

Proseminar Software Desaster
Desaster: Internetvirus

Oliver Kutter

12. November 2002

Inhaltsverzeichnis

1	Einleitung - Was passierte	2
2	Virus oder Wurm ?	2
3	Chronologie der Ereignisse	3
3.1	Mittwoch 2.11 - Der Beginn	3
3.2	Donnerstag 3.11 - schlechte Nachrichten	3
3.3	Freitag 4.11 - Beobachtungen am Virus	4
3.4	Samstag 5.11 - Danach	4
4	Das Vorgehen des Virus	5
4.1	Wie drang es ein?	5
4.2	Wen griff es an?	5
4.3	Welche Rechner griff es an?	5
4.4	Was es nicht tat	6
5	Die Attacken im Detail	6
5.1	Sendmail Debug	6
5.2	Finger Daemon Bug	6
5.3	Rexec und Passwörter	7
5.4	Rsh	7
5.5	Informationsfluß	7
6	Selbstschutzmechanismen des Virus	8
7	Designfehler	8
7.1	Mehrfachinfizierung	8
7.2	ungenutzte Schwachstellen	9
8	Gegenmaßnahmen	9
9	Die Struktur des Virus	11
9.1	Strukturdiagramm	11
9.2	Die Hauptroutinen	12
9.2.1	Cracking Routinen	12
9.2.2	Angriffsroutinen	13
9.3	Das eingebaute Wörterbuch	15
10	Schlußfolgerungen	15
11	Literaturverzeichnis	17

1 Einleitung - Was passierte

Im November 1988, wurde das Internet, damals ein Netzwerk von 60000 Computern die das TCP/IP-Protokoll implementierten, von einem Virus angegriffen. Das Programm kompromitierte die fingerd und sendmail Daemons der angegriffenen Rechner und verbreitete sich auch über rexec und rsh rasch im gesamten Netzwerk. Dieser Vortrag soll einen Überblick darüber geben, was damals passierte, wie das Virus vorging, an welchen Punkten sein Angriff ansetzte und wie es sich selbst reproduzierte und verbreitete.

2 Virus oder Wurm ?

Virus Im biologischen Sinne versteht man unter einem Virus, einen Erreger der sich nur durch die Zellen eines Wirtes vermehren kann. Viren fehlt die Fähigkeit zur eigenständigen Reproduktion. Ein Virus dringt in einen Wirt ein und bemächtigt sich einer Zelle. In die DNA der Zelle injiziert das Virus seine eigene DNA. Die umgeschriebene DNA der Zelle bewirkt das die Zelle fortan nur noch ununterbrochen neue Viren produziert. Bis sich schließlich innerhalb der Zelle soviele Viren befinden, daß die Zelle stirbt und damit die Viren freigibt. Es gibt aber auch Viren die ihre Wirtszelle nicht zerstören, sondern nur als Fabrik für Viren benutzen.

Wurm Ein Wurm ist ein Organismus mit einem langen, segmentierten Körper. Aufgrund ihrer Körperform können sich Würmer um Hindernisse herumschlängeln und somit fast überall hin vordringen. Der Bandwurm z. Bsp. ist ein Parasit. Er lebt in einem Wirtsorganismus, und ernährt sich direkt von den Nährstoffen die eigentlich für die Zellen seines Wirtes bestimmt wären. Diese Würmer vermehren sich indem sie einen, mit Eiern gefüllten Teil ihres Körpers abtrennen.

Im Bezug auf das Programm, daß das Internet infizierte müssen wir entscheiden, welcher Teil des Systems der Wirt ist. In Frage kommen das Netzwerk, ein einzelner Rechner, ein Programm oder ein Prozess. Ebenso müssen wir die Handlungsweise und das Vorgehen des Programmes berücksichtigen.

Das Vorgehen des Programms, es drang in einzelne Rechner ein, vervielfältigte sich und verließ den Wirt dann wieder, auf der Suche nach einem neuen Wirt, spricht für einen Wurm. Ebenso zerstörte es weder Daten noch den Wirt, dies spricht wieder eher für einen Wurm als für ein Virus. Das das Programm aber keine verbundenen Segmente besaß, scheidet der Wurm als Möglichkeit aus. Bleibt also noch das Virus übrig. Wie bereits gesagt gibts es auch Viren die ihre Wirte nicht zerstören.

Betrachtet man aber nun einen Prozess als den Wirt so war, daß das Programm, auf jeden Fall ein Virus. Es drang in den Wirt über einen *Daemon*-Prozess ein, brachte ihn dazu einen viralen Prozess zu starten, der sich dann vervielfältigen und die Infektion verbreiten würde.

