With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988 *

Mark W. Eichin and Jon A. Rochlis

Massachusetts Institute of Technology
77 Massachusetts Avenue, E40-311
Cambridge, MA 02139

February 9, 1989

Abstract

In early November 1988 the Internet, a collection of networks consisting of 60,000 host computers implementing the TCP/IP protocol suite, was attacked by a virus, a program which broke into computers on the network and which spread from one machine to another. This paper is a detailed analysis of the virus programitself, as well as the reactions of the besieged Internet community. We discuss the structure of the actual program, as well as the strategies the virus used to reproduce itself. We present the chronology of events as seen by our team at MIT, one of a handful of groups around the country working to take apart the virus, in an attempt to discover its secrets and to learn the network's vulnerabilities. We describe the lessons that this incident has taught the Internet community and topics for future consideration and resolution. A detailed routine by routine description of the virus program including the contents of its built in dictionary is provided.

1 Introduction

The Internet[1][2], a collection of interconnected networks linking approximately 60,000 computers, was attacked by a virus program on 2 November 1988. The Internet community is comprised of academic, corporate, and goverment research users, all seeking to exchange information to enhance their research efforts.

The virus broke into Berkeley Standard Distribution (BSD) UNIX[1] and derivative systems. Once resident in a computer, it attempted to break into other machines on the network. This paper is an analysis of that virus program and of the reaction of the Internet community to the attack.

1.1 Organization

In Section 1 we discuss the categorization of the program which attacked the Internet, the goals of the teams working on isolating the virus and the methods they employed, and summarize what the virus did and did not actually do. In Section 2 we discuss in more detail the strategies it employed, the specific attacks it used, and the effective and ineffective defenses proposed by the community. Section 3 is a detailed presentation of the chronology of the virus. It describes how our group at MIT found out and reacted to the crisis, and relate the experiences and actions of select other groups throughout the country, especially as they interacted with our group. Once the crisis had passed, the Internet community had time not only to explore the vulnerabilities which had allowed the attack to succeed, but also to consider how future attacks could be prevented. Section 4 presents our views on the lessons learned and problems to be faced in the future. In Section 5 we acknowledge the people on our team and the people at other sites who aided us in the effort to understand the virus.

We present a subroutine by subroutine description of the virus program itself in Appendix A, including a diagram of the information flow through the routines which comprise the ‘‘cracking engine’’. Appendix B contains a list of the words included in the built-in dictionary carried by the virus. Finally in Appendix C we provide an alphabetized list of all the people mentioned in this paper, their affiliations, and their network mail addresses.

 A version of this paper will be presented at the 1989 IEEE Symposium on Research in Security and Privacy.

[1] UNIX is a trademark of AT&T. DEC, VAX, and Ultrix are trademarks of Digitial Equipment Corporation. Sun, SunOS, and NFS are trademarks of Sun Microsystems, Inc. IBM is a trademark of International Business Machines, Inc.

1.2 A Rose by Any Other Name

The question of how to classify the program which infected the Internet has received a fair amount of attention. Was it a ''virus'' or ''worm''; or was it something else?

There is confusion about the term ''virus.'' To a biologist a virus is an agent of infection which can only grow and reproduce within a host cell. A lytic virus enters a cell and uses the cell's own metabolic machinery to replicate. The newly created viruses (more appropriately called ''virons'') break out of the infected cell, destroying it, and then seek out new cells to infect. A lysogenetic virus, on the other hand, alters the genetic material of its host cells. When the host cell reproduces it unwittingly reproduces the viral genes. At some point in the future, the viral genes are activated and many virons are produced by the cell. These proceed to break out of the cell and seek out other cells to infect[3]. Some single strand DNA viruses do not kill the host cell; they use the machinery of the host cell to reproduce (perhaps slowing normal celluar growth by diverting resources) and exit the cells in a non-destructive manner[4].

A ''worm'' is an organism with an elongated segmented body. Because of the shape of their bodies worms can snake around obstacles and work their way into unexpected places. Some worms, for example the tapeworm, are parasites. They live inside of a host organism, feeding directly from nutrients intended for host cells. These worms reproduce by shedding one of their segments which contains many eggs. They have difficulty in reaching new hosts, since they usually leave an infected host through its excretory system and may not readily come into contact with another host[5].

In deciding which term fits the program which infected the Internet, we must decide which part of the system is analogous to the ''host''. Possibilities include the network, host computers, programs, and processes. We must also consider the actions of the program and its structure.

Viewing the network layer as the ''host'' is not fruitful; the network was not attacked, specific hosts on the network were. The infection never spread beyond the Internet even though there were gateways to other types of networks. One could view the infection as a worm, which ''wiggled'' throughout the network. But as Beckman points out[6] the program didn't have connected ''segments'' in any sense. Thus it can't be a worm.

A model showing the computers as the ''host'' is more promising. The infection of 2 November entered the hosts, reproduced, and exited in search of new hosts to infect. Some people might argue that since the host was not destroyed in this process, that the infecting program was more like a worm than a virus. But, as mentioned earlier, not all viruses destroy their host cells. Denning [7] defines a computer worm as a program which enters a workstation and disables it. In that sense the infection could be considered a worm, but we reject this definition. The infected computers were affected but not all were ''disabled''. There is also no analog to the segments of a biological worm.

Denning has described how many personal computer programs have been infected by viral programs[7]. These are frequently analogous to lysogenetic viruses because they modify the actual program code as stored in the computer's secondary storage. As the infected programs are copied from computer to computer through normal software distribution, the viral code is also copied. At some point the viral code may activate and perform some action such as deleting files or displaying a message. Applying this definition of a virus while viewing programs as ''hosts'' does not work for the Internet infection, since the virus neither attacked nor modified programs in any way.

If, however, processes are view as ''hosts'', then the Internet infection can clearly be considered a viral infection. The virus entered hosts through a daemon process, tricking that process into creating a viral process, which would then attempt to reproduce. In only one case, the finger attack, was the daemon process actually changed; but as we noted above only lysogenetic viruses actually change their host's genetic material.

Denning defines a bacterium as a program which replicates itself and feeds off the host's computational resources. While this seems to describe the program which infected the Internet, it is an awkward and vague description which doesn't seem to convey the nature of the infection at all.

Thus we have chosen to call the program which infected the Internet a virus. We feel it is accurate and descriptive.

1.3 Goals and Targets

The program that attacked many Internet hosts was itself attacked by teams of programmers around the country. The goal of these teams was to find out *all* the inner workings of the virus. This included not just understanding how to stop further attacks, but also understanding whether any permanent damage had been done, including destruction or alteration of data during the actual infection, or possible ''time bombs'' left for later execution.

There were several steps in achieving these goals: including

- isolating a specimen of the virus in a form which could be analyzed.
- ''decompiling'' the virus, into a form that could be shown to reduce to the executable of the real thing, so that the higher level version could be interpreted.
- analyzing the strategies used by the virus, and the elements of its design, in order to find weaknesses and methods of defeating it.

The first two steps were completed by the morning of 4 November 1988. Enough of the third was complete to determine that the virus was harmless, but there were no clues to the higher level issues, such as the reason for the virus' rapid spread.

Once the decompiled code existed, and the threat of the virus known to be minimal, it was clear to the MIT team and those at Berkeley that the code should be protected. We understood that the knowledge required to write such a program could not be kept secret, but felt that if the code were publicly available, someone could too easily modify it and release a damaging mutated strain. If this occurred before many hosts had removed the bugs which allowed the penetration in the first place, much damage would be done.

There was also a clear need to explain to the community what the virus was and how it worked. This information, in the form of this report, can actually be *more* useful to interested people than the source code could be, since it includes discussion of the side effects and results of the code, as well as flaws in it, rather than merely listing the code line by line. Conversely, there are people interested in the intricate detail of how and why certain routines were used; there should be enough detail here to satisfy them as well. Readers will also find Seely[8] and Spafford's[9] papers interesting.

1.4 Major Points

This section provides an outline of the how the virus attacked and who it attacked. It also lists several things the virus did not do, but which many people seem to have attributed to the virus. All of the following points are described in more detail in Section 2.

1.4.1 How it entered

- sendmail (needed debug mode, as in SunOS binary releases)
- finger[10] (only VAX hosts were victims)
- remote execution system, using
  - rexec
  - rsh

1.4.2 Who it attacked

- accounts with obvious passwords, such as
  - none at all
  - the user name
  - the user name appended to itself
  - the "nickname"
  - the last name
  - the last name spelled backwards
- accounts with passwords in a 432 word dictionary (see Appendix B)
- accounts with passwords in `/usr/dict/words`
- accounts which trusted other machines via the `.rhosts` mechanism

1.4.3 What it attacked

- SUNs and VAXes only
- machines in `/etc/hosts.equiv`
- machines in `/.rhosts`
- machines in cracked accounts' `.forward` files
- machines in cracked accounts' `.rhosts` files
- machines listed as network gateways in routing tables
- machines at the far end of point-to-point interfaces
- possibly machines at randomly guessed addresses on networks of first hop gateways

1.4.4 What it did NOT do

- gain privileged access (it almost never broke in as `root`)
- destroy or attempt to destroy any data
- leave time bombs behind
- differentiate among networks (such as MILNET, ARPANET)
- use UUCP at all
- attack specific well-known or privileged accounts such as `root`

2 Strategies

2.1 Attacks

This virus attacked several things, directly and indirectly. It picked out some deliberate targets, such as specific network daemons through which to infect the remote host. There were also less direct targets, such as mail service and the flow of information about the virus.

2.1.1 Sendmail Debug Mode

The virus exploited the "debug" function of `sendmail`, which enables debugging mode for the duration of the current connection. Debugging mode has many features, including the ability to send a mail message with a program as the recipient (i.e. the program would run, with all of its input coming from the body of the message). This is inappropriate and rumor[11] has it that the author included this feature to allow him to circumvent security on a machine he was using for testing. It certainly exceeds the intended design of the Simple Mail Transfer Protocol (SMTP) [12].

Specification of a program to execute when mail is received is normally allowed in the `sendmail` aliases file or users' `.forward` files directly, for `vacation` [2], mail archive programs, and personal mail sorters. It is *not* normally allowed for incoming connections. In the virus, the

[2] A program which accepts incoming mail and sends back mail to the original sender, usually saying something like "I am on vacation, and will not read your mail until I return."

‘‘recipient’’ was a command to strip off the mail headers and pass the remainder of the message to a command interpreter. The body was a script that created a C program, the ‘‘grappling hook,’’ which transfered the rest of the modules from the originiating host, and the commands to link and execute them. Both VAX and Sun binaries were transfered and both would be tried in turn, no attempt to determine the machine type was made. On other architectures the programs would not run, but would use resources in the linking process. All other attacks used the same ‘‘grappling hook’’ mechanism, but used other flaws to inject the ‘‘grappling hook’’ into the target machine.

The fact that debug was enabled by default was reported to Berkeley by several sources during the 4.2BSD release. The 4.3BSD release as well as Sun releases still had this option enabled by default [13]. The then current release of Ultrix did not have debug mode enabled, but the beta test version of the newest release did have debug enabled (it was disabled before finally being shipped). MIT’s Project Athena was among a number of sites which went out of its way to disable debug mode; however, it is unlikely that many binary-only sites were able to be as diligent.

2.1.2 Finger Daemon Bug

The virus hit the finger daemon (`fingerd`) by overflowing a buffer which was allocated on the stack. The overflow was possible because `fingerd` used a library function which did not do range checking. Since the buffer was on the stack, the overflow allowed a fake stack frame to be created, which caused a small piece of code to be executed when the procedure returned [3]. The library function in question turns out to be a backward-compatibility routine, which should not have been needed after 1979 [14].

Only 4.3BSD VAX machines were attacked this way. The virus did not attempt a Sun specific attack on finger and its VAX attack failed when invoked on a Sun target. Ultrix was not vulnerable to this since production releases did not include a `fingerd`.

2.1.3 Rexec and Passwords

The virus attacked using the Berkeley remote execution protocol, which required the user name and plaintext password to be passed over the net. The program only used pairs of user names and passwords which it had already tested and found to be correct on the local host. A common, world readable file (`/etc/passwd`) that contains the user names and encrypted passwords for every user on the system facilitated this search. Specifically:

[3] MIT’s Project Athena has a ‘‘write’’ daemon which has a similar piece of code with the same flaw but it explicitly exits rather than returning, and thus never uses the (damaged) return stack. A comment in the code notes that it is mostly copied from the finger daemon.

- this file was an easy-to-obtain list of user names to attack,
- the dictionary attack was a method of verifying password guesses which would not be noted in security logs.

The principle of ‘‘least privilege’’ [15] is violated by the existence of this password file. Typical programs have no need for a list of user names and password strings, so this privileged information should not be available to them. For example, Project Athena’s network authentication system, *Kerberos* [16], keeps passwords on a central server which logs authentication requests, thus hiding the list of valid user names. However, once a name is found, the authentication ‘‘ticket’’ is still vulnerable to dictionary attack.

