan' only, logged out and back in to assure that effective rights were accurately updated, and repeated the test from RUN 1 exactly as before. This time, our controls indicated that the Netware subdirectory rights worked correctly. According to RUN 2, the tested file attributes were completely irrelevant to protection from viruses. Again, this conflicts with the manuals (p227 of the Concepts Manual) which state that file attributes dominate directory rights. Upon further examination, we found that most (but not all) of the file attributes are DOS or MacIntosh protection bits, and further that they only apply to the operating system the user is using! Thus, Read-Only protection isn't supposed to work on MacIntosh computers, while the MacIntosh 'Inhibit' settings don't impact DOS access! Again, the complexity of mechanism introduced substantial problems, but in this case, the situation is far worse because protection seems to depend on DOS, which has no effective protection.

The results from RUN 2 can be summarized in one simple equation:

$$\overline{S}(\overline{F} + \overline{R} + \overline{W})$$

This is intended to convey that with the 'Supervisor' AND any of 'FileScan', 'Read', or 'Write' effective rights turned off for a given user, that user's PC being infected with the Jerusalem virus does NOT cause files in that directory to become infected. All of the other rights and attributes tested were apparently irrelevant to DOS virus protection.

We suspected that the reason 'FileScan' protection limited viral spread in this case was because the Jerusalem virus infects only on program execution, and the DOS command interpreter requires FileScan in order to execute programs. Thus the programs that could not be run were not infected. For this reason we were not yet convinced that FileScan being turned off was effective against viruses when programs are run from a DOS 'Exec' call, which does not have the same restriction as the command interpreter.

We eventually performed RUN 9 to determine the effectiveness of this protection equation against a slightly more sophisticated attack. In this case, we exhaustively tried to 'open' files in the protected directories for 'Read', 'Write', and 'Read/Write' access using a simple 'C' program. We found that files in $\mathrm{NY}^5\mathrm{NY}$ and NYNY^5 could be modified, leading to the conclusion that only $\mathrm{NY}^2\mathrm{NY}^4$ was effective. In other words, only $\overline{SW}$ was effective against this sort of attack. Since the 'A' attribute allows you to alter all but the 'S' attribute, $\overline{A}$ must also be included for effective protection, leading to the new equation:

$$\overline{SWA}$$

RUN 1 and RUN 2 results also led us to RUN 3 in which we wished to evaluate the effectiveness of the previous equation against a broader spectrum of viruses, and to evaluate the utility of the 'Read Only' file attribute in virus defense. RUN 3 required only 4 directories ('SR', 'SW', 'SF', and 'YYYYYYYY' corresponding to $\overline{SR}$, $\overline{SW}$, $\overline{SF}$, and full access respectively) with 4 files per directory (2 'COM' and 2 'EXE' files designated A.COM, B.COM, C.EXE, and D.EXE corresponding to Read-Only, Read-Write, Read-Only, and Read-Write respectively). The same process used in RUN 2 was used in RUN 3 except that each of the different viruses from a test suite was used in turn, only the 4 directories of interest were used, and the experiment was performed in the 'TEST2' directory.

RUN 3 surprisingly showed that 'Read-Only' is ineffective against the 1704 virus which succeeded in infecting 'A.COM' in the 'YYYYYYYY' directory even though it was write protected. No similar infection technique was available for 'EXE' files, and no virus in our experiment succeeded in infecting the 'C.EXE' program in the 'YYYYYYYY' directory, even though several infected the 'D.EXE' file. It turns out that this DOS attribute is trivially modified with the 'M' attribute turned on, and this attribute can in turn be altered if the 'A' attribute is enabled, leading to the second equation that could not be attacked by the methods described so far:

$$\overline{SAM}\mathrm{Ro}$$

Netware provides an Execute Only attribute which can never be unset once it is set, and which is supposed to prevent all subsequent access to the file. It does not prevent renaming or deleting of files, but prevents 'Read' and 'Write' operations to some extent, even against the Supervisor. Unfortunately, an Execute-Only file can be read by a user on a DOS based system by performing a DOS 'Load Program' call, and intercepting the file as it is passed to the loader. By reading it in this way, deleting the old version, and rewriting a new version, we can completely bypass Execute-Only protection, but a more dangerous attack appears to be presented by a 'Companion' virus.