3 Chronologie der Ereignisse

3.1 Mittwoch 2.11 - Der Beginn

Um **9:30pm** wird eine private Workstation (*wombat.mit.edu*) des MIT Athena Projekts infiziert. Kurz darauf berichten die *Universitäten von Maryland und Berkeley* das sie ebenfalls betroffen sind.

3.2 Donnerstag 3.11 - schlechte Nachrichten

Immer mehr Institutionen, akademische und kommerzielle werden Opfer des Virus. Da das Virus sich offensichtlich mittels *Sendmail* Servern, die mit *debug*-Option kompiliert wurden, verbreitet werden die Mailserver größtenteils abgeschaltet. Nachrichten werden per *motd* (Message of the day) oder über Newsgroups im USENET verbreitet. Erste Beobachtungen berichten, daß das Virus VAX und SUN Binaries verschickt, und die Dateien *.rhosts* und *hosts.equiv* benutzt. Bis jetzt glaubt man das Rechner ohne *Sendmail*, bzw. ohne *Sendmail* mit *debug* sicher sind. An Universitäten bilden sich die ersten Gruppen, die versuchen der Infektion Herr zu werden. Patches und Bug Fixes werden eingereicht. Erste Gegenmassnahmen waren,

- *Sendmail* abzuschalten oder neu ohne *debug*-Option zu kompilieren
- Den Unix C-Compiler und Loader umbenennen, da das Virus sich noch kompilieren musste. Diese Maßnahme bot auch Schutz vor weiteren Angriffsarten des Virus, die noch unbekannt waren.

Es wird bekannt, daß das Virus sich auch *rsh* bedient. Es sucht in *.rhosts* und *hosts.equiv* nach Rechnern, um sie anzugreifen. Ebenso fällt auf das es besonders gerne *Gateways* angreift, die es in den Routingtabellen der infizierten Rechner findet.

Gegen 6:00pm beobachtet Ron Hoffman (MIT), wie das Virus mittels remote Login versucht auf einen Router zu gelangen. Der Versuch geht von einem Rechner aus, der bis dato als immun eingestuft worden ist, da der Mailer ohne *debug* Kommando läuft. Es scheint als wäre der Angriff durch den *fingerd* Daemon ermöglicht worden. Dieser Verdacht wird durch den Anruf einer Gruppe aus *Berkley* bestätigt.

Am Abend gelingt es ein VAX Binary und einen *Core Dump* des Virus sicherzustellen. Der Gegenangriff kann beginnen. Die Mailingliste *phague@purdue.edu* wird erstellt, über sie läuft die Kommunikation der Teams die gegen das Virus arbeiten. Um 10:18pm werden ein verbesserter *Sendmail*patch und neuer Quellcode für *fingerd*, mit *fgets* anstatt *gets* und *exit* anstatt *return*, gepostet.

Die Medien werden auf das Virus aufmerksam. Inzwischen sind geschätzte 10% der Rechner im Internet infiziert. Es wird entschieden das Virus zu *decompilieren*, und mittels *reverse Engineering* die Arbeitsweise des Virus im Detail zu verstehen.

3.3 Freitag 4.11 - Beobachtungen am Virus

Die einzelnen Routinen des Virus sind nun bekannt. Zu weiteren Tests wird eine speziell präparierte Testmaschine mit dem Virus infiziert. Diese Maschine war

- mit speziellen Log Mechanismen versehen, hauptsächlich packet Monitors
- markiert. D.h. sie hatte Verweise auf andere Maschinen, um zu verstehen wie das Virus seine nächsten Ziele auswählt und wie es dabei vorgeht.
- isoliert von allen Netzwerkanschlüssen.

Es kommt schließlich heraus, daß das Virus keine Daten zerstörte, es versuchte es nicht einmal. Vermutlich entkam das Virus während Testläufen und bevor es vollendet war.

3.4 Samstag 5.11 - Danach

Der Code des Virus wird unter Verschuß gehalten. Es gilt als zu gefährlich ihn zu veröffentlichen, da die Hemmschwelle zwei Zeilen schädlichen Code in das Virus einzubauen wesentlich niedriger ist als, das Virus neu zu schreiben. Auch verlangt es einiges mehr an Wissen aus den Berichten der Forscher über das Virus, das Virus *from scratch* zu schreiben und zu verändern als sich den Quellcode zu besorgen und diesen zu verändern.