2.1.4 Rsh and Trust

The virus attempted to use the Berkeley remote shell program (called `rsh`) to attack other machines without using passwords. The remote shell utility is similar in function to the remote execution system, although it is ‘‘friendlier’’ since the remote end of the connection is a command interpreter, instead of the *exec* function. For convenience, a file `/etc/hosts.equiv` can contain a list of hosts trusted by this host. The `.rhosts` file provides similar functionality on a per-user basis. The remote host can pass the user name from a trusted port (one which can only be opened by `root`) and the local host will trust that as proof that the connection is being made for the named user.

This system has an important design flaw, which is that the local host only knows the remote host by its network address, which can often be forged. It also trusts the machine, rather than any property of the user, leaving the account open to attack at all times rather than when the user is present [16]. The virus took advantage of the latter flaw to propagate between accounts on trusted machines. Least privilege would also indicate that the lists of trusted machines be only accessible to the daemons who need to decide to whether or not to grant access.

2.1.5 Information Flow

When it became clear that the virus was propagating via `sendmail`, the first reaction of many sites was to cut off mail service. This turned out to be a *serious* mistake, since it cut off the information needed to fix the problem. Mailer programs on major forwarding nodes, such as *relay.cs.net*, were shut down delaying some critical messages by as long as twenty hours. Since the virus had alternate infection channels (`rexec` and `finger`), this made the isolated machine a safe haven for the virus, as well as cutting off information from machines further ‘‘downstream’’ (thus placing them in greater danger) since no information about the virus could reach them by mail[4]. Thus, by attacking `sendmail`, the virus indirectly attacked the flow of information that was the only real defense against its spread.

2.2 Self Protection

The virus used a number of techniques to evade detection. It attempted both to cover it tracks and to blend into the normal UNIX environment using camouflage. These techniques had had varying degrees of effectiveness.

2.2.1 Covering Tracks

The program did a number of things to cover its trail. It erased its argument list, once it had finished processing the arguments, so that the process status command would not show how it was invoked.

It also deleted the executing binary, which would leave the data intact but unnamed, and only referenced by the execution of the program. If the machine were rebooted while the virus was actually running, the file system salvager would recover the file after the reboot. Otherwise the program would vanish after exiting.

The program also used resource limit functions to prevent a core dump. Thus, it prevented any bugs in the program from leaving tell-tale traces behind.

2.2.2 Camouflage

It was compiled under the name `sh`, the same name used by the Bourne Shell, a command interpreter which is often used in shell scripts and automatic commands. Even a diligent system manager would probably not notice a large number of shells running for short periods of time.

The virus forked, splitting into a parent and child, approximately every three minutes. The parent would then exit, leaving the child to continue from the exact same place. This had the effect of ‘‘refreshing’’ the process, since the new fork started off with no resources used, such as CPU time or memory usage. It also kept each run of the virus short, making the virus a more difficult to seize, even when it had been noticed.

All the constant strings used by the program were obscured by XOR'ing each character with the constant 81_{16}. This meant that one could not simply look at the binary to determine what constants the virus refered to (e.g. what files it opened). But it was a weak method of hiding the strings; it delayed efforts to understand the virus, but not for very long.

[4] USENET news [17] was an effective side-channel of information spread, although a number of sites disabled that as well.

2.3 Flaws

The virus also had a number of flaws, ranging from the subtle to the clumsy. One of the later messages from Berkeley posted fixes for some of the more obvious ones, as a humorous gesture.

2.3.1 Reinfection prevention

The code for preventing reinfection of an actively infected machine harbored some major flaws. These flaws turned out to be critical to the ultimate ‘‘failure’’ of the virus, as reinfection drove up the load of many machines, causing it to be noticed and thus counterattacked.

The code had several timing flaws which made it unlikely to work. While written in a ‘‘paranoid’’ manner, using weak authentication (exchanging ‘‘magic’’ numbers) to determine whether the other end of the connection is indeed a copy of the virus, these routines would often exit with errors (and thus *not* attempt to quit) if:

- several viruses infected a clean machine at once, in which case all of them would look for listeners; none of them would find any; all of them would attempt to become listeners; one would succeed; the others would fail, give up, and thus be invulnerable to future checking attempts.
- several viruses starting at once, in the presence of a running virus. If the first one ‘‘wins the coin toss’’ with the listening virus, other new-starting ones will have contacted the losing one and have the connection closed upon them, permitting them to continue.
- a machine is slow or heavily loaded, which could cause the virus to exceed the timeouts imposed on the exchange of numbers, especially if the compiler was running (possibly multiple times) due to a new infection; note that this is exacerbated by a busy machine (which slows down further) on a moderately sized network.

Note that ‘‘at once’’ means ‘‘within a 5-20 second window’’ in most cases, and is sometimes looser.

A critical weakness in the interlocking code is that even when it *does* decide to quit, all it does is set the variable `pleasequit`. This variable does not have an effect until the virus has gone through

- collecting the entire list of host names to attack
- collecting the entire list of user names to attack
- trying to attack all of the ‘‘obvious’’ permutation passwords (see Section A.4.3)
- trying ten words selected at random from the internal dictionary (see Appendix B) against all of the user names

Since the virus was careful to clean up temporary files, its presence alone didn't interfere with reinfection.

Also, a multiply infected machine would spread the virus faster, perhaps proportionally to the number of infections it was harboring, since

- the program scrambles the lists of hosts and users it attacks; since the random number generator is seeded with the current time, the separate instances are likely to hit separate targets.
- the program tries to spend a large amount of time sleeping and listening for other infection attempts (which never report themselves) so that the processes would share the resources of the machine fairly well.

Thus, the virus spread much more quickly than the perpetrator expected, and was noticed for that very reason. The MIT Media Lab, for example, cut themselves completely off from the network because the computer resources absorbed by the virus were detracting from work in progress, while the lack of network service was a minor problem.

2.3.2 Heuristics

One attempt to make the program not waste time on non-UNIX systems was to sometimes try to open a telnet or rsh connection to a host before trying to attack it and skipping that host if it refused the connection. If the host refused telnet or rsh connections, it was likely to refuse other attacks as well. There were several problems with this heuristic:

- A number of machines exist which provide mail service (for example) but that do not provide telnet or rsh service, and although vulnerable, would be ignored under this attack. The MIT Project Athena mailhub, *athena.mit.edu*, is but one example.
- The telnet ‘‘probing’’ code immediately closed the connection upon finding that it had opened it. By the time the ‘‘inet daemon’’, the ‘‘switching station’’ which handles most incoming network services, identified the connection and started a telnet daemon, the connection was already closed, causing the telnet daemon to indicate an error condition of high enough priority to be logged on most systems. Thus the times of the earliest attacks were noted, if not the machines they came from.

2.3.3 Vulnerabilities not used

The virus did not exploit a number of obvious opportunities.

- When looking for lists of hosts to attack, it could have done ‘‘zone transfers’’ from the Internet domain name servers to find names of valid hosts [18]. Many of these records also include host type, so the search could have limited itself to the appropriate processor and operating system types.
- It did not attack both machine types consistently. If the VAX finger attack failed, it could have tried a Sun attack, but that hadn’t been implemented.
- It did not try to find privileged users on the local host (such as `root`).

2.4 Defenses

There were many attempts to stop the virus. They varied in inconvenience to the end users of the vulnerable systems, in the amount of skill required to implement them, and in their effectiveness.

- Full isolation from network was frequently inconvenient, but was very effective in stopping the virus, and was simple to implement.
- Turning off mail service was inconvenient both to local users and to ‘‘downstream’’ sites, was ineffective at stopping the virus, but was simple to implement.
- Patching out the `debug` command in `sendmail` was only effective in conjunction with other fixes, did not interfere with normal users, and simple instructions for implementing the change were available.
- Shutting down the finger daemon was also effective only if the other holes were plugged as well, was annoying to users if not actually inconvenient, and was simple to perform.
- Fixing the finger daemon required source code, but was as effective as shutting it down, without annoying the users at all.
- `mkdir /usr/tmp/sh` was convenient, simple, and effective in preventing the virus from propagating [5] (See Section A.8.2.)
- Defining `pleasequit` in the system libraries was convenient, simple, and did almost nothing to stop the virus (See Section A.3.2.)
- Renaming the UNIX C compiler and linker (`cc` and `ld`) was drastic, and somewhat inconvenient to users (though much less so than cutting off the network, since different names were available) but effective in stopping the virus.
- Requiring new passwords for all users (or at least all users who had passwords which the virus could guess) was difficult, but it only inconvenienced those users with weak passwords to begin with, and was effective in conjunction with the other fixes (See Section A.4.3 and Appendix B.)

After the virus was analyzed, a tool which duplicated the password attack (including the virus’ internal dictionary) was posted to the network. This tool allowed system administrators to analyze the passwords in use on their system. The spread of this virus should be effective in raising the awareness of users (and administrators) to the importance of choosing ‘‘difficult’’ passwords. Lawrence Livermore National Laboratories went as far as requiring all passwords be changed, and modifying the password changing program to test new passwords against the lists that include the passwords attacked by the virus [6].

[5] However, both sets of binaries were still compiled, consuming processor time on an attacked machine.

3 Chronology

This is a description of the chronology of the virus, as seen from MIT. It is intended as a description of how one major Internet site discovered and reacted to the virus. This includes the actions of our group at MIT which wound up decompiling the virus and discovering its inner details, and the people across country who were mounting similar efforts. It is our belief that the people involved acted swiftly and effectively during the crisis and deserve many thanks. Also, there is much to be learned from the way events unfolded. Some clear lessons for the future emerged, and as usual, many unresolved and difficult issues have also risen to the forefront to be considered by the networking and computer community.

The events described took place between Wednesday 2 November 1988 and Friday 11 November 1988. All times are stated in eastern standard time.

3.1 Wednesday: Genesis

Gene Myers[6] of the NCSC analyzed the Cornell[6] mailer logs. He found that testing of the `sendmail` attack first occurred on 19 October 1988 and continued through 28 October 1988. On 29 October 1988, there was an increased level of testing; Gene believes the virus author was attempting to send the binaries over the SMTP connections, an attempt which was bound to fail since the SMTP is only defined for 7 bit ASCII data transfers[12]. The author appeared to go back to the drawing board, returning with the ''grappling hook'' program (see section A.7) on Wednesday 2 November 1988. The virus was tested or launched at 5:01:59pm. The logs show it infecting a second Cornell machine at 5:04pm. This may have been the genesis of the virus, but that is disputed by reports in the New York Times[11] in which Paul Graham of Harvard states the virus started on a machine at the MIT AI Lab via remote login from Cornell. Cliff Stoll of Harvard also believes that the virus was started from the MIT AI Lab. At the time this paper was written, nobody has analyzed the infected Cornell machines to determine where the virus would have gone next if they were indeed the first infected machines.

In any case, Paul Flaherty of Stanford reported to the *tcp-group@ucsd.edu* mailing list on Friday that Stanford was infected at 9:00pm and that it got to ''most of the campus UNIX machines (cf. ˜ 2500 boxes).'' He also reported the virus originated from *prep.ai.mit.edu*. This is the earliest report of the virus we have seen.

At 9:30pm Wednesday, *wombat.mit.edu*, a private workstation at MIT Project Athena maintained by Mike Shanzer was infected. It was running a version of `sendmail` with the `debug` command turned on. Mike believes that the attack came from *prep.ai.mit.edu* since he had an account on *prep* and *wombat* was listed in his `.rhosts`, a file which specifies a list of hosts and users on those hosts who may log into an account over the network without supplying a password. Unfortunately the appropriate logs were lost, making the source of the infection uncertain. (The logs on *prep* were forwarded via `syslog`, the 4.3BSD UNIX logging package, to another host which was down and by the time anybody looked the `wtmp` log, which records logins, it was truncated, perhaps deliberately, to some point on Thursday. The lack of logging information and the routine discarding of what old logs did exist hampered investigations.)

Mike Muuss of BRL reported at the NCSC meeting that RAND was also hit at 9:00pm or soon thereafter; Steve Miller of the University of Maryland (UMD) reports the UMD was first hit at 10:54pm; Phil Lapsley of the University of California, Berkeley (UCB) stated that Berkeley was hit at 11:00pm.

3.2 Thursday Morning: "This isn't April First"

3.2.1 More People Notice the Virus

Dave Edwards, of SRI International, said at the NCSC meeting that SRI was hit at midnight. Chuck Cole and Russell Brand of the Lawrence Livermore National Laboratory (LLNL) reported that they were assembling their response team by 2:00am, and John Bruner independently reported spotting the virus on the S1 machines at LLNL about that time.

Pascal Chesnais of the MIT Media Lab was one of the first people at MIT to spot the virus, after 10:00pm Wednesday, but assumed it was just ''a local runaway program''. A group at the Media lab killed the anomalous shell and compilers processes, and all seemed normal. After going for an dinner and ice cream, they figured out that it was a virus and it was coming in via mail. Their response was to shut down network services such as mail and to isolate themselves from the campus network. The MIT Telecommunications Network Group's monitoring information shows the Media Lab gateway first went down at 11:40pm Wednesday, but was back up by 3:00am. At 3:10am Pascal gave the first notice of the virus at MIT, by creating a message of the day on *media-lab* (see Figure 1).