We were successful in creating a 'Companion' virus example which was successful against Execute-Only protection. In this case, we rename the original executable and create a new executable containing the virus using the same name as the original executable, and giving it the same file size and other characteristics. [4] The replacement executable is then given the same file attributes as the original program, and made to automatically call the original after performing infection and/or damage. In this case, even the systems administrator cannot accurately detect the change because Execute-Only protection prevents the systems administrator from checking file contents or integrity. It would seem more appropriate to allow the systems administrator Read access to Execute-Only files so that they can be properly backed up and restored, and their integrity can be checked.

One idea for protection against this companion virus is to prevent renamimg of Execute-Only files, but the same attack can be made at any place up the directory tree from the Execute-Only file (i.e. by renaming a directory and placing a Trojan directory in it's place), so to be safe we would have to prevent directory renaming from the Execute-Only file up to the Root of the file system. This in turn would allow any user to prevent the systems administrator from renaming significant portions of the file structure by simply making a file Execute-Only. It would also force operations on directories to depend on all files lower then them in the file tree, and thus exacerbate the other protection problems we have discussed. It seems that Execute-Only protection in a network is ill-conceived except in systems where hardware protection and encryption make the implementation of such a facility feasible. [$White{-}87$]

To protect against these Companion viruses, you need to prohibit at least the Modify, AccessControl, Create, and System Rights, since trivial attacks can be found without these Rights eliminated. By combining the equations from the previous two strong configurations with this one, we reach the equation below:

$$\overline{SMAC}(\overline{W}+\mathrm{Ro})$$

This equation describes necessary but not sufficient settings for files and effective directory protection. We have found them effective in our experiments, and except for the time lag

[4]setting the characteristics in this way is not required in order for the virus to operate, but it makes detection far more difficult.

problem described earlier, they appear to be sound, but we have not tested them sufficiently to prove their correctness. This is in line with Netware's suggested 'RF' access rights. This protection setting also precludes changing information stored on the server, effectively making the server Read-Only! For directories where authorized users are permitted to make alterations, infections of files in those directories by those users is undefended by Netware, as in all other systems whose sole protection results from access control.

We also found from user reports, that a common method of virus spread in Netware networks comes from the infection of the 'Login.Exe' program stored on the server and used by all users to login to the network. As delivered, Netware allows the Supervisor to modify this file, and thus if the Supervisor EVER logs into a Netware server from an infected workstation, EVERY user that logs in subsequently may become infected. Furthermore, current versions of Netware cannot be logged into from the console, and thus once this file is infected, only a user with a local copy of a clean Login.Exe can ever login safely. Even worse, the Supervisor is automatically granted all rights to all files (except modification and read access to Execute-Only files) regardless of directory and file protections, so a Companion virus of the sort described above is UNDEFENDABLE with Netware access controls once the Supervisor logs in from an infected machine. Therefore, substantial protection must be provided against the Supervisor EVER logging in from an infected machine, or the entire protection scheme becomes ineffective. In a recent conference paper [Novell−92] Novell advised that a special user be created with 'RF' Trustee Rights to all files and directories, and that when a virus is suspected, this user be used to check for viruses. They also confirmed that logging in as Supervisor with a virus present presented a substantial danger and noted that many low-level viruses could infect Netware file servers.

All of the results presented above have now been substantially confirmed by independent research teams, and similar equations have been developed for older versions of Netware with substantially different Trustee Rights.