4 Das Vorgehen des Virus

4.1 Wie drang es ein?

- Über Sendmail. Sendmail musste aber mit der *debug* Option compiliert sein, um den Angriff zu ermöglichen. Dies war bei den SunOS Releases per Default so.
- fingerd Mittels eines Buffer Overruns, konnte das Virus einen viralen Prozess starten, der die Infektion weitertragen würde. Nur VAX Architekturen waren von diesem Angriff betroffen. Für SUNs war dieser Angriff nicht implementiert.
- remote execution
 - rexec - einen Befehl auf einem entfernten Rechner ausführen
 - rsh - Kommandos an einen Kommandointerpreter auf einem entfernten Rechner weitergeben

4.2 Wen griff es an?

- Benutzeraccounts mit offensichtlichen Passwörtern, wie
 - keinem Passwort
 - dem Benutzernamen als Passwort
 - dem Benutzernamen an sich selbst angehängt
 - einem Spitznamen
 - dem Nachnamen
 - dem Vorname
- Accounts mit einem Passwort aus dem im Virus eingebauten Wörterbuch
- Accounts mit Passwörtern aus */usr/dict/words*
- Accounts die anderen Rechnern vertrauten nach dem *.rhosts* Prinzip

4.3 Welche Rechner griff es an?

- nur SUNs und VAXes
- Rechner in */etc/hosts.equiv*
- Rechner in *./rhosts*
- Rechner in *.forward* Dateien von geknackten Accounts
- Rechner in *.rhosts* Dateien von geknackten Accounts

- Rechner die als *Gateways* in den Routingtabellen auftauchten
- Rechner an den Enden von Point-to-Point Verbindungen
- Rechner mit zufällig geratenen Adressen

4.4 Was es nicht tat

- sich Zugang zu privilegierten Accounts (root) verschaffen
- Daten zerstören, oder versuchen Daten zu zerstören
- Zeitbomben hinterlassen
- das UUCP Protokoll benutzen
- es versuchte nie allgemein bekannte oder privilegierte Accounts zu knacken

5 Die Attacken im Detail

5.1 Sendmail Debug

Das Virus nutzte die *debug* Funktion des *Sendmail* Mailers aus. Der *debug* Modus ermöglicht es eine Mail zu verschicken mit einem Programm als Empfänger. Diese Feature übersteigt bei weitem das vorgesehene Design des *smtp*-Protokolls. Normalerweise legt man in *sendmail aliases* oder in *.forward* fest welches Programm aufgerufen werden soll wenn Post eintrifft, z.Bsp. Filterprogramme wie *procmail*, *spamfilter*. Dies ist aber normalerweise nicht für eingehende Verbindungen erlaubt. Bei dem Virus, war der Empfänger ein kleines Programm, das die Mailheader der Datei entfernte und es an einen Kommandointerpreter weiterreichte. Der Rumpf der Mail war ein Script, das ein C-Programm erzeugte, das die restlichen Virusmodule vom Quellrechner transferierte und diese dann compilierte, linkte und versuchte auszuführen. Es wurde jeweils versucht Binaries für SUNs und VAXes zum Laufen zu bringen, es wurde nicht versucht die Architektur des Wirtsrechners zu bestimmen. Auf anderen Architekturen konnten die Programme nicht ausgeführt werden, das Virus verbrauchte aber dennoch Rechenzeit und Speicher beim Linken der Binaries.

5.2 Finger Daemon Bug

Beim *finger daemon* Bug erzeugte das Virus einen Buffer Overflow eines Puffers der auf dem Stack lag. Der Überlauf des Puffers war möglich, weil die Implementierung von *fingerd* Bibliotheksfunktionen benutzte die ohne Bereichschecks (*range checking*) arbeiteten. Da der Puffer auf dem Stack lag, erlaubte es der Überlauf einen falschen Stack Frame zu erzeugen. Dies ermöglichte es Code auszuführen

wenn die Prozedur mit einem *return* zurückkehrte. Die fragliche Bibliotheksfunktion, stellte sich später als abwärtskompatible Routine heraus, die nach 1979 nicht mehr hätte benutzt werden sollen. Diese Attacke betraf nur Rechner der Architektur VAX mit 4.3BSD. Das Virus versuchte keine SUN spezifische Attacke auf *fingerd*. Die VAX *fingerd* Attacke schlug bei SUNs auch fehl.

5.3 Rexec und Passwörter

Bei seinen Angriffen bediente sich das Virus auch des *remote execution protocols*. Dieses erfordert einfach einen Benutzernamen und das passende Passwort, die über das Netzwerk übermittelt werden (die Übermittlung des Namens und Passworts erfolgt *plaintext* und nicht verschlüsselt). Das Virus benutzte nur Accounts und Passwörter die es schon getestet und für gültig befunden hatte. Im Allgemeinen gab es damals auf jedem UNIX eine für jeden lesbare Datei */etc/passwd*, die die Benutzernamen mit ihren verschlüsselten Passwörtern enthält. Das erleichterte die Suche nach gültigen Accounts natürlich ungemein. Mittels der Wörterbuchattacke, war es einfach Accounts zu knacken ohne in den Log Dateien des Systems aufzutreten. Das Virus verschlüsselte einfach zufällige Wörter aus seinem Wörterbuch und verglich sie mit den Passwörtern aus */etc/passwd*.