3.2.2 False Alarms or Testing?

Pascal later reported that logs on *media-lab* show several scattered messages, ''ttloop: peer died: No such file or directory'', which frequently occurred just before the virus

[6] Cornell systems personel had discovered unusual messages in their mailer logs and passed the logs to Berkeley which passed them to the NCSC. Later it was reported that the alleged author of the virus was a Cornell graduate student[19].

```
A Virus has been detected on media-lab, we suspect that whole internet is
infected by now.  The virus is spread via mail of all things... So Mail
outside of media-lab will NOT be accepted.  Mail addressed to foreign
hosts will NOT be delivered.  This situation will continue until someone
figures out a way of killing the virus and telling everyone how to
do it without using email...

--- lacsap Nov 3 1988 03:10am
```

Figure 1: Thursday morning's message of the day on *media-lab.mit.edu*.

attacked (see section A.5.2). There were a few every couple of days, several during Wednesday afternoon and many starting at 9:48pm. The logs on *media-lab* start on 25 October 1988 and entries were made by `telnetd` on the following dates before the swarm on Wednesday night: Oct 26 15:01:57, Oct 28 11:26:55, Oct 28 17:36:51, Oct 31 16:24:41, Nov 1 16:08:24, Nov 1 18:02:43, Nov 1 18:58:30, Nov 2 12:23:51, and Nov 2 15:21:47.

It is not clear whether these represent early testing of the virus, or if they were just truly accidental premature closings of telnet connections. We assume the latter. With hindsight we can say a `telnetd` that logged its peer address, even for such error messages, would have been quite useful in tracing the origin and progress of the virus.

3.2.3 E-mail warnings

The first posting mentioning the virus was by Peter Yee of NASA Ames at 2:28am on Wednesday to the *tcp-ip@sri-nic.arpa* mailing list. Peter stated that UCB, UCSD, LLNL, Stanford, and NASA Ames had been attacked, and described the use of sendmail to pull over the virus binaries, including the `x*` files which the virus briefly stored in `/usr/tmp`. The virus was observed sending VAX and Sun binaries, having DES tables built in, and making some use of `.rhosts` and `hosts.equiv` files. A phone number at Berkeley was given and Phil Lapsley and Kurt Pires were listed as being knowledgeable about the virus.

At 3:34am Andy Sudduth from Harvard made his anonymous posting[7] to *tcp-ip@sri-nic.arpa* [8] The posting said that a virus might be loose on the Internet and that there were three steps to take to prevent further transmission. These included not running `fingerd` or fixing it not to overwrite the stack when reading its arguments from the net[9], being sure `sendmail` was compiled without the `debug` command, and not running `rexecd`.

Mike Patton, Network Manager for the MIT Laboratory for Computer Science (LCS), was the first to point out to us the peculiarities of this posting. It was made from an Annex terminal server[10] at Aiken Laboratory at Harvard, by telneting to the SMTP port of *iris.brown.edu*. This is obvious since the message was from "foo%bar.arpa" and because the last line of the message was "qui\177\177\177", an attempt to get rubout processing out of the Brown SMTP server, a common mistake when faking Internet mail.

It was ironic that this posting did almost no good. Figure 2 shows the path it took to get to Athena. There was a 20 hour delay before the message escaped from *relay.cs.net*[11] and got to *sri-nic.arpa*. Another 6 hours went by before the message was received by *athena.mit.edu* [12]. Other sites have reported similar delays.

3.2.4 Yet More People Notice the Virus

About 4:00am Thursday Richard Basch of MIT Project Athena noticed a "text table full" `syslog` message from *paris.mit.edu*, an Athena development machine. Since there was only one message and he was busy doing a project for a digital design lab course, he ignored it.

At 4:51am Chris Hanson of the MIT AI Laboratory reported spotting anomalous telnet traffic to serveral gateways coming from machines at LCS. He noted that the attempts were occurring every one or two seconds and had been happening for several hours.

At 5:58am Thursday morning Keith Bostic of Berkeley made the first bug fix posting. The message went

[7] In a message to the same mailing list on Saturday 5 November 1988, he acknowledged being the author of the Thursday morning message and stated he had posted the message anonymously because "at the time I didn't want to answer questions about how I knew."

[8] An "obscure electronic bulletin board", according to the New York Times[11]. Nothing could be further from the truth.

[9] This was a level of detail that only the originator of the virus could have known at that time. To our knowledge nobody had yet identified the finger bug, since it only affected certain VAX hosts, and certainly nobody had discovered its mechanism.

[10] Perhaps ironically named *influenza.harvard.edu*.

[11] This is probably because *relay.cs.net* was off the air during most of the crisis.

[12] Phil Lapsley and Mike Karels of Berkeley reported that the only way to get mail to *tcp-ip@sri-nic.arpa* to flow quickly is to call up Mark Lottor at SRI and ask him to manually push the queue through.

```
Received: by ATHENA.MIT.EDU (5.45/4.7) id AA29119; Sat, 5 Nov 88 05:59:13 EST
Received: from RELAY.CS.NET by SRI-NIC.ARPA with TCP; Fri, 4 Nov 88 23:23:24 PST
Received: from cs.brown.edu by RELAY.CS.NET id aa05627; 3 Nov 88 3:47 EST
Received: from iris.brown.edu (iris.ARPA) by cs.brown.edu (1.2/1.00)
          id AA12595; Thu, 3 Nov 88 03:47:19 est
Received: from  (128.103.1.92) with SMTP via tcp/ip
          by iris.brown.edu on Thu, 3 Nov 88 03:34:46 EST
```

Figure 2: Path of Andy Sudduth's warning message from Harvard to MIT.

to the *tcp-ip@sri-nic.arpa* mailing list and the newsgroups *comp.bugs.4bsd.ucb-fixes*, *news.announce*, and *news.sysadmin*. It supplied the ‘‘compile without the debug command’’ fix to `sendmail` (or patch the `debug` command to a garbage string), as well as the very wise suggestion to rename the UNIX C compiler and loader (`cc` and `ld`), which was effective since the virus needed to compile and link itself, and which would be effective at protecting against non-`sendmail` attacks, whatever those might have turned out to be. It also told people that the virus renamed itself to ‘‘(sh)’’ and used temporary files in `/usr/tmp` named XNNN,vax.o, XNNN,sun3.o, and XNNN,l1.c (where NNN were random numbers, possibly process id's), and suggested that you could identify infected machine by looking for these files. That was somewhat difficult to do in practice, however, since the virus quickly got rid of all of these files. A somewhat better solution was proposed later in the day by, among others, John Kohl of DEC and Project Athena, who suggested doing a `cat -v /usr/tmp`, thus revealing the raw contents of the directory, including the names of deleted files whose directory slots had not yet been re-used[13].

The `fingerd` attack was not even known, much less understood, at this point. Phil Lapsley reported at the NCSC meeting that Ed Wang of Berkeley discovered the `fingerd` mechanism around 8:00am and sent mail to Mike Karels, but this mail went unread until after the crisis had passed.

At 8:06am Gene Spafford of Purdue forwarded to the *nntp-managers@ucbvax.berkeley.edu* mailing list Keith Bostic's fixes. Ted Ts'o of MIT Project Athena forwarded this to an internal Project Athena hackers list (*watchmakers@athena.mit.edu*) at 10:07am. He expressed disbelief (‘‘no, it's not April 1st’’), and thought Athena machines were safe. Though no production Athena servers were infected, several private workstations and development machines were, so this proved overly optimistic.

Mark Reinhold, a MIT LCS graduate student, reacted to the virus around 8:00am by powering off some network equipment in LCS. Tim Shepard, also a LCS graduate student, soon joined him. They were hampered by a growing number of people who wanted information about what was happening. Mark and Tim tried to call Peter Yee several times and eventually managed to get through to Phil Lapsley who relayed what was then known about the virus.

At about this time, Richard Basch returned to his workstation (you can only do so much school-work after all) and noticed many duplicates of the ‘‘text table full’’ messages from *paris* and went to investigate. He discovered several suspicious logins from old accounts which should have long ago been purged. The load was intolerably high, and he only managed to get one line out of a `netstat` command before giving up, but that proved quite interesting. It showed an outgoing `rsh` connection from *paris* to *fmgc.mit.edu*, which is a standalone non-UNIX gateway.

During Thursday morning Ray Hirschfeld spotted the virus on the MIT Math department Sun workstations and shut down the math gateway to the MIT backbone at 10:15am. It remained down until 3:15pm.

Around 11:00am the MIT Statistics Center called Dan Geer, Manager of System Development at Project Athena. One of their Sun workstations, *dolphin.mit.edu* had been infected via a Project Athena guest account with a weak password, along with the account of a former staff member. This infection had spread to all hosts in the Statistics Center. They had been trying for some time prior to call Dan to eradicate the virus, but the continual reinfection among their local hosts had proved insurmountably baffling.

Keith Bostic sent a second virus fix message to *comp.4bsd.ucb-fixes* at 11:12am. It suggested using 0xff instead of 0x00 in the binary patch to `sendmail`. The previous patch, while effective against the current virus, would drop you into debug mode if you sent an empty command line. He also suggested using the UNIX `strings` command to look in the `sendmail` binary for the string ‘‘debug’’. If it didn't appear at all then your version of `sendmail` was safe.

About 11:30am Pascal Chesnais requested that the Network Group isolate the Media Lab building and it remained

[13] Jerry Saltzer, MIT EECS Professor and Technical Director of Project Athena, included similar detection advice in a message describing the virus to the Athena staff sent at 11:17am on Friday.

so isolated until Friday at 2:30pm.

Russ Mundy of the Defense Communications Agency reported at the NCSC meeting that the MILNET to ARPANET mailbridges were shut down at 11:30am and remained down until Friday at 11:00am.

In response to complaints from non-UNIX users, Mark Reinhold and Stan Zanarotti, another LCS graduate student, turned on the repeaters at LCS which had been previously powered down and physically disconnected UNIX machines from the network around 11:15am. Tim Shepard reloaded a root partition of one machine from tape (to start with known software), and added a feature to `find`, a UNIX file system scanner, to report low-level modification times. Working with Jim Fulton of the X Consortium, Tim inspected *allspice.lcs.mit.edu*; by 1:00pm, they had verified that the virus had not modified any files on *allspice* and had installed a recompiled `sendmail`.

3.3 Thursday Afternoon: "This is Bad News"

3.3.1 Word Spreads

By the time Jon Rochlis of the MIT Telecommunications Network Group arrived for work around noon on Thursday 3 November 1988, the Network Group had received messages from MIT Lincoln Laboratory saying they had ''been brought to their knees'' by the virus, from Sergio Heker of the John Von Neumann National Supercomputer Center warning of network problems, and from Kent England of Boston University saying they had cut their external links. The MIT Network Group loathed the thought of severing MIT's external connections and never did throughout the crisis.

At 1:30pm Dan Geer and Jeff Schiller, Manager of the MIT Network and Project Athena Operations Manager, returned to the MIT Statistics Center and were able to get both VAX and Sun binaries from infected machines.

Gene Spafford posted a message at 2:50pm Thursday to a large number of people and mailing lists including *nntp-managers@ucbvax.berkeley.edu*, which is how we saw it quickly at MIT. It warned that the virus used `rsh` and looked in `hosts.equiv` and `.rhosts` for more hosts to attack.

Around this time the MIT group in E40 (Project Athena and the Telecommunications Network Group) called Milo Medin of NASA and found out much of the above. Many of us had not yet seen the messages. He pointed out that the virus just loved to attack gateways, which were found via the routing tables, and remarked that it must have not been effective at MIT where we run our own C Gateway code on our routers, not UNIX. Milo also said that it seemed to randomly attack network services, swamping them with input. Some daemons that ran on non-standard ports had logged such abnormal input. At the time we thought the virus might be systematically attacking all possible network services exploiting some unknown common flaw. This was not true but it seemed scary at the time. Milo also informed us that DCA had shut down the mailbridges which serve as gateways between the MILNET and the ARPANET. He pointed us to the group at Berkeley and Peter Yee specifically.

3.3.2 It uses finger

At about 6:00pm on Thursday, Ron Hoffmann, of the MIT Telecommunications Network Group, observed the virus attempting to log into a standalone router using the Berkeley remote login protocol; the remote login attempt originated from a machine previously believed immune since it was running a mailer with the `debug` command turned off. The virus was running under the user name of `nobody`, and it appeared that it had to be attacking through the finger service, the only network service running under that user name. At that point, we called the group working at Berkeley; they confirmed our suspicions that the virus was spreading through `fingerd`.

On the surface, it seemed that `fingerd` was too simple to have a protection bug similar to the one in `sendmail`; it was a very short program, and the only program it invoked (using the UNIX *exec* system call) was named using a constant pathname. A check of the modification dates of both `/etc/fingerd` and `/usr/ucb/finger` showed that both had been untouched, and both were identical to known good copies located on a read-only filesystem.

Berkeley reported that the attack on finger involved ''shoving some garbage at it'', probably control A's; clearly an overrun buffer wound up corrupting something.