4.2 Unix

We installed a default Unix configuration per the manual provided with the system, specified a 'Read/Write' file structure accessible over the network by user 'US1' with an effective 'Root' directory corresponding to '\home\us1", and configured 512 directories belonging to user US1 and placed under US1's home directory, with names consisting of all possible combinations of octal protection codes depending on the placement of those rights in the standard Unix protection display as shown below:

	Owner	Group	World
read	400	40	4
write	200	20	2
execute	100	10	1

which is displayed as (by example) '561' to stand for 'read' and 'execute' for the 'owner' $(400 + 100 = 500)$, 'read' and 'write' for 'group' members $(40 + 20 = 60)$, and 'execute' for the rest of the 'world' $(500+60+1 = 561)$. Three users (us1, us2, and us3) were created with owner, group, and world identifications respectively so that the protection space is exhausted by accessing each directory from each user. We provided 3 executable files designated A-C.COM in each directory with protection settings of 666, 444, and 111 respectively.

As in the case of Netware, truly exhaustive testing was infeasible. To create the necessary 512 files of each sort ('.Exe' and '.Com') in each of 512 directories, would require over 500,000 files, over 250 Mbytes, and runtimes over 138 hours for a single test run at 1 second per program execution. Actually, our examination is ignoring an additional 2 protection bits per file (setUID and setGID), which grant the user running a program the effective rights of the owner or group respectively of the file, and therefore have a substantial impact on access rights. This would extend the time by a factor of 4 (from 5.8 days to over 23 days). Even though this time is not beyond the range of feasibility, our experiments were limited in time, so that we could not do these tests. Again, we urge any interested parties to pursue these tests for a more thorough examination of the problem.

One workstation was configured with PC-NFS software which allows remote file sharing, and was used for all of the experiments. Because Unix allows multiuser multiprocessing from the console, we were able to monitor operations and setup further experiments without additional workstations.

In RUN 4, we exhaustively tested directory protection settings and found that no program could be run without directory access of at least $R + X$ for the user, group, or world, depending on the rights associated with the logged in user. We also found that directory protections of each directory dominated protections of other directories, and thus that actual protection at any given moment are dictated by the protections of the directory containing the file. We were somewhat surprised to find that programs would run even though they had no read rights set for the user attempting to execute them. Since 'execute' requires 'read' from the file server to the PC (similar to the Execute-Only protection in Netware), this led us to RUN 5 in which we exhaustively explored file protections.

In RUN 5, we set out to exhaustively tested file protection settings in a single directory over which the user had full rights. We created 512 files in the 'TEST2' directory with names and protection settings in the range from 000_8 thru 777_8, all owned by user US1. The run began with a surprise. Not only could we execute programs with NO access rights for the user, but they also became infected! Thus, file access rights were completely ineffective! Then, another surprise. After running and infecting files 000, 001, 002, 003, and 004 the workstation could not continue processing. We rebooted and tried again, and found that the PC could not proceed past file 003. Subsequent retries resulted in the same problem. All of this is more than a bit disconcerting because a substantial history of experiments with Unix protection has shown file access rights to be quite effective against Unix programs making system calls. Hence, networked workstations appear to have more access than processes on the server itself.

Next, we explored the impact of directory protection on file protection and found that with a directory protection of 500, the owner could only execute programs which were readable by owner (i.e. $400_8 - 777_8$). This seemed appropriate, but we then found that all of the programs that were run were infected. Even files protected with settings of $400_8 - 577_8$ (i.e. those without Write enabled) in a directory protected 500 (i.e. without Write enabled) were infected!

Next, we tried similar experiments with another user, US2, first using identical groups, and then again with non-identical groups. We found that for directory protections allowing group or world 'Read' access, file access control determined infectability, while for all other directory access settings, no infection occurred. Files could be infected whenever both 'Write' access was enabled for the file's owner and both 'Read' and 'Write' access were available to the infected user. This is quite strange, since it is somewhat different from normal Unix protection. For example, Unix manuals commonly claim that Unix grants all 'World' rights to group members and owners, and all 'Group' rights to the owner, but in the networked example, group members and owners could not access files accessible by a 'World' user. Similarly, a 'World' user could read and modify a file that the file's owner could not read in a directory to which the owner had no access!