5.4 Rsh

Mittels Rsh (*remote shell execution*) versuchte das Virus auch entfernte Maschine ohne Passwörter anzugreifen. Rsh ist ähnlich zu rexec, wobei am anderen Ende aber ein Kommando Interpreter anstatt einer exec Funktion ist. Die Datei */etc/hosts.equiv* enthält eine Liste von Rechnern und Benutzer die ohne Passwortauthentifizierung Kommandos auf ihnen ausführen dürfen. Die Datei *.rhosts* bietet dasselbe aber auf Einzeluserbasis. War einmal ein Account auf dem lokalen Rechner geknackt, so fielen damit auch alle Rechner im Netzwerk für die der Benutzer in den beiden genannten Dateien gelistet war.

5.5 Informationsfluß

Mit der Erkenntnis, daß das Virus sich per *sendmail* verbreitete, war die erste Reaktion die Rechner vom Netz zu nehmen und alle Mailer herunterzufahren. Dies stellte sich als schwerer Fehler heraus, da es die Möglichkeit zur Kommunikation und Informationsaustausch um gegen das Virus anzukommen unmöglich machte. Da das Virus ja außer *sendmail* noch andere Möglichkeiten hatte sich zu verbreiten, machte es diese Maschinen zu sicheren Häfen für es. Es versetzte Rechner die tiefer in der Hierarchie lagen nur noch mehr in Gefahr, das es keine Möglichkeit gab, diese per Mail zu warnen.

6 Selbstschutzmechanismen des Virus

Um sich vor Entdeckung zu schützen, unternahm das Virus einiges um seine

Spuren zu verwischen. Es löschte seine Argumentenliste, nachdem es sie abgearbeitet hatte. So war es unmöglich mit dem Prozess Status Kommando (*ps*) zu sehen, wie und mit welchen Optionen es aufgerufen worden war. Es löschte ebenfalls das ausführende Binary, welches die Daten intakt aber unbenannt lassen würden. Nachdem das Programm sich beendete sah es aus als wäre es nie da gewesen. Es benutzte auch Funktionen die seine Ressourcen beschränkten. Damit wurde ein Core Dump des Virus verhindert.

Tarnung Es wurde auf den Namen *sh* übersetzt, derselbe Name den die Bourne Shell (Kommandointerpreter) benutzt. Auf einem Unix laufen immer wieder zahlreiche Prozesse unter diesem Namen, um z. Bsp. Scripte auszuführen oder wenn sie von Scripten aufgerufen werden. Nichts was verdächtig erscheinen würde. Selbst erfahrene SysOps würden sich bei mehreren *sh* Prozessen nichts denken.

Das Virus *forkte* auch in einen Elternprozess und einen Kindprozess. Dies tat es ungefähr alle 3 Minuten. Der Elternprozess beendete sich, und der Kindprozess fuhr an der exakt gleichen Stelle fort. Dies hatte den Effekt das der Prozess wieder aufgefrischt, verjüngt wurde. Da der neu erzeugte Kindprozess mit neuen Ressourcen (CPU Zeit, Speicherverbrauch) startete, fiel er nicht ohne weiteres auf. Ebenso lief das Virus immer nur sehr kurz unter derselben PID (*Process ID*), was es schwierig machte es zu fassen, selbst wenn man es bemerkte.

Alle Strings die vom Programm benutzt wurden, waren mittels XOR mit einer Konstanten (#81) verschlüsselt. Man konnte also dem Binary nicht einfach ansehen, welche Dateien es öffnete. Dies behinderte es natürlich das Virus zu verstehen, aber nicht für allzu lang.

7 Designfehler

Das Virus war aber auch nicht perfekt. Es selbst hatte aber auch einige Designfehler und Bugs.

7.1 Mehrfachinfizierung

Der Codeteil der die Mehrfachinfizierung verhindern sollte, beherbergte einige Fehler. Diese stellten sich als kritisch für das Virus heraus, da eine Mehrfachinfektion die Last (load) einer Maschine hinaufsetze, was dazu führte, daß das Virus erkannt und bekämpft wurde.

Der Code hatte mehrere Timingfehler, die ihn nicht gerade stabil machten. Die Routinen die eine Mehrfachinfektion überprüften, stürzten oft mit Fehlern ab. Dies geschah dann, wenn

- mehrere Viruse gleichzeitig eine noch nicht infizierte Maschine infizierten. In diesem Fall würden alle nach Lauschern suchen, keiner würde einen finden, sie alle würden versuchen einer zu werden, einer würde es schaffen, die anderen versagen und aufgeben. Und wären bei zukünftige Checks nicht zu bemerken.
- mehrere Viruse gleichzeitig starten würden, während bei bereits ein Virus ausgeführt wird. Sollte der Erste gegen das bereits lauschende Virus gewinnen, hätten andere neu-startende den Verlierer bemerkt, die Verbindung beendet, was ihnen erlaubt hätte fortzufahren.