Bill Sommerfeld of Apollo Computer and MIT Project Athena guessed that this bug might involve overwriting the saved program counter in the stack frame; when he looked at the source for `fingerd`, he found that the buffer it was using was located on the stack; in addition, the program used the C library *gets* function, which assumes that the buffer it is given is long enough for the line it is about to read. To verify that this was a viable attack, he then went on to write a program which exploited this hole in a benign way. The test virus sent the string ''Bozo!'' back out the network connection.

Miek Rowan and Mike Spitzer also report having discovered the `fingerd` mechanism at about the same time and forwarded their discovery to Gene Spafford and Keith Bostic, but in the heat of the moment the discovery went unrecognized. Liudvikas Bukys of the University of Rochester posted to the *comp.bugs.4bsd* newsgroup a detailed description of the `fingerd` mechanism at 7:21pm. The message also stated that the virus used telnet but perhaps that was only after cracking passwords. In reality it only sometimes used telnet to "qualify" a machine for later attack, and only used `rsh` and `rexec` to take advantage of passwords it had guessed.

A *risks@kl.sri.com* digest[20] came out at 6:52pm. It included a message from Cliff Stoll which described the spread of the virus on MILNET and suggested that MILNET sites might want to remove themselves from the network. Cliff concluded by saying, "This is bad news." Other messages were from Gene Spafford, Peter Neumann of SRI, and Matt Bishop of Dartmouth. They described the `sendmail` propagation mechanism.

3.4 Thursday Evening: "With Microscope and Tweezers"

3.4.1 Getting Down To Work

In the office of the Student Information Processing Board (SIPB), Stan Zanarotti and Ted Ts'o had managed to get a VAX binary and core dump from the virus while it was running on a machine at LCS.

Stan and Ted started attacking the virus. Pretty soon they had figured out the xor encoding of the text strings embedded in the program and were manually decoding them. By 9:00pm Ted had written a program to decode all the strings and we had the list of strings used by the program, except for the built-in dictionary which was encoded in a different fashion (by turning on the high order bit of each character).

At the same time they discovered the ip address of *ernie.berkeley.edu*, 128.32.137.13, in the program; they proceeded to take apart the virus routine *send_message* to figure out what it was sending to *ernie*, how often, and if a handshake was involved. Stan told Jon Rochlis in the MIT Network Group of the SIPB group's progress. The people in E40 called Berkeley and reported the finding of *ernie*'s address. Nobody seemed to have any idea why that was there.

At 9:20pm Gene Spafford created the mailing list *phage@purdue.edu*. It included all the people he had been mailing virus information to since the morning; more people were to be added during the next few days. This list proved invaluable, since it seemed to have many of the "right" people on it and seemed to work in near real time despite all the network outages.

At 10:18pm Keith Bostic made his third bug fix posting. It included new source code for `fingerd` which used *fgets* instead of *gets* and did an *exit* instead of *return*. He also included a more general `sendmail` patch which disabled the `debug` command completely.

3.4.2 The Media Descends

About this time a camera crew from WNEV-TV Channel 7 (the Boston CBS affiliate) showed up at the office of James D. Bruce, MIT EECS Professor and Vice President for Information Systems. He called Jeff Schiller and headed over to E40. They were both were interviewed and stated that there were 60,000 Internet hosts[14], along with an estimate of 10% infection rate for the 2,000 hosts at MIT. The infection rate was a pure guess, but seemed reasonable at the time. These numbers were to stick in a way we never anticipated. Some of the press reports were careful to explain the derivation of the numbers they quoted, including how one could extrapolate that as many as 6,000 computers were infected, but many reports were not that good and simply stated things like "at least 6,000 machines had been hit."

We were unable to show the TV crew anything "visual" caused by the virus, something which eventually became a common media request and disappointment. Instead they settled for people looking at workstations talking "computer talk."

The virus was the lead story on the 11:00pm news and was mentioned on National Public Radio as well. We were quite surprised that the real world would pay so much attention. Sound bites were heard on the 2:00am CBS Radio News, and footage shot that evening was shown on the CBS morning news (but by that point we were too busy to watch).

After watching the story on the 11:00pm news we realized it was time to get serious about figuring out the detailed workings of the virus. We all agreed that decompiling was the route to take, though later we also mounted an effort to infect a specially instrumented machine to see the virus in operation. As Jerry Saltzer said in a later message to the Project Athena staff, we undertook a "wizard-level analysis" by going over the virus "with microscope and tweezers."

3.5 Friday: "Where's Sigourney Weaver?"

3.5.1 Decompiling in Earnest

Tim Shepard joined the group in E40, just before midnight on Thursday. We thought we saw packets going to *ernie* and replies coming back, though this later proved to be an illusion. Tim had hundreds of megabytes of packet headers gathered Thursday morning from a subnet at LCS which was known to have had infected machines on it. Unfortunately the data was sitting on a machine at LCS, which was still off the network, so Tim decided to go back and look through his data. Within an hour or two, Tim called back to say that he found no unusual traffic to *ernie* at all. This was our first good confirmation that the *ernie* packets were

[14] This was based on Mark Lottor's presentation to the October 1988 meeting of the Internet Engineering Task Force.

a red-herring or at least that they didn't actually wind up being sent.

Serious decompiling began after midnight. Stan and Ted soon left the SIPB office and joined the group working in E40, bringing with them the decoding of the strings and much of the decompiled main module for the virus. Mark Eichin, who had recently spent a lot of time disassembling-assembling some ROMs and thus had recent experience at reverse engineering binaries, took the lead in dividing the project up and assigning parts to people. He had also woke up in late afternoon and was the most prepared for the long night ahead.

At 1:55am Mark discovered the first of the bugs in the virus. A *bzero* call in *if_init* was botched. At 2:04am Stan had a version of the main module that compiled. We called Keith Bostic at Berkeley at 2:20am and arranged to do FTP exchanges of source code on an MIT machine (both Berkeley and MIT had never cut their outside network connections). Unfortunately, Keith was unable to get the hackers at Berkeley to take a break and batch up their work, so no exchange happened at that time.

At 2:45am Mark started working on *checkother*[15] since the Berkeley folks were puzzled by it. Jon Rochlis was working on the later *cracksome* routines. By 3:06am Ted had figured out that *ha* built a table of target hosts which had telnet listeners running. By 3:17am Ted and Hal Birkeland from the Media Lab had determined that the *crypt* routine was the same as one found in the C library. Nobody had yet offered a reason why it was included in the virus, rather than being picked up at link time[16]. Mark had finished *checkother* and Ted had finished *permute* at 3:28am. We worked on other routines throughout the morning.

3.5.2 Observations from Running the Virus

The first method of understanding the virus was the decompilation effort. A second method was to watch the virus as it ran, in an attempt to characterize what it was doing – this is akin to looking at the symptoms of a biological virus, rather than analyzing the DNA of the virus.

We wanted to do several things to prepare for observing the virus:

- Monitoring. We wanted to set up a machine with special logging, mostly including packet monitors.
- Pointers. We wanted to ''prime'' the machine with pointers to other machines so we could watch how the virus would attack its targets. By placing the names of the target machines in many different places on the ''host'' computer we could also see how the virus created its lists of targets.
- Isolation. We considered isolating the machines involved from the network totally (for paranoia's sake) or by a link-layer bridge to cut down on the amount of extraneous traffic monitored. True isolation proved more than we were willing to deal with at the time, since all of our UNIX workstations assume access to many network services such as nameservers and file servers. We didn't want to take the time to build a functional standalone system, though that would have been feasible if we had judged the risk of infecting other machines too great.

Mike Muuss reported that the BRL group focused on monitoring the virus in action. They prepared a special logging kernel, but even in coordination with Berkeley were unable to re-infect the machine in question until Saturday.

By 1:00am Friday we had set up the monitoring equipment (an IBM PC running a packet monitor) and two workstations (one acting as the target, the other running a packet monitoring program and saving the packet traces to disk), all separated from the network by a link-layer bridge and had dubbed the whole setup the ''virus net''. We, too, were unsuccessful in our attempt to get our target machine infected until we had enough of the virus decompiled to understand what arguments it wanted. By 3:40am John Kohl had the virus running on our "virus net" and we learned a lot by watching what it did. The virus was soon observed trying telnet, SMTP, and finger connections to all gateways listed in the routing table. Later it was seen trying `rsh` and `rexec` into one of the gateways.

At 4:22am, upon hearing of the virus going after yet another host in a ''new'' manner, Jon Rochlis remarked ''This really feels like the movie *Aliens*. So where is Sigourney Weaver?'' Seeing the virus reach out to infect other machines seemed quite scary and beyond our control.

At 5:45am we called the folks at Berkeley and finally exchanged code. A number of people in Berkeley had punted to get some sleep, and we had a bit of difficulty convincing the person who answered Keith Bostic's phone that we weren't the bad guy trying to fool them. We gave him a number at MIT that showed up in the NIC's whois database, but he never bothered to call back.

At this point a bunch of us went out and brought back some breakfast.

3.5.3 The Media Really Arrives

We had been very fortunate that the press did not distract us, and that we were thus able to put most of our time into our decompilation and analysis efforts. Jim Bruce and the

[15] This and all the other routines mentioned here are described in detail in Appendix A. The routines mentioned here are not intended to be an exhaustive list of the routines we worked on.

[16] It turned out that we were wrong and the version of *crypt* was *not* the same as library version[9]. Not everything you do at 3:00am turns out to be right.

MIT News Office did a first rate job of dealing with most of the press onslaught. By early morning Friday there was so much media interest that MIT News Office scheduled a press conference for noon in the Project Athena Visitor Center in E40.

Just before the press conference, we briefed Jim on our findings and what we thought was important: the virus didn't destroy or even try to destroy any data; it did not appear to be an ''accident;'' we understood enough of the virus to speak with authority; many people (especially the people we had talked to at Berkeley) had helped to solve this.

We were amazed at the size of the press conference – there were approximately ten TV camera crews and twenty-five reporters. Jeff Schiller spent a good amount of time talking to reporters before the conference proper began, and many got shots of Jeff pointing at the letters ''(sh)'' on the output of a `ps` command. Jim and Jeff answered questions as the decompiling crew watched from a vantage point in the back of the room. At one point a reporter asked Jeff how many people had enough knowledge to write such a virus and in particular, if Jeff could have written such a program. The answer was of course many people could have written it and yes, Jeff was one of them. The obvious question was then asked: ''Where were you on Wednesday night, Jeff?'' This was received with a great deal of laughter. But when a reporter stated that sources at the Pentagon had said that the instigator of the virus had come forward and was a BU or MIT graduate student, we all gasped and hoped it hadn't really been one of our students.

After the conference the press filmed many of us working (or pretending to work) in front of computers, as well as short interviews.

The media was uniformly disappointed that the virus did nothing even remotely visual. Several also seemed pained that we weren't moments away from World War III, or that there weren't large numbers of companies and banks hooked up to ''MIT's network'' who were going to be really upset when Monday rolled around. But the vast majority of the press seemed to be asking honest questions in an attempt to grapple with the unfamiliar concepts of computers and networks. At the NCSC meeting Mike Muuss said, ''My greatest fear was that of seeing a *National Enquirer* headline: Computer Virus Escapes to Humans, 96 Killed.'' We were lucky that didn't happen.

Perhaps the funniest thing done by the press was the picture of the virus code printed in Saturday's edition of the *Boston Herald*[21]. Jon Kamens of MIT Project Athena had made a window dump of the assembly code for the start of the virus (along with corresponding decompiled C code), even including the window dump command itself. The truly amusing thing was that the *Herald* had gotten an artist to add tractor feed holes to the printout in an attempt to make it look like something that a computer might have generated. We're sure they would have preferred a dot matrix printer to the laser printer we used.

Keith Bostic called in the middle of the press zoo, but we were too busy dealing with the press, so we cut the conversation short. He called us back around 3:00pm and asked for our affiliations for his next posting [17]. Keith also asked if we liked the idea of posting bug fixes to the virus itself, and we instantly agreed with glee. Keith made his fourth bug fix posting at 5:04pm, this time with fixes to the virus. Again he recommended renaming *ld*, the UNIX linker.

Things began to wind down after that, though the press was still calling and we managed to put off the NBC *Today* show until Saturday afternoon. Most of us got a good amount of sleep for the first time in several days.

3.6 Saturday: Source Code Policy

Saturday afternoon, 5 November 1988, the *Today* show came to the SIPB Office, which they referred to as the ''computer support club'' (*sic*), to find a group of hackers. They interviewed Mark Eichin and Jon Rochlis and used Mark's description of what hackers really try to do on Monday morning's show.

After the *Today* show crew left, many of us caught up on our mail. It was then that we first saw Andy Sudduth's Thursday morning posting to *tcp-ip@sri-nic.arpa* and Mike Patton stopped by and pointed out how strange it was.

We soon found ourselves in the middle of a heated discussion on *phage@purdue.edu* regarding distribution of the decompiled virus source code. Since we had received several private requests for our work, we sat back and talked about what to do, and quickly reached consensus. We agreed with most of the other groups around the country who had come to the decision not to release the source code they had reverse engineered. We felt strongly that the details of the inner workings of the virus should *not* be kept hidden, but that actual source code was a different matter. We (and others) intended to write about the algorithms used by the virus so that people would learn what the Internet community was up against. This meant that somebody could use those algorithms to write a new virus; but the knowledge required to do so is much greater than what is necessary to recompile the source code with a new, destructive line or two in it. The energy barrier for this is simply too low. The people on our team (not the MIT administration) decided to keep our source private until things calmed down; then we would consider to whom to distribute the program. A public posting of the MIT code was not going to happen.