This combination of rights applied separately to files and directories is quite strange in some cases. For example, a file with access rights of 206 and group ID 100 in a directory with access rights of 050 and group ID 200 could be infected by a user in group 100, but not by the owner or a user in group ID 200. This is far different from the results you would get when logged in as a Unix user to the same system.

We conclude that the NFS network drivers do not use normal file access controls, and suspect that the entire file system is susceptible to attack, at least from the effective 'Root' directory down the directory tree. We were unable to access files in other portions of the file server's directory space, and suspect that this aspect of protection is effective, except for the fact that symbolic links could completely bypass any such mechanism, and that the network interface operates on Unix i-nodes, which may be exploited to gain access to otherwise inaccessible areas of the same file system.

Directory protection appears to be effective to the extent that preventing all access works, but clearly write prevention is a critical protection need if we are to run programs without modifying them, and this was ineffective at the directory level. We did not explore the impact of symbolic links, but believe it will lead to further problems of this sort. Again, we believe that directory access controls may be bypassed through direct access to i-nodes.

Clearly, a simple virus could alter file protections, alter files, and restore file protections, and thus, even effective file access controls would not alter the situation in the case of a serious attacker. Thus, a file's owner is almost always able to cause infection, regardless of protection settings, while other users may be able to infect a file even if the owner doesn't have write access enabled.

A further aspect of Unix based remote file sharing is that in all the examples we have seen, file sharing allows access by multiple users from a single remote computer based on user IDs and passwords provided on the remote machine only. In other words, the root user on any machine in the network authorized to perform remote file sharing has the ability to grant or deny effective access rights to any user ID on the server. More specifically, any remote machine which can be taken over by an unauthorized user or process, can access all files on the mounted file system on the server except those owned by user identities explicitly remapped through the Unix user ID remapping capability associated with remote file system mounts. In the case of PCs running the DOS operating system, viruses designed to take advantage of this problem should be able to gain all access attainable by all combinations of users with access.

Remote file sharing under Unix also permits virtual file systems to be mounted as 'Read-Only', which we did not test, and mounting of those file systems with specific user ID's remapped, which we also did not test. Unlike Netware, Unix servers can be used from the console, and use different executable files than DOS, and thus we believe that with prudence, we may be able to prevent the infection of the file server operating system, but this is by no means certain. As in Netware and all other systems we are aware of, viruses can spread through the normal access rights regardless of the inadequacies of the particular implementation.

Unix based LANs also have a serious problem caused by the delay between the setting of protections and the effect of those settings on files. In particular, when remote systems use disk cacheing for network files systems (which can be disabled or enabled on each machine in the network), changes in protection settings on one machine may not appear on other machines for some time. For example, if machine 'A' is connected to a remote file system on server 'S', and A is doing disk cacheing, then a change in file protection made on S will not be 'written through' to A immediately. Thus, a file protection change which prevents access on S will not prevent that access from A, assuming A has cached the protection information and thus does not have to look at the current value on S to determine accessibility.

This attack is theoretical at this time, and has not been confirmed by experiment, however, it is apparent from the manner in which the system operates and from examination of the source code of '386BSD' Unix [5], that this will happen. Cache updates are often delayed by as much as 30 seconds. This also implies that other information which is cached may be inconsistent when viewed from various systems in a network. This leads to serious potential problems in the use of shared databases based on file access rather than remote procedure calls and server based locking mechanisms. Even a 'rename' operation may not take effect for some time on remote machines.

In the case of Unix, we also feel it is important to note that there are a multitude of different vendors with different version of the operating system, and that the results from this particular version may not reflect the entire field. We also feel compelled to mention that the basic networking scheme used by Unix systems is flawed in that it extends too many privileges to the network drivers, has inadequate authentication, and is not designed with the uniform view of protection taken by the remainder of the Unix operating system. These inconsistencies appear to lead to implementation flaws such as those we have noted here.