7.2 ungenutzte Schwachstellen

Folgende Schwachstellen wurden vom Virus nicht angegagen:

- Auf der Suche nach Rechnern, machte es keine Zonentransfers aus den Internet Domain Name Servern um weitere gültige Hostnamen zu finden. Diese Einträge beinhalten zum Großteil auch den Typ des Rechners, somit hätte die Suche nach passenden Rechnern noch weiter eingeschränkt werden können.
- Es griff nicht immer beide Typen von Rechnern an. Wenn die *fingerd* Attacke für VAX fehlschlug hätte es SUN Attacken probieren können, was aber nicht implementiert war.
- Es versuchte nie die Kontrolle über privilegierte Accounts (root) zu erhalten.

8 Gegenmaßnahmen

Es gab einige Maßnahmen um sich gegen das Virus zu schützen. Einige waren recht simpel und ohne große Einschränkungen für die Benutzer der Systeme, andere brachten jedoch erhebliche Einschnitte mit sich.

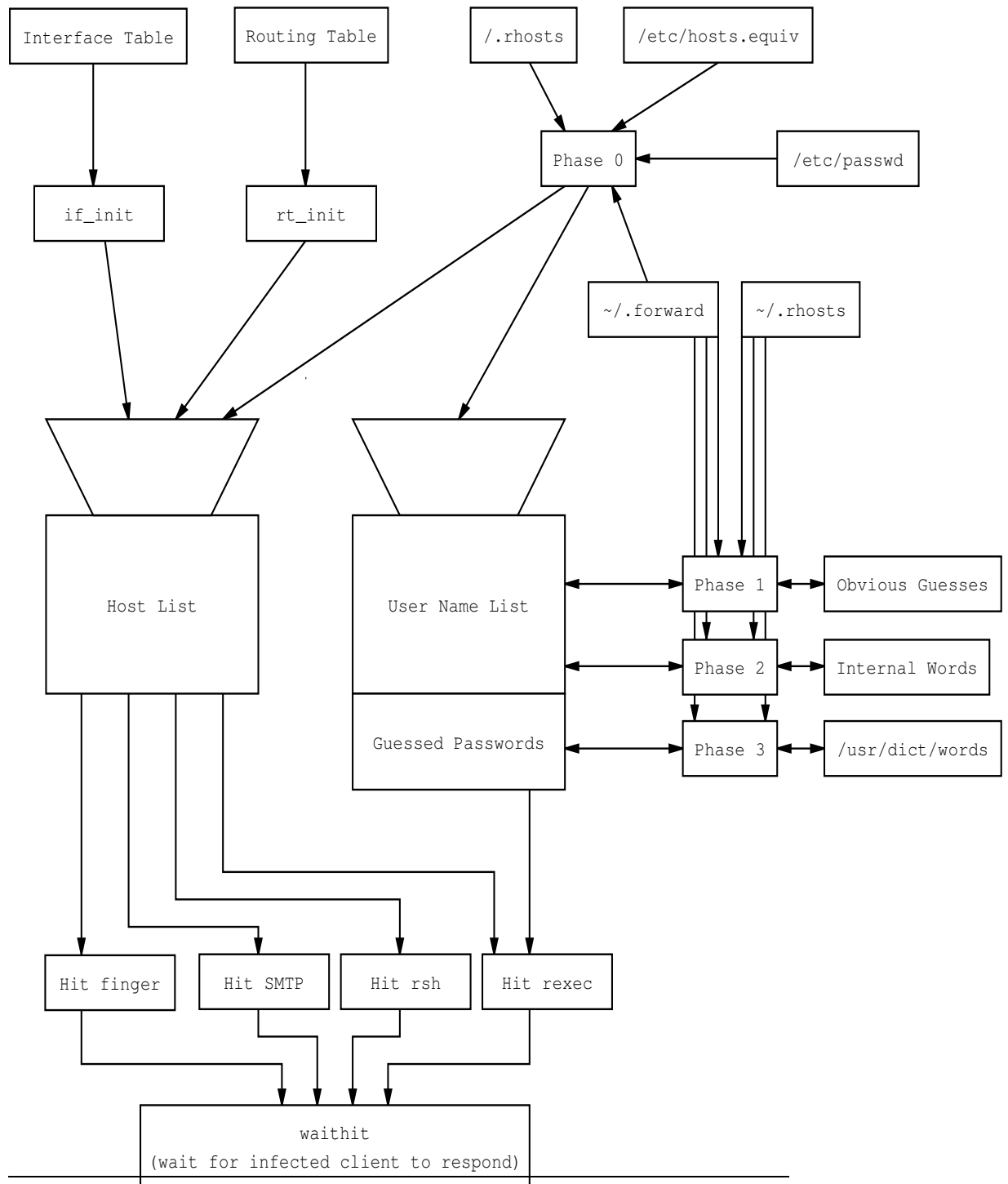
- Komplette Isolierung des Netzwerks vom Internet.
- Abschalten des Mailers, war für die Benutzer und Rechner tiefer in der Hierarchie nicht sehr vorteilhaft, da es den Informationsfluß zum Erliegen brachte. War aber einfach zu implentieren, aber dennoch ineffektiv.
- Patches von *sendmail*, war nur sinnvoll im Zusammenhang mit anderen Fixes. Patches und Anleitungen lagen bereit.
- Herunterfahren des *finger* Daemons. War nur sinnvoll wenn alle anderen Sicherheitslöcher gestopft wurden.
- Die Bugs im *fingerd* Source Code beseitigen und *fingerd* neu installieren.