Jerry Saltzer, among others, has argued forcefully that the code itself should be publicly released at some point in

[17] He almost got them right, except that he turned the Laboratory for Computer Science into the Laboratory for Computer Services.

the future. After sites have had enough time to fix the holes with vendor supplied bug fixes, we might do so.

3.7 Tuesday: The NCSC Meeting

On Tuesday 8 November 1988 Mark Eichin and Jon Rochlis attended the Baltimore post-mortem meeting hosted by the NCSC. We heard about the meeting indirectly at 2:00am and flew to Baltimore at 7:00am. Figuring there was no time to waste with silly things like sleep, we worked on drafts of this document. The meeting will be described in more detail by the NCSC, but we will present a very brief summary here.

Attending the meeting were representatives of the National Institute of Science and Technology (NIST), formerly the National Bureau of Standards, the Defense Communications Agency (DCA) , the Defense Advanced Research Projects Agency (DARPA), the Department of Energy (DOE), the Ballistics Research Laboratory (BRL), the Lawrence Livermore National Laboratory (LLNL), the Central Intelligence Agency (CIA), the University of California Berkeley (UCB), the Massachusetts Institute of Technology (MIT), Harvard University, SRI International, the Federal Bureau of Investigation (FBI), and of course the National Computer Security Center (NCSC). This is not a complete list. The lack of any vendor participation was notable.

Three-quarters of the day was spent discussing what had happened from the different perspectives of those attending. This included chronologies, actions taken, and an analysis of the detailed workings of the virus; Meanwhile our very rough draft was duplicated and handed out.

The remaining time was spent discussing what we learned from the attack and what should be done to prepare for future attacks. This was much harder and it is not clear that feasible solutions emerged, though there was much agreement on several motherhood and apple-pie suggestions. By this we mean the recommendations sound good and and by themselves are not objectionable, but we doubt they will be effective.

3.8 Wednesday-Friday: The Purdue Incident

On Wednesday evening 9 November 1988, Rich Kulawiec of Purdue posted to *phage@purdue.edu* that he was making available the `unas` disassembler that he (and others at Purdue) used to disassemble the virus. He also made available the output of running the virus through this program. Rumor spread and soon the NCSC called several people at Purdue, including Gene Spafford, in an attempt to get this copy of the virus removed. Eventually the President of Purdue was called and the file was deleted. The New York Times ran a heavily slanted story about the incident on Friday 11 November 1988[22].

Several mistakes were made here. First the NCSC was concerned about the wrong thing. The disassembled virus was not important and was trivial for any infected site to generate. It simply was not anywhere near as important as the decompiled virus, which could have very easily have been compiled and run. When the MIT group was indirectly informed about this and discovered exactly what was publicly available, we wondered what the big deal was. Secondly, the NCSC acted in a strong-handed manner that upset the people at Purdue who got pushed around.

Other similar incidents occurred around the same time. Jean Diaz of the MIT SIPB, fowarded a partially decompiled copy of the virus[18] to *phage@purdue.edu* at some time on Friday 4 November 1988, but it spent several days in a mail queue on *hplabs.hp.com* before surfacing. Thus it had been posted before any of the discussion of source code release had occurred. It also was very incomplete and thus posed little danger since the effort required to turn it into a working virus was akin to the effort required to write the virus from scratch.

These two incidents, however, caused the press to think that a second outbreak of the virus had once again brought the network to its knees. Robert French, of the MIT SIPB and Project Athena, took one such call on Thursday 10 November and informed the reporter that no such outbreak had occurred. Apparently rumors of source code availability (the Purdue incident and Jean's posting) led to the erroneous conclusion that enough information of some sort had been let out and damage had been done. Rumor control was once again shown to be important.

4 Lessons and Open Issues

The virus incident taught many important lessons. It also brought up many more difficult issues which need to be addressed in the future .

4.1 Lessons from the Community's Reactions

The chronology of events is interesting. The manner in which the Internet community reacted to the virus attack points out areas of concern or at least issues for future study.

- Connectivity was important. Sites which disconnected from the network at the first sign of trouble hurt themselves and the community. Not only could they not report their experiences and findings, but they couldn't get timely bug fixes. Furthermore, other sites using

[18] This was the work of Don Becker of Harris Corporation.

them as mail relays were crippled, thus delaying delivery of important mail, such as Andy Sudduth's Thursday morning posting, until after the crisis had passed. Sites like MIT and Berkeley were able to collaborate in a meaningful manner because they never took themselves off the network.

- The "old boy network" worked. People called and sent electronic mail to the people they knew and trusted and much good communication happened. This can't be formalized but it did function quite well in the face of the crisis.

- Late night authentication is an interesting problem. How did you know that it really is MIT on the phone? How did you know that Keith Bostic's patch to `sendmail` is really a fix and isn't introducing a new problem? Did Keith really send the fix or was it his evil twin, Skippy?

- Whom do you call? If you need to talk to the manager of the Ohio State University network at 3:00am whom do you call? How many people can find that information, and is the information up to date?

- Speaker phones and conference calling proved very useful.

- How groups formed and who led them is a fascinating topic for future study. Don Alvarez of the MIT Center for Space Research presented his observations on this at the NCSC meeting.

- Misinformation and illusions ran rampant. Mike Muuss categorized several of these at the NCSC meeting. Our spotting of a handshake with *ernie* is but one example.

- Tools were not as important as one would have expectd. Most of decompiling work was done manually with no more tools than a disassembler (`adb`) and an architecture manual. Based on its experience with PC viruses, the NCSC feels that more sophisticated tools must be developed. While this may be true for future attacks, it was not the case for this attack.

- Source availability was important. All of the sites which responded quickly and made progress in truly understanding the virus had UNIX source code.

- The academic sites performed best. Government and commercial sites lagged behind places like Berkeley and MIT in figuring out what was going on and creating solutions.

- Managing the press was critical. We were not distracted by the press and were able to be quite productive. The MIT News office did a fine job keeping the press informed and out of the way. Batching the numerous requests into one press conference helped tremendously. The Berkeley group, among others, reported that it was difficult to get work done with the press constantly hounding them.

4.2 General Points for the Future

More general issues have popped to the surface because of the virus. These include the following:

- Least Privilege. This basic security principle is frequently ignored and this can result in disaster.

- "We have met the enemy and he is us." The alleged author of the virus has made contributions to the computer security field and was by any definition an insider; the attack did not come from an outside source who obtained sensitive information, and restricting information such as source code would not have helped prevent this incident.

- Diversity is good. Though the virus picked on the most widespread operating system used on the Internet and on the two most popular machine types, most of the machines on the network were never in danger. A wider variety of implementations is probably good, not bad. There is a direct analogy with biological genetic diversity to be made.

- "The cure shouldn't be worse than the disease." Chuck Cole made this point and Cliff Stoll also argued that it may be more expensive to prevent such attacks than it is to clean up after them. Backups are good. It may be cheaper to restore from backups than to try to figure out what damage an attacker has done[6].

- Defenses *must* be at the host level, not the network level. Mike Muuss and Cliff Stoll have made this point quite eloquently[6]. The network performed its function perfectly and should not be faulted; the tragic flaws were in several application programs. Attempts to fix the network are misguided. Jeff Schiller likes to use an analogy with the highway system: anybody can drive up to your house and probably break into your home, but that does not mean we should close down the roads or put armed guards on the exit ramps.

- Logging information is important. The `inetd` and `telnetd` interaction logging the source of virus attacks turned out to be a lucky break, but even so many sites did not have enough logging information available to identify the source or times of infection. This greatly hindered the responses, since people frequently had to install new programs which logged more information. On the other hand, logging information tends to accumulate quickly and is rarely referenced. Thus it is frequently automatically purged. If we log helpful information, but find it is quickly purged, we have not improved the situtation much at all. Mike Muuss-points out that frequently one can retrieve such information from backups[6], but this is not always true.

- Denial of service attacks are easy. The Internet is amazingly vulnerable to such attacks. These attacks are quite difficult to prevent, but we could be much better prepared to identify their sources than we are today. For example, currently it is not hard to imagine writing a program or set of programs which crash two-thirds of the existing Sun Workstations or other machines implementing Sun's Network Filesystem (NFS). This is serious since such machines are the most common computers connected to the Internet. Also, the total lack of authentication and authorization for network level routing makes it possible for an ordinary user to disrupt communications for a large portion of the Internet. Both tasks could be easily done in a manner which makes tracking down the initiator extremely difficult, if not impossible.

- A central security fix repository may be a good idea. Vendors *must* participate. End users, who likely only want to get their work done, must be educated about the importance of installing security fixes.

- Knee-jerk reactions should be avoided. Openness and free flow of information is the whole point of networking, and funding agencies should not be encouraged to do anything damaging to this without very careful consideration. Network connectivity proved its worth as an aid to collaboration by playing an invaluable role in the defense and analysis efforts during the crisis, despite the sites which isolated themselves.

5 Acknowledgments

Many people contributed to our effort to take apart the virus. We would like to thank them all for their help and insights both during the immediate crisis and afterwards.

5.1 The MIT team

The MIT group effort encompassed many organizations within the Institute. It included people from Project Athena, the Telecommunications Network Group, the Student Information Processing Board (SIPB), the Laboratory for Computer Science, and the Media Laboratory.

The SIPB's role is quite interesting. It is a volunteer student organization that represents students on issues of the MIT computing environment, does software development, provides consulting to the community, and other miscellaneous tasks. Almost all the members of the MIT team which took apart the virus were members of the SIPB, and the SIPB office was the focus for early efforts at virus catching until people gathered in the Project Athena offices.

Mark W. Eichin (Athena and SIPB) and Stanley R. Zanarotti (LCS and SIPB) led the team disassembling the virus code. The team included Bill Sommerfeld (Athena/Apollo Computer and SIPB), Ted Y. Ts'o (Athena and SIPB), Jon Rochlis (Telecommunications Network Group and SIPB), Ken Raeburn (Athena and SIPB), Hal Birkeland (Media Laboratory), and John T. Kohl (Athena/DEC and SIPB).

Jeffrey I. Schiller (Campus Network Manager, Athena Operations Manager, and SIPB) did a lot of work in trapping the virus, setting up an isolated test suite, and dealing with the media. Pascal Chesnais (Media Laboratory) was one of the first at MIT to spot the virus. Ron Hoffmann (Network Group) was one of the first to notice an MIT machine attacked by finger.

Tim Shepard (LCS) provided information about the propagation of the virus, as well as large amounts of ''netwatch'' data and other technical help.

James D. Bruce (EECS Professor and Vice President for Information Systems) and the MIT News Office did an admirable job of keeping the media manageable and letting us get our work done.

5.2 The Berkeley Team

We communicated and exchanged code with Berkeley extensively throughout the morning of 4 November 1988. The team there included Keith Bostic (Computer Systems Research Group, University of California, Berkeley), Mike Karels (Computer Systems Research Group, University of California, Berkeley), Phil Lapsley (Experimental Computing Facility, University of California, Berkeley), Dave Pare (FX Development, Inc.), Donn Seeley (University of Utah), Chris Torek (University of Maryland), and Peter Yee (Experimental Computing Facility, University of California, Berkeley).

5.3 Others

Numerous others across the country deserve thanks; many of them worked directly or indirectly on the virus, and helped coordinate the spread of information. Special thanks should go to Gene Spafford (Purdue) for serving as a central information point and providing key insight into the workings of the virus. Don Becker (Harris Corporation) has provided the most readable decompilation of the virus which we have seen to date. It was most helpful.

An attempt was made to provide a review copy of this paper to all people mentioned by name. Most read a copy and many provided useful corrections.

People who offered particularly valuable advice included Judith Provost, Jennifer Steiner, Mary Vogt, Stan Zanarotti, Jon Kamens, Marc Horowitz, Jenifer Tidwell, James Bruce, Jerry Saltzer, Steve Dyer, Ron Hoffmann and many unnamed people from the SIPB Office. Any remaining flaws in this paper are our fault, not theirs.

Special thanks to Bill Sommerfeld for providing the description of the finger attack and its discovery.

A The Program

This Appendix describes the virus program subroutine by subroutine. For reference, the flow of information among the subroutines is shown in Figure 3.

A.1 Names

The core of the virus is a pair of binary modules, one for the VAX architecture and the other for the Sun architecture. These are linkable modules, and thus have name lists for their internal procedures. Many of the original names are included here with the descriptions of the functions the routines performed.

It is surprising that the names are included, and astonishing that they are meaningful. Some simple techniques, such as randomizing the procedure names, would have removed a number of clues to the function of the virus.

A.2 main

The main module, the starting point of any C language program, does some initialization, processes its command line, and then goes off into the loop which organizes all of the real work.