We also note that the vast majority of Unix systems appear to be derivitives of only a few vendor's original source codes, and that the protection decisions in almost all Unix systems have been left unchanged and unexamined for something like 20 years. For example, the 386BSD Unix access control decisions are apparently verbatim copies of those from BSD Unix, which according to several sources are identical to those made in System 5 Unix. Thus would imply that the vast majority of vendor protection code is identical and that these fundamental flaws in one system appear in the vast majority of other systems. We would thus expect that the strange access control behavior would appear on many systems, and hope that others will perform experiments in other environments to confirm or refute these results for those systems.

5 Summary, Conclusions, and Further Work

This research is still in its early stages, but we have already gained a great deal of insight into the protection provided both against viruses and in a more general sense by the protection mechanisms of two of the most commonly used network file servers. Our results show that viruses are a serious threat in all of these environments, and the protection mechanisms in place are less effective than we would expect. In effect, the only safe file server is the Read-Only file server, and even then, there are serious threates from Supervisors under

[5] A version of BSD Unix freely available in source form

Novell and serious attackers under Unix.

More specifically, re;latively safe protection settings for Netware were identified as $\overline{SMAC}(\overline{W}+\text{Ro})$, relatively safe protection settings for Unix were identified as $\overline{R_{directory}+W_{owner}}$, and a number of serious problems have been revealed in both Unix and Netware.

It seems clear that architecturally, there are flaws in Netware and Unix networking that make network protection difficult to attain. Netware has difficult to understand and maintain protection settings, and while Unix servers seem to have far better consistency in the meaning of protection settings, the implementation fails to meet the theory.

In terms of automation, neither system provides any substantial useful automation for either the user or the administrator to maintain a consistently well protected environment. The tools for protection provided with these file servers are far too difficult to use for effective protection, and without adequate tools, it is unrealistic to expect even expert administrators to do an adequate job. Better tools are required if the virus threat to these networks is to be controlled effectively through the protection mechanisms provided.

As we have discussed, our results at this time are preliminary, and a great deal of further research is required in order to get an effective handle on this problem over the full spectrum of networking environments available today. Studies of spread rates would be a valuable tool in determining cost effective defenses. More thorough studies of the effectiveness of network protection are called for both in the environments discussed here and in other environments not yet studied systematically. We have not yet started to study the effectiveness of these environments when additional protection measures such as virus scanning and integrity mechanisms are put in place, and this is an area where the results would be directly applicable to current users. A great deal of work is also called for in the area of protection management and the design of tools to automate systems administration so as to allow normal administrators to provide effective protection without undue effort.

We hope that others will follow in our footsteps, extend our results, perform exhaustive tests we were unable to perform, and carry on some of the many other experiments required to address the LAN virus protection issue. More importantly, we hope vendors will start a program of systematic, comprehensive, and regular testing to detect and resolve similar problems before release. Finally, we hope that these results will lead to a deeper study of network protection issues, which are clearly underconsidered in current networking environments.

References

[1] F. Cohen, “Computer Viruses - Theory and Experiments”, IFIP-sec 84, also appearing in IFIP-TC11 “Computers and Security”, V6(1987), pp22-35.

[2] F. Cohen and S. Mishra, "Some Initial Results From The QUT Virus Research Network", 2nd Virus Bulletin Conference, Edinburgh Scotland, Aug, 1992

[3] W. Gleissner, "A Mathematical Theory for the Spread of Computer Viruses", "Computers and Security", IFIP TC-11, V8#1, Jan. 1989 pp35-41.

[4] S. White, "A Status Report on IBM Computer Virus Research", Italian Computer Virus Conference, 1990.

[5] S. White and D. Chess, "The Abyss Processor", IEEE Computer Security and Privacy Conference, Oakland, CA 1987

[6] J. Rochlis and M. Eichin, "With Microscope and Tweezers: The Worm from MIT's Perspective", CACM, V32#6, June, 1989.