- `mkdir /usr/tmp/sh` war einfach und effektiv um das Virus daran zu hindern sich weiterzubreiten.
- Den UNIX C-Compiler und Linker umbenennen. Dies war effektiv um das Virus zu stoppen, da es sich compilieren musste um sich weiterzubreiten.

Nach dem das Virus analysiert und seine Attacken bekannt waren, wurde ein Tool geschrieben das die Passwortattacke nachahmte. Dies war nützlich um Accounts mit schwachen Passwörtern ausfindig zu machen und die Benutzer darauf hinzuweisen, diese zu ändern. Ebenso wurde das `passwd` Kommando verbessert, so daß es nur noch Passwörter akzeptierte die Tests gegenüber einer Liste standhielten, die auch die Passwörter des Virus enthielt.

9 Die Struktur des Virus

9.1 Strukturdiagramm



9.2 Die Hauptroutinen

Der Kern des Virus ist ein Paar binärer Module, eines für die VAX Architektur, eines für SUNs. Die Module sind linkbar und haben deshalb Namenslisten für die internen Prozeduren. Überraschend war das die Namen beigefügt waren und das sie wichtig sind. Schon mit einfachen Techniken, wie zufälligen Prozedurnamen, hätte man die Hinweise auf die Funktionsweise des Virus verringern können.

main Das main Modul, starting point jedes C Programms. Hier wird einiges initialisiert und die Argumentenliste abgearbeitet. Danach geht es in die Schleife, die eigentlich die Hauptarbeit macht.

doit Routine Der Hauptteil des Programms. Hier initialisiert das Programm den Zufallsgenerator und speichert die Zeit, um später zu messen wie lange das Virus läuft. Danach versucht es *hg* aufzurufen. Schlägt dies fehl, versucht es *hl* aufzurufen, schlägt dies ebenso fehl so versucht es *ha* aufzurufen. Danach prüft das Programm ob bereits schon eine Kopie des Virus auf dem Rechner läuft. Dieser Teil des Codes hat einige Fehler. So wird nur in einem von sieben Fällen wirklich versucht zu ermitteln ob der Rechner bereits infiziert ist.

Eine Endlosschleife enthält die Hauptkomponenten des Programms. Die *crack-some* Routine wird aufgerufen, sie versucht Rechner zu finden und in diese einzubrechen. Daraufhin wartet es 30 sec, und prüft ob andere Viruse versuchen einzubrechen, danach versucht es wieder auf anderen Rechnern einzubrechen. Nach den Angriffen forkt das Virus. Der Elternprozess stirbt, der Kindprozess hat alle Informationen des Elternprozess und eine neue PID (*process id*) sowie frische Ressourcen (CPU TIME, MEM USAGE).

Als nächstes suchen die *hg*, *hl* und *ha* Routinen nach neuen zu infizierenden Rechnern. Das Programm ruht dann für 2 Minuten und prüft dann ob es schon länger als 12 h läuft.

Schliesslich bevor es sich wiederholt, prüft es die globale Variable *pleasequit*. Falls diese gesetzt ist und es schon mehr als 10 Wörter aus seinem Wörterbuch gegen Passwörter auf dem System getestet hat, beendet es sich. Trotz allem bringt es reichlich wenig gegen das Virus, in den System Bibliotheken die Variable *pleasequit* zu setzen.

9.2.1 Cracking Routinen

cracksome Routine Diese Routine entscheidet welche Strategie als Nächstes benutzt wird.

Phase 0 Als erstes lass sich das Virus durch */etc/hosts.equiv*, */.rhosts*, */home/\$USER/.rhosts* und */home/\$USER/.forward*, um weitere Rechner zu finden die es angreifen konnte. Ebenso lass es */etc/passwd* ein, speicherte die Accountnamen, die Userinfor-

mationen und die verschlüsselten Passwörter in einer internen Liste ab. War dies abgeschlossen ging es weiter mit Phase 1.