A.2.1 Initialization

The program first takes some steps to hide itself. It changes the ''zeroth'' argument, which is the process name, to `sh`. Thus, no matter how the program was invoked, it would show up in the process table with the same name as the Bourne Shell, a program which often runs legitimately.

The program also sets the maximum core dump size to zero blocks. If the program crashed[19] it would not leave a core dump behind to help investigators. It also turns off handling of write errors on pipes, which normally cause the program to exit.

The next step is to read the clock, store the current time in a local variable, and use that value to seed the random number generator.

A.2.2 Command line argument processing

The virus program itself takes an optional argument `-p` which must be followed by a decimal number, which seems to be a process id of the parent which spawned it. It uses this number later to kill that process, probably to ''close the door'' behind it.

The rest of the command line arguments are ''object names''. These are names of files it tries to load into its address space. If it can't load one of them, it quits. If the `-p` argument is given, it also deletes the object files, and later tries to remove the disk image of running virus, as well as the file `/tmp/.dumb`. (This file is not referenced anywhere else in the virus, so it is unclear why it is deleted.)

The program then tried a few further steps, exiting (''bailing out'') if any of them failed:

- It checked that it had been given at least one object on the command line.
- It checked to see if it had successfully loaded in the object `l1.c`.

If the ''-p'' argument was given, the program closes all file descriptors, in case there are any connections open to the parent.

The program then erases the text of the argument array, to further obscure how it was started (perhaps to hide anything if one were to get a core image of the running virus.)

It scans all of the network interfaces on the machine, gets the flags and addresses of each interface. It tries to get the point-to-point address of the interface, skipping the loopback address. It also stores the netmask for that network [23].

Finally, it kills off the process id given with the ''-p'' option. It also changes the current process group, so that it doesn't die when the parent exits. Once this is cleaned up, it falls into the *doit* routine which performs the rest of the work.

A.3 doit routine

This routine is where the program spends most of its time.

A.3.1 Initialization

Like the main routine, it seeds the random number generator with the clock, and stores the clock value to later measure how long the virus has been running on this system.

It then tries *hg*. If that fails, it tries *hl*. If that fails, it tries *ha*.

It then tries to check if there is already a copy of the virus running on this machine. Errors in this code contributed to the large amounts of computer time taken up by the virus. Specifically:

- On a one-in-seven chance, it won't even try to test for another virus.
- The first copy of the virus to run is the only one which listens for others; if multiple infections occur ''simultaneously'' they will not ''hear'' each other, and all but one will fail to listen (see section A.12).

The remainder of the initialization routine seems designed to send a single byte to address 128.32.137.13, which is *ernie.berkeley.edu*, on port 11357. This never happens, since the author used the *sendto* function on a TCP

[19] For example, the virus was originally compiled using 4.3BSD declaration files. Under 4.2BSD, the alias name list did not exist, and code such as the virus which assumes aliases are there can crash and dump core.

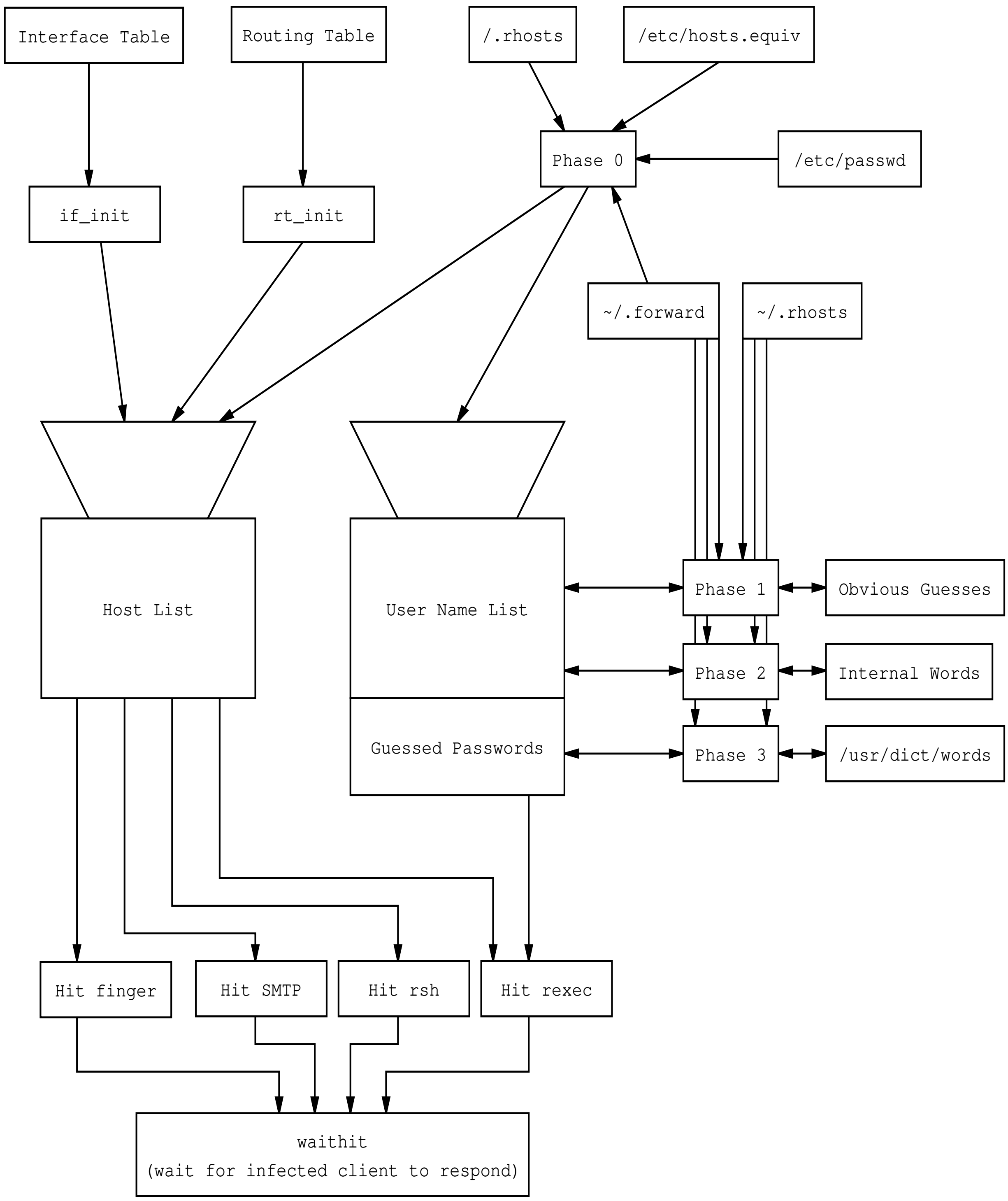

Figure 3: The structure of the attacking engine.

stream connection, instead of a UDP datagram socket.[20] We have no explanation for this; it only tries to send this packet with a one in fifteen random chance.

A.3.2 Main loop

An infinite loop comprises the main active component of the virus. It calls the *cracksome* routine[21] which tries to find some hosts that it can break in to. Then it waits 30 seconds, listening for other virus programs attempting to break in, and tries to break into another batch of machines.

After this round of attacks, it forks, creating two copies of the virus; the original (parent) dies, leaving the fresh copy. The child copy has all of the information the parent had, while not having the accumulated CPU usage of the parent. It also has a new process id, making it hard to find.

Next, the *hg*, *hl*, and *ha* routines search for machines to infect (see Appendix A.5). The program sleeps for 2 minutes, and then checks to see if it has been running for more than 12 hours, cleaning up some of the entries in the host list if it has.

Finally, before repeating, it checks the global variable `pleasequit`. If it is set, *and* if it has tried more than 10 words from its own dictionary against existing passwords, it quits. Thus forcing `pleasequit` to be set in the system libraries will do very little to stem the progress of this virus[22].

A.4 Cracking routines

This collection of routines is the ‘‘brain’’ of the virus. *cracksome*, the main switch, chooses which of four strategies to execute. It is would be the central point for adding new strategies if the virus were to be further extended. The virus works each strategy through completely, then switches to the next one. Each pass through the cracking routines only performs a small amount of work, but enough state is remembered in each pass to continue the next time around.

A.4.1 cracksome

The *cracksome* routine is the central switching routine of the cracking code. It decides which of the cracking strategies is actually exercised next. Again, note that this routine was named in the global symbol table. It could have been given a confusing or random name, but it was actually clearly labelled, which lends some credence to the idea that the virus was released prematurely.

[20] If the author had been as careful with error checking here as he tried to be elsewhere, he would have noted the error ‘‘socket not connected’’ every time this routine is invoked.

[21] This name was actually in the symbol table of the distributed binary!

[22] Although it was suggested very early [24].

A.4.2 Phase 0

The first phase of the *cracksome* routines reads through the `/etc/hosts.equiv` file to find machine names that would be likely targets. While this file indicates what hosts the current machine trusts, it is fairly common to find systems where all machines in a cluster trust each other, and at the very least it is likely that people with accounts on this machine will have accounts on the other machines mentioned in `/etc/hosts.equiv`.

It also reads the `/.rhosts` file, which lists the set of machines that this machine trusts root access from. Note that it does not take advantage of the trust itself [25] but merely uses the names as a list of additional machines to attack. Often, system managers will deny read access to this file to any user other than root itself, to avoid providing any easy list of secondary targets that could be used to subvert the machine; this practice would have prevented the virus from discovering those names, although `/.rhosts` is very often a subset of `/etc/hosts.equiv`.

The program then reads the entire local password file, `/etc/passwd`. It uses this to find personal `.forward` files, and reads them in search of names of other machines it can attack. It also records the user name, encrypted password, and *GECOS* information string, all of which are stored in the `/etc/passwd` file. Once the program scanned the entire file, it advanced to Phase 1.

A.4.3 Phase 1

This phase of the cracking code attacked passwords on the local machine. It chose several likely passwords for each user, which were then encrypted and compared against the encryptions obtained in Phase 0 from `/etc/passwd`:

- No password at all.
- The user name itself.
- The user name appended to itself.
- The second of the comma separated *GECOS* information fields, which is commonly a nickname.
- The remainder of the full name after the first name in the *GECOS* fields, i.e. probably the last name, with the first letter converted to lower case.
- This ‘‘last name’’ reversed.

All of these attacks are applied to fifty passwords at a time from those collected in Phase 0. Once it had tried to guess the passwords for all local accounts, it advanced to Phase 2.

A.4.4 Phase 2

Phase 2 takes the internal word list distributed as part of the virus (see Appendix B) and shuffles it. Then it takes the words one at a time and decodes them (the high bit is set on all of the characters to obscure them) and tries them against all collected passwords. It maintains a global variable `nextw` as an index into this table. The main loop uses this to prevent `pleasequit` from causing the virus to exit until at least ten of the words have been checked against all of the encryptions in the collected list.

Again, when the word list is exhausted the virus advances to Phase 3.

A.4.5 Phase 3

Phase 3 looks at the local `/usr/dict/words` file, a 24474 word list distributed with 4.3BSD (and other UNIX systems) as a spelling dictionary. The words are stored in this file one word per line. One word at a time is tried against all encrypted passwords. If the word begins with an upper case letter, the letter is converted to lower case and the word is tried again.

When the dictionary runs out, the phase counter is again advanced to 4 (thus no more password cracking is attempted).

A.5 H routines

The ‘‘h routines’’ are a collection of routines with short names, such as *hg*, *ha*, *hi*, and *hl*, which search for other hosts to attack.

A.5.1 hg

The *hg* routine calls *rt_init* (if it has not already been called) to scan the routing table, and records all gateways except the loopback address in a special list. It then tries a generic attack routine to attack via `rsh`, `finger`, and SMTP. It returns after the first successful attack.

A.5.2 ha

The *ha* routine goes through the gateway list and connects to TCP port 23, the telnet port, looking for gateways which are running telnet listeners. It randomizes the order of such gateways and calls *hn* (our name) with the network number of each gateway. The *ha* returns after *hn* reports that it has succeeded broken into a host.

A.5.3 hl

The *hl* routine iterates through all the addresses for the local machine calling *hn* with the network number for each one. It returns if *hn* indicates success in breaking into a host.

A.5.4 hi

The *hi* routine goes through the internal host list (see section A.4.2) and tries to attack each host via `rsh`, `finger`, and SMTP. It returns if when one host is infected.

A.5.5 hn

The *hn* routine (our name) followed *hi* takes a network number as an argument. Surprisingly it returns if the network number supplied is the same as the network number of any of the interfaces on the local machine. For Class A addresses it uses the Arpanet IMP convention to create possible addresses to attack (net.[1-8].0.[1-255]). For all other networks it guesses hosts number one through 255 on that network. It randomizes the order of this list of possible hosts and tries to attack up to twenty of them using `rsh`, `finger`, and SMTP. If a host does not accept connections on TCP port 514, the `rsh` port, *hn* will not try to attack it. If a host is successfully attacked *hn* returns.

A.5.6 Usage

The ‘‘h routines’’ are called in groups in the main loop; if the first routine succeedes in finding a vulnerable host the remaining routines are not called in the current pass. Each routine returns after it finds one vulnerable host. The *hg* routine is always called first, which indicates the virus really wanted to infect gateway machines. Next comes *hi* which tried to infect normal hosts found via *cracksome*. If *hi* fails, *ha* is called, which seemed to try breaking into hosts with randomly guessed addresses on the far side of gateways. This assumes that all the addresses for gateways had been obtained (which is not trivial to verify from the convoluted code in *rt init*), and implies that the virus would prefer to infect a gateway and from there reach out to the gateway's connected networks, rather than trying to hop the gateway directly. If *hg*, *hi*, and *ha* all failed to infect a host, then *hl* is called which is similar to *ha* but uses for local interfaces for a source of networks.

It is not clear that *ha* and *hl* worked. Because *hn* returns if the address is local, *hl* appears to have no chance of succeeding. If alternate addresses for gateways are indeed obtained by other parts of the virus then *ha* could work. But if only the addresses in the routing table were used it could not work, since by definition these addresses must be on a directly connected network. Also, in our monitoring we never detected an attack on a randomly generated address. These routines do not seem to have been functional.

A.6 Attack routines

There are a collection of attack routines, all of which try to obtain a Bourne Shell running on the targeted machine. See Appendix A.7 for a description of the `l1.c` program, used by all the attack routines.

A.6.1 hu1

The *hu1* routine is called by the Phase 1 and Phase 3 *cracksome* subroutines. Once a password for user name guessed correctly, this routine is called with a host name read from either the user's `.forward` or `.rhosts` files. In order to assume the user's id it then tries to connect to the local machine's `rexec` server using the guessed name and password. If successful it runs an `rsh` to the target machine, trying to execute a Bourne Shell, which it uses to send over and compile the `l1.c` infection program.

A.6.2 Hit SMTP

This routine make a connection to TCP port 25, the SMTP port, of a remote machine and used it to take advantage of the sendmail bug. It attempts to use the debug option to make `sendmail` run a command (the "recipient" of the message), which transfers the `l1.c` program included in the body of the message.

A.6.3 Hit finger

The "hit finger" routine tries to make a connection to TCP port 79, the finger port, of the remote machine. Then it creates a "magic packet" which consists of

- A 400 byte "runway" of VAX "nop" instructions, which can be executed harmlessly.
- A small piece of code which executes a Bourne Shell.
- A stack frame, with a return address which would hopefully point into the code.

Note that the piece of code is VAX code, and the stack frame is a VAX frame, in the wrong order for the Sun. Thus, although the Sun finger daemon has the same bug as the VAX one, this piece of code cannot exploit it.

The attack on the finger daemon is clearly a lysogenetic "viral" attack (see Section 1.2), since although a worm doesn't modify the host machine at all, the finger attack does modify the running finger daemon process. The "injected DNA" component of the virus contained the VAX instructions shown in Figure 4.

The *execve* system call causes the current process to be replaced with an invocation of the named program; `/bin/sh` is the Bourne Shell, a UNIX command interpreter. In this case, the shell winds up running with its input coming from, and its output going to, the network connection. The virus then sends over the `l1.c` bootstrap program.

A.6.4 Hit rsh

This unlabeled routine tries `rsh` to the target host (assuming it can get in as the current user). It tries three different names for the `rsh` binary,

- `/usr/ucb/rsh`
- `/usr/bin/rsh`
- `/bin/rsh`

If one of them succeeds, it tries to resynchronize (see Appendix A.8.1) the connection; if that doesn't succeed within thirty seconds it kills off the child process. If successful the connection can then be used to launch the `l1.c` "grappling hook" program at the victim.

Note that this infection method doesn't specify a user name to attack; if it gets into the remote account, it is because the user that the virus is running as also has an account on the other machine which trusts the originating machine.

A.6.5 Hit rexec

The *hit rexec* routine uses the remote execution system which is similar to `rsh`, but designed for use by programs. It connects and sends the user name, the password, and `/bin/sh` as the command to execute.

A.6.6 makemagic

This routine tries to make a telnet connection to each of the available addresses for the current victim. It broke the connections immediately, often producing error reports from the telnet daemon, which were recorded, and provide some of the earliest reports of attack attempts.[23]

If it succeedes in reaching the host, it creates a TCP listener on a random port number which the infected machine would eventually connect back to.

A.7 Grappling Hook

A short program, named `l1.c`, is the common grappling hook that all of the attack routines use to pull over the rest of the virus. It is robustly written, and fairly portable. It ran on a number of machines which were neither VAX or Sun, loading them down as well, but only making them peripheral victims of the virus.

The first thing it does is delete the binary it was running from. It checks that it has three arguments (exiting if there aren't three of them). It closes all file descriptors and then forks, exiting if the fork fails. If it succeeds, the parent exits; this leaves no connection from the child to the infection route.

Next, it creates a TCP connection back to the address given as the first argument, and the port given as the second. Then it sends over the magic number given as the third. The text of each argument is erased immediately after it is used. The stream connection is then reused as the program's standard input and output.

[23] On fast machines, such as the DEC VAX 3200, there may be no record of these attacks, since the connection is handed off fast enough to satisfy the daemon.

```
pushl   $68732f      push '/sh<NUL>'
pushl   $6e69622f    push '/bin'
movl    sp,r10       save address of start of string
pushl   $0           push 0 (arg 3 to execve)
pushl   $0           push 0 (arg 2 to execve)
pushl   r10          push string addr (arg 1 to execve)
pushl   $3           push argument count
movl    sp,ap        set argument pointer
chmk    $3b          do "execve" kernel call.
```

Figure 4: VAX intructions for the `finger` attack.

A loop reads in a length (as a network byte order 32-bit integer) and then a filename. The file is unlinked and opened for write, and then the file itself is read in (using the number of bytes read in earlier.) On any error, all of the files are unlinked. If the length read in is -1, the loop exits, and a Bourne Shell is executed (replacing the `l1` program, and getting its input from the same source.)

A.8 Install Routines

There are a variety of routines used to actually move the virus from one machine to the other. They deal with the ''virus protocol'' connection made by the `l1.c` injected program or with the shell that it spawns.

A.8.1 resynch

The *resynch* routine sends commands to a remote shell, requesting that it echo back a specific randomly chosen number. It then waits a certain amount of time for a response. This routine is used to indicate when the various subprograms of the infection procedure have compiled or executed and a Bourne Shell prompt is available again.

A.8.2 waithit

This routine does much of the high level work. It waits (up to 2 minutes) for a return connection from a victim (which has had `l1.c` injected into it.) It then tries to read a magic number (which had been previously sent to that victim as a command line argument to the `l1` program) and gives up after ten seconds.

After the connection is established, all of the current ''objects'' in storage in the virus are fed down the connection into the victim. Then it tries to resynchronize, and if it succeeds, sends down commands to

- set the `PATH` of the victim shell
- try to delete `sh` in the current directory (`/usr/tmp`)
- if the delete fails, pick a random name to use instead [24]
- scan the list of objects, looking for names ending in `.o`
- link and run each of these, with the command line arguments
    - `-p $$`, where `$$` is the process id of the victim shell
    - each object name
- resynchronize; if this fails, assume that the virus succeeded (since the `-p` option tells the virus to kill off the parent shell) and set flag bit 1 of the host list entry (the host list is detailed in section A.9).
- delete the compiled program, and go on to the next object.

Thus, to add another machine type, the virus merely needs to be started with a new object binary as a command line option, which will then be propagated to the next infected host and tried.

Note that the path used here was `PATH= bin: /usr/bin: /usr/ucb` which is certainly reasonable on most systems. This protects systems with ''unusual'' filesystem layouts, and suggests that complete consistency among systems makes them more vulnerable.

A.9 Host modules

These are a set of routines designed to collect names and addresses of target hosts in a master list. Each entry contains up to six addresses, up to twelve names, and a flags field.

A.9.1 Name to host

This routine searches the host list for a given named host, returns the list entry describing it, and optionally adds it to the list if it isn't there already.

[24] Since the delete command used (`rm -f`) did not remove directories, creating a directory `/usr/tmp/sh` stoped the virus[26]. However, the virus would still use CPU resources attempting to link the objects, even though it couldn't write to the output file (since it was a directory).

A.9.2 Address to host

This routine searches the host list for a given host address, returns the list entry describing it, and optionally adds it to the list if it isn't there already.

A.9.3 Add address/name

These two routines added an address or a name to a host list entry, checking to make sure that the address or name was not already known.

A.9.4 Clean up table

This routine cycles through the host list, and removes any hosts which only have flag bits 1 and 2 set (and clears those bits.) Bit 1 is set when a resynchronize (in *waithit*) fails, probably indicating that this host ''got lost''. Bit 2 is set when a named host has no addresses, or when several different attack attempts fail. Bit 3 is set when Phase 0 of the crack routines successfully retrieves an address for the host.

A.9.5 Get addresses

This routine takes an entry in the host table and tries to fill in the the gaps. It looks up an address for a name it has, or looks up a name for the addresses it has. It also includes any aliases it can find.

A.10 Object routines

These routines are what the system uses to pull all of its pieces into memory when it starts (after the host has been infected) and then to retrieve them to transmit to any host it infects.

A.10.1 Load object

This routine opens a file, determines its length, allocating the appropriate amount of memory, reads it in as one block, decodes the block of memory (with XOR). If the object name contains a comma, it moves past it and starts the name there.

A.10.2 Get object by name

This routine returns a pointer to the requested object. This is used to find the pieces to download when infecting another host.

A.11 Other initialization routines

A.11.1 if init

This routine scans the array of network interfaces. It gets the flags for each interface, and makes sure the interface is UP and RUNNING (specific fields of the flag structure). If the entry is a point to point type interface, the remote address is saved and added to the host table. It then tries to enter the router into the list of hosts to attack.

A.11.2 rt init

This routine runs `netstat -r -n` as a subprocess. This shows the routing table, with the addresses listed numerically. It gives up after finding 500 gateways. It skips the default route, as well as the loopback entry. It checks for redundant entries, and checks to see if this address is already an interface address. If not, it adds it to the list of gateways.

After the gateway list is collected, it shuffles it and enters the addresses in the host table.

A.12 Interlock routines

The two routines *checkother* and *othersleep* are at the heart of the excessive propagation of the virus. It is clear that the author intended for the virus to detect that a machine was already infected, and if so to skip it. The code is actually fraught with timing flaws and design errors which lead it to permit multiple infections, probably more often than the designer intended[25].

An active infection uses the *othersleep* routine for two purposes, first to sleep so that it doesn't use much processor time, and second to listen for requests from ''incoming'' viruses. The virus which is running *othersleep* is referred to as the ''listener'' and the virus which is running *checkother* is referred to as the ''tester''.

A.12.1 Checkother

The tester tries to connect to port 23357 on the local machine (using the loopback address, 127.0.0.1) to see if it can connect to a listener. If any errors occur during this check, the virus assumes that no listener is present, and tries to become a listener itself.

If the connection is successful, the checker sends a magic number[26], and listens (for up to 300 seconds) for a magic number from the listener[27]. If the magic number is wrong, the checker assumes it is being spoofed and continues to run.

The checker then picks a random number, shifts it right by three (Twing away the lower three bits) and sends it to the listener. It expects a number back within ten seconds, which it adds to the one sent. If this sum is even, the

[25] This behavior was noted by both looking at the code and by creating a testbed setup, manually running a program that performs the checking and listening functions.

[26] $874697_{16}, 8865431_{10}, 041643227_8$

[27] $148898_{16}, 1345688_{10}, 05104230_8$

sender increments `pleasequit`, which (as noted in section A.3.2) does very little.

Once it has finished communicating (or failing to communicate) with the listener, the checker sleeps for five seconds and tries to become a listener. It creates a TCP stream socket, sets the socket options to indicate that it should allow multiple binds to that address (in case the listener *still* hasn't exited, perhaps?) and then binds the socket to port 23357, and listens on it (permitting a backlog of up to ten pending connections.)

A.12.2 Othersleep

The *othersleep* routine is run when the main body of the virus wants to idle for a period of time. This was apparently intended to help the virus "hide" so that it wouldn't use enough processor time to be noticed. While the main program sleeps, the listener code waits to see if any checkers have appeared and queried for the existence of a listener, as a simple "background task" of the main virus.

The routine first checks to see if it has been set up as a listener; if not, it calls the normal *sleep* function to sleep for the requested number of seconds, and returns.

If it is set up as a listener, it listens on the checking port with a timeout. If it times out, it returns, otherwise it deals with the connection and subtracts the elapsed real time from the time out value.

The body of the listener "accepts" the connection, and sends a magic number to the checker. It then listens (for up to 10 seconds) for the checker's magic number, and picks a random number. It shifts the random number right by three, discarding the lower bits, and sends it up to the checker; it then listens (for up to 10 seconds) for a random number from the checker. If any of these steps fail, the connection is closed and the checker is ignored.

Once the exchanges have occurred, the address of the incoming connection is compared with the loopback address. If it is not from the loopback address, the attempt is ignored. If it is, then if the sum of the exchanged random numbers is odd, the listener increments `pleasequit` (with little effect, as noted in section A.3.2) and closes the listener connection.