Phase 1 Nun wurde versucht 50 Passwörter auf dem lokalen System zu entschlüsseln. Zuerst wurde es mit keinem Passwort versucht, dann der Benutzername, der Benutzername an sich selbst angehängt, Daten aus den Userinformation aus */etc/passwd* oder der Nachname rückwärts.

Phase 2 nahm das eingebaute kodierte Wörterbuch und mischte es durch. Danach dekodierte es jedes Wort, und testete es gegen alle gesammelten Passwörter.

Phase 3 lass die Datei */usr/dict/words* ein. Jedes Wort aus dieser Datei wurde mit jedem verschlüsseltem Passwort verglichen. Es wurden auch alle Kombination von Wörter mit Groß/Kleinbuchstaben am Wortanfang ausprobiert. Nach Ende dieser Phase wurde der Phasenzähler auf 4 erhöht, obwohl keine Phase 4 zur Verfügung stand.

h Routinen Die h Routinen sind einige Routinen die andere Rechner für Angriffe suchen. *hg* zum Beispiel scant die Routing Tabelle und merkt sich alle Gateways. Danach versucht es diverse Attacken mittels *rsh*, *finger* oder *stmp*. Nach der ersten erfolgreichen Attacke beendet sich *hg*.

9.2.2 Angriffsroutinen

hu1 Wird in Phase 1 und 3 von der *cracksome* Routine aufgerufen. Nachdem einmal ein passendes Passwort für einen Account gefunden wurde, wird als nächstes ein Rechner aus den Dateien *.rhosts*, *.forward* des geknackten Accounts ausgesucht. Danach wird versucht sich mittels *rexec* und der *ID* des kompromitierten Account mit dem *rexec*-Server des Ziels zu verbinden. Bei Erfolg wird mit *rsh* eine Bourne Shell auf dem Zielhost gestartet, welche benutzt wird das *ll.c* Programm zu transferieren und zu compilieren.

Hit SMTP Diese Routine öffnet eine Verbindung zum TCP Port 25 des Ziels und benutzt diese um den *Sendmail* Bug auszunutzen. Es versucht mittels der *debug* Option, *Sendmail* zum Ausführen eines Kommandos zu bringen, welches das *ll.c* Programm innerhalb einer Mail verschickt.

Hit finger Die "*hit finger*" Routine versucht eine Verbindung auf Port 79 des Zielhosts aufzubauen. Danach wird ein *magic packet* erzeugt, das aus folgendem besteht:

- 400 Byte *nop* Anweisungen für VAXes
- ein Stück Code das eine Bourne Shell ausführt

- ein Stack Frame, mit einer Return Adresse die hoffentlich auf eine Adresse innerhalb des Codes zeigt.

Dieser Code war reiner VAX Code, der Stack Frame war ebenso nur für VAX Architekturen, in der falschen Ordnung für eine SUN. Deshalb konnte dieser Code nur auf VAX Maschinen den Bug des *fingerd* Daemons ausnutzen, obwohl dieser ebenfalls auch die SUNs betraff.

Hit Rsh Diese Routine versucht *rsh* auf dem Opfer zu starten. Es werden drei verschiedenen Namen für das *rsh* versucht.

- */usr/ucb/rsh*
- */usr/bin/rsh*
- */bin/rsh*

Gelingt einer der drei, wird die Verbindung benutzt um auf dem Opfer das 11.c Programm auszuführen. Gelingt es nicht innerhalb 30 sec. eine Verbindung zu erstellen, wird der Prozess beendet. Dabei ist wichtig das bei dieser Art der Attacke kein Account benötigt wird. Dies geht weil der Benutzer unter dem das Virus gestartet wurde, ebenso einen Account auf dem Zielrechner hat, und dieser vertraut eben dem Rechner und Benutzer, von dem der Angriff ausgeht.

Hit rexec Die *hit rexec* Routine funktioniert nach dem selben Schema wie *hit rsh*, nur das hierbei nur ein einzelnes Kommando ausgeführt wird. Die Routine sendet Benutzernamen und Passwort und startet dann als Kommando */bin/sh*.

11.c Programm Das 11.c Programm ist der Teil des Virus, denn jede Angriffsroutine benutzte um die restlichen Virusteile zwischen den Rechnern zu transferieren. Es selbst ist stabil und portabel geschrieben, und lief selbst auf Rechnern die weder SUNs noch VAXes waren. Natürlich waren die transferierten Binarys auf diesen Systemen nicht lauffähig.