B Built in dictionary

432 words were included:

aaa	academia	aerobics
airplane	albany	albatross
albert	alex	alexander
algebra	aliases	alphabet
ama	amorphous	analog
anchor	andromache	animals
answer	anthropogenic	anvils
anything	aria	ariadne
arrow	arthur	athena
atmosphere	aztecs	azure
bacchus	bailey	banana
bananas	bandit	banks
barber	baritone	bass
bassoon	batman	beater
beauty	beethoven	beloved
benz	beowulf	berkeley
berliner	beryl	beverly
bicameral	bob	brenda
brian	bridget	broadway
bumbling	burgess	campanile
cantor	cardinal	carmen
carolina	caroline	cascades
castle	cat	cayuga
celtics	cerulean	change
charles	charming	charon
chester	cigar	classic
clusters	coffee	coke
collins	commrades	computer
condo	cookie	cooper
cornelius	couscous	creation
creosote	cretin	daemon
dancer	daniel	danny
dave	december	defoe
deluge	desperate	develop
dieter	digital	discovery
disney	dog	drought
duncan	eager	easier
edges	edinburgh	edwin
edwina	egghead	eiderdown
eileen	einstein	elephant
elizabeth	ellen	emerald
engine	engineer	enterprise
enzyme	ersatz	establish
estate	euclid	evelyn
extension	fairway	felicia
fender	fermat	fidelity
finite	fishers	flakes
float	flower	flowers
foolproof	football	foresight
format	forsythe	fourier
fred	friend	frighten
fun	fungible	gabriel
gardner	garfield	gauss
george	gertrude	ginger
glacier	gnu	golfer
gorgeous	gorges	gosling
gouge	graham	gryphon
guest	guitar	gumption
guntis	hacker	hamlet
handily	happening	harmony

harold	harvey	hebrides
heinlein	hello	help
herbert	hiawatha	hibernia
honey	horse	horus
hutchins	imbroglio	imperial
include	ingres	inna
innocuous	irishman	isis
japan	jessica	jester
jixian	johnny	joseph
joshua	judith	juggle
julia	kathleen	kermit
kernel	kirkland	knight
ladle	lambda	lamination
larkin	larry	lazarus
lebesgue	lee	leland
leroy	lewis	light
lisa	louis	lynne
macintosh	mack	maggot
magic	malcolm	mark
markus	marty	marvin
master	maurice	mellon
merlin	mets	michael
michelle	mike	minimum
minsky	moguls	moose
morley	mozart	nancy
napoleon	nepenthe	ness
network	newton	next
noxious	nutrition	nyquist
oceanography	ocelot	olivetti
olivia	oracle	orca
orwell	osiris	outlaw
oxford	pacific	painless
pakistan	pam	papers
password	patricia	penguin
peoria	percolate	persimmon
persona	pete	peter
philip	phoenix	pierre
pizza	plover	plymouth
polynomial	pondering	pork
poster	praise	precious
prelude	prince	princeton
protect	protozoa	pumpkin
puneet	puppet	rabbit
rachmaninoff	rainbow	raindrop
raleigh	random	rascal
really	rebecca	remote
rick	ripple	robotics
rochester	rolex	romano
ronald	rosebud	rosemary
roses	ruben	rules
ruth	sal	saxon
scamper	scheme	scott
scotty	secret	sensor

serenity	sharks	sharon
sheffield	sheldon	shiva
shivers	shuttle	signature
simon	simple	singer
single	smile	smiles
smooch	smother	snatch
snoopy	soap	socrates
sossina	sparrows	spit
spring	springer	squires
strangle	stratford	stuttgart
subway	success	summer
super	superstage	support
supported	surfer	suzanne
swearer	symmetry	tangerine
tape	target	tarragon
taylor	telephone	temptation
thailand	tiger	toggle
tomato	topography	tortoise
toyota	trails	trivial
trombone	tubas	tuttle
umesh	unhappy	unicorn
unknown	urchin	utility
vasant	vertigo	vicky
village	virginia	warren
water	weenie	whatnot
whiting	whitney	will
william	williamsburg	willie
winston	wisconsin	wizard
wombat	woodwind	wormwood
yacov	yang	yellowstone
yosemite	zap	zimmerman

C Cast of Characters

This is an alphabetical list of all the people mentioned in section 3, their network addresses, and affiliations.

Don Alvarez <boomer@space.mit.edu>
MIT Center for Space Research

Richard Basch <probe@athena.mit.edu>
MIT Athena and SIPB

Don Becker <becker@trantor.harris-atd.com>
Harris Corporation and MIT SIPB.

Matt Bishop <bishop@bear.dartmouth.edu>
Dartmouth University

Hal Birkeland <hkbirke@athena.mit.edu>
MIT Media Laboratory

Keith Bostic <bostic@okeeffe.berkeley.edu>

University of California, Berkeley

Russell Brand <brand@lll-crg.llnl.gov>
Lawrence Livermore National Laboratory

James D. Bruce <jdb@delphi.mit.edu>
MIT Information Systems

John Bruner <jdb@mordor.s1.gov>
Lawrence Livermore National Laboratory

Liudvikas Bukys <bukys@cs.rochester.edu>
University of Rochester

Chuck Cole <cole@lll-crg.llnl.gov>
Lawrence Livermore National Laboratory

Pascal Chesnais <lacsap@media-lab.media.mit.edu>
MIT Media Laboratory

Jean Diaz <ambar@athena.mit.edu>
Oracle Corporation and MIT SIPB

Dave Edwards <dle@sri.com>
SRI, International

Mark Eichin <eichin@athena.mit.edu>
MIT Athena and SIPB

Kent England <kwe@bu-cs.bu.edu>
Boston University

Paul Flaherty <paulf@jessica.stanford.edu>
Stanford University

Jim Fulton <jim@expo.lcs.mit.edu>
MIT X Consortium

Robert French <rfrench@athena.mit.edu>
MIT SIPB and Project Athena

Dan Geer <geer@athena.mit.edu>
MIT Project Athena

Paul Graham <pg@harvard.edu>
Harvard University

Chris Hanson <cph@zurich.ai.mit.edu>
MIT AI Laboratory

Sergio Heker <heker@jvnca.csc.org>
John Von Neumann National Supercomputer Center

Ray Hirschfeld <ray@math.mit.edu>
MIT Math Department/AI Laboratory

Ron Hoffmann <hoffmann@bitsy.mit.edu>
MIT Telecommunications Network Group

Jon Kamens <jik@athena.mit.edu>
MIT Project Athena and SIPB

Mike Karels <karels@ucbarpa.berkeley.edu>
University of California, Berkeley

John Kohl <jtkohl@athena.mit.edu>
Digital Equipment Corporation, MIT Athena and SIPB

Rich Kulawiec <rsk@mace.cc.purdue.edu>
Purdue

Phil Lapsley <phil@berkeley.edu>
University of California, Berkeley

Milo Medin <medin@nsipo.nasa.gov>
NASA Ames

Steve Miller <steve@umiacs.umd.edu>
University of Maryland

Russ Mundy <mundy@beast.ddn.mil>
Defense Communications Agency

Mike Muuss <mike@brl.mil>
Ballistic Research Laboratory

Eugene Myers <EDMyers@dockmaster.arpa>
National Computer Security Center

Peter Neumann <neumann@csl.sri.com>
SRI International

Mike Patton <map@lcs.mit.edu>
MIT LCS

Kurt Pires <kjpires@berkeley.edu>
University of California, Berkeley

Mark Reinhold <mbr@lcs.mit.edu>
MIT Laboratory of Computer Science

Jon Rochlis <jon@bitsy.mit.edu>
MIT Telecommunications Network Group and SIPB

Miek Rowan <mtr@mace.cc.purdue.edu>
Purdue University

Jerry Saltzer <Saltzer@athena.mit.edu>
MIT Laboratory of Computer Science and Project Athena

Jeff Schiller <jis@bitsy.mit.edu>
MIT Telecommunications Network Group, Athena, and SIPB

Mike Shanzer <shanzer@athena.mit.edu>
MIT Project Athena

Tim Shepard <shep@ptt.lcs.mit.edu>
MIT Laboratory of Computer Science

Bill Sommerfeld <wesommer@athena.mit.edu>
Apollo Computer, MIT Athena and SIPB

Gene Spafford <spaf@cs.purdue.edu>
Purdue University

Mike Spitzer <mjs@mentor.cc.purdue.edu>
Purdue University

Cliff Stoll <cliff@cfa200.harvard.edu>
Harvard University

Andy Sudduth <sudduth@harvard.edu>
Harvard University

Ted Ts'o <tytso@athena.mit.edu>
MIT Athena and SIPB

Edward Wang <edward@berkeley.edu>
University of California, Berkeley

Peter Yee <yee@ames.arc.nasa.gov>
NASA Ames

Stan Zanarotti <srz@lcs.mit.edu>
MIT Laboratory of Computer Science and SIPB

References

[1] R. Hinden, J. Haverty, and A. Sheltzer, “The DARPA Internet: Interconnecting Heterogeneous Computer Networks with Gateways,” *IEEE Computer Magazine*, vol. 16, num. 9, pp. 38–48, September 1983.

[2] J. S. Quarterman and J. C. Hoskins, “Notable Computer Networks,” in *Communications of the ACM*, vol. 29, num. 10, pp. 932–971, October 1986.

[3] S. E. Luria, S. J. Gould, and S. Singer, *A View of Life*. Menlo Park, California: Benjamin/Cummings Publishing Company, Inc., 1981.

[4] J. Watson *et al.*, *Molecular Biology of the Gene*. Menlo Park, California: Benjamin/Cummings Publishing Company, Inc., 1987.

[5] G. G. Simpson and W. S. Beck, *Life: An Introduction to Biology*. New York, New York: Harcourt, Brace and Ward, Inc., 1965.

[6] L. Castro *et al.*, “Post Mortem of 3 November ARPANET/MILNET Attack.” National Computer Security Center, Ft. Meade MD, 8 November 1988.

[7] P. J. Denning, “Computer Viruses,” *American Scientist*, vol. 766, pp. 236–238, May-June 1988.

[8] D. Seeley, “A Tour of the Worm,” in *USENIX Association Winter Conference 1989 Proceedings*, pp. 287–304, January 1989.

[9] E. H. Spafford, “The Internet Worm Program: An Analysis,” *ACM SIGCOM*, vol. 19, January 1989.

[10] K. Harrenstien, “NAME/FINGER Protocol Protocol,” Request For Comments NIC/RFC 742, Network Working Group, USC ISI, Novemeber 1977.

[11] J. Markoff, “Computer Snarl: A ‘Back Door’ Ajar,” *New York Times*, p. B10, 7 November 1988.

[12] J. B. Postel, “Simple Mail Transfer Protocol,” Request For Comments NIC/RFC 821, Network Working Group, USC ISI, August 1982.

[13] S. Bellovin, “The worm and the debug option,” in *Forum on Risks to the Public in Computers and Related Systems*, vol. 7, num. 74, ACM Committee on Computers and Public Policy, 10 November 1988.

[14] J. Collyer, “Risks of unchecked input in C programs,” in *Forum on Risks to the Public in Computers and Related Systems*, vol. 7, num. 74, ACM Committee on Computers and Public Policy, 10 November 1988.

[15] J. Saltzer and M. Schroeder, “The Protection of Information in Computer Systems,” in *Proc. IEEE*, vol. 63, num. 9, pp. 1278–1308, IEEE, September 1975.

[16] J. Steiner, C. Neuman, and J. Schiller, “Kerberos: An Authentication Service for Open Network Systems,” in *USENIX Association Winter Conference 1988 Proceedings*, pp. 191–202, February 1988.

[17] M. R. Horton, “How to Read the Network News,” *UNIX User's Supplementary Documents*, April 1986.

[18] P. Mockapetris, ''Domain Names - Concepts And Facilities,'' Request For Comments NIC/RFC 1034, Network Working Group, USC ISI, November 1987.

[19] J. Markoff, ''Author of Computer 'Virus' Is Son of U.S. Electronic Security Expert,'' *New York Times*, p. A1, 5 November 1988.

[20] P. G. Neumann, ed., *Forum on Risks to the Public in Computers and Related Systems*, vol. 7, num. 69, ACM Committee on Computers and Public Policy, 3 November 1988.

[21] ???, ''College Whiz ''Put Virus in Computers'','' *Boston Herald*, p. 1, 5 November 1988.

[22] J. Markoff, ''U.S. Is Moving to Restrict Access To Facts About Computer Virus,'' *New York Times*, p. A28, 11 November 1988.

[23] J. Mogul and J. B. Postel, ''Internet Standard Subnetting Procedure,'' Request For Comments NIC/RFC 950, Network Working Group, USC ISI, August 1985.

[24] G. Spafford, ''A cure!!!!!,'' in *Forum on Risks to the Public in Computers and Related Systems*, vol. 7, num. 70, ACM Committee on Computers and Public Policy, 3 November 1988.

[25] R. W. Baldwin, *Rule Based Analysis of Computer Security.* PhD thesis, MIT EE, June 1987.

[26] G. Spafford, ''A worm ''condom'','' in *Forum on Risks to the Public in Computers and Related Systems*, vol. 7, num. 70, ACM Committee on Computers and Public Policy, 3 November 1988.