Als erstes löscht das Programm, das Programm von dem es aufgerufen wurde. Dann überprüft es ob ihm drei Argumente übergeben wurden, ist dem nicht so beendet es sich. Anschließend versucht es zu *forken*, schlägt dies fehl beendet es sich. Bei Erfolg beendet sich der Elternprozess, was es nicht mehr ermöglicht vom Kindprozess aus eine Verbindung zur Quelle der Infektion herzustellen. Als nächstes öffnet es eine TCP Verbindung, zu der Adresse die ihm in seinem ersten Argument übergeben wurde und auf den Port, welcher das zweite Argument war. Danach sendet es sein drittes Argument (magic number) über die Verbindung. Danach beginnt die eigentlich Transaktion und Installation des Virus auf dem Rechner.

9.3 Das eingebaute Wörterbuch

Das Virus verfügte über ein 432 Wörter fassendes internes Wörterbuch. Diese war ebenso wie alle Strings im Programmcode kodiert. Hier ein kleiner Auszug:

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 comrades 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

10 Schlußfolgerungen

Nachdem der Angriff des Virus überstanden war, und man ihn analysiert hatte, wurde klar wo die Schwachstellen im System saßen und wie man sich richtig bei einem erneuten Angriff zu verhalten hätte.

- Kommunikation ist immens wichtig. Rechner die sich präventiv vom Netz isolierten, schaden nicht nur sich selbst sondern auch dem Rest des Systems.

Erstens können sie nicht mehr über ihre Erfahrungen mit dem Virus berichten, sondern sind auch nicht in der Lage selbst Hilfe zu erhalten, oder bug fixes über das Netz zu beziehen. Ferner war jeder Rechner der von einem abgeschalteten Gateway oder Mailserver abhängig war, nur noch mehr verwundbar, da er keine Möglichkeiten hatten, Nachrichten von außerhalb zu bekommen. Die Rechner des *MIT* und von *Berkley* gingen nie komplett vom Netzwerk, und konnten so trotz allem effektiv zusammen arbeiten.

- Die Verfügbarkeit des Source Codes war ebenfalls sehr wichtig. Nur wer den UNIX Code hatte, konnte schnell und sicher die Fehler in ihm entdecken die das Virus ausnutzte.
- Eine Datei die sowohl Benutzernamen und auch die verschlüsselten Passwörter enthält und auch noch world readable ist, ist ein immenses Sicherheitsrisiko, da dies erst die Wörterbuchattacke ermöglichte. Inzwischen gibt es für Passwörter die Datei */etc/shadow* die nur noch vom Benutzer *root* gelesen und geschrieben werden kann.
- Log Files sind wichtig! Mit Hilfe der logs war es einfacher die Quelle der Infektion zu erkennen und zurückzuverfolgen.
- Gegenmaßnahmen müssen am Rechner selbst nicht auf Netzwerkebene ansetzen. Während der ganzen Zeit der Virusattacke funktionierte das Netzwerk einwandfrei, das Problem trat erst auf den Hostcomputern auf.
- Generelle Sicherheitsaspekte, wie sichere Passwörter. Nicht einfach nur den Nachnamen, Vornamen oder Spitznamen verwenden. Sondern Kombinationen von Buchstaben, Zahlen, Sonderzeichen und Groß - und Kleinschreibung. Jeder Benutzer sollte seine Daten schützen, wozu gibt es das Rechtssystem auf einem UNIX. Und niemand sollte mehr Rechte haben als die, die er für seine Arbeit an einem Rechner wirklich benötigt.
- Rsh. Das Modell mit dem Rsh arbeitet, eine Liste von Hosts und Benutzern denen Zugang ohne Passwortauthentifizierung gewährt wird, ist zwar bequem aber auch sehr sehr gefährlich. Selbst bei Authentifizierung per Passwort, wird dieses Plaintext über das Netzwerk verschickt. Inzwischen gibts es als sicherere Alternative zu *Rsh* die Secure Shell *Ssh*. Es werden alle Kommandos wie *rsh*, *rcp* ... neu, aber dabei sicherer, implementiert. Passwörter werden nur noch verschlüsselt übertragen.
- Heterogenität des Netzes. Obwohl das Virus die zwei meist verbreitetsten Rechnerarchitekturen und das populärste Betriebssystem der damaligen Zeit angriff, waren die meisten Rechner jedoch ausser Gefahr. Wie in der Biologie sind Monokulturen oder homogene Systeme immer anfälliger für Angriffe als heterogene Systeme, da sich der Erreger nur auf einem kleinen Teil des ganzen Systems verbreiten kann.

11 Literaturverzeichnis

Literatur

- [1] MARK W. EICHIN AND JON A. ROCHLIS.
With Microscope and Tweezers
An Analysis of the Internet Virus of November 1988.
<http://www.mit.edu/people/eichin/virus/main.html>.
- [2] ELLEN SIEVER AND O'REILLY ASSOCIATES 2ND EDITION
Linux in a Nutshell. O'Reilly 1999.
- [3] OLAF KIRCH & TERRY DAWSON
Linux Network Administrator Guide 2nd Edition
O'Reilly 2000