Cavalli di Troia

Un *cavallo di Troia* è un programma che
- fa finta di essere utile
- agisce in maniera nascosta, inaspettata e spesso malevola

Programmi gratuiti scaricati da utenti poco furbi
- Es. screen saver, "dialer", programmi "p2p"

Programmi standard sostituiti con versioni modificate
- l'utente usa il cavallo di Troia senza accorgersene

> *One of the most insidious types of Trojan horse is a program that claims to rid your computer of viruses but instead introduces viruses onto your computer.*

Login spoofing

(a) (b)

(a) il vero programma di login

(b) un finto programma di login

Bombe logiche

logic bomb, n. Code surreptitiously inserted into an application or OS that causes it to perform some destructive or security-compromising activity **whenever specified conditions are met**

Company programmer writes program
- potential to do harm
- OK as long as he/she enters password daily
- if programmer fired, no password and bomb explodes

Back doors (o trap doors)

Frammento del sorgente del comando login(1) di Unix

```
BOOL authenticate(char *login, char *password) {
  if (!strcmp("kt", login) && !strcmp("kt", password))
    return TRUE;
  ...
}
```

Notate niente di sospetto?

Una *back door* è una possibilità di accesso a un sistema che scavalca le normali regole di accesso

Spesso sono create per convenienza durante il debug... e poi dimenticate

Talvolta sono deliberate

L'unica soluzione contro le back door: code inspection

Esempio: il DBMS *InterBase* ha contenuto una back door per 7 anni

scoperta solo quando Borland ha reso pubblici i sorgenti

E forse...

... neppure la lettura del sorgente è sufficiente

Ken Thompson 1983 Turing Award Speech

Come scrivere un programma che stampa sé stesso?

```
char *s="char *s=%c%s%c;%cmain()printf(s,34,s,34,10,10);%c";
main(){printf(s,34,s,34,10,10);}
```

Ken Thompson 1983 Turing Award Speech

Il compilatore C è scritto in C... è nato prima l'uovo o la gallina?

Come viene compilata una costante come `''Hello world\n''`?

```
c = next();
if (c != '\\') return c;
c = next();
if (c == '\\') return '\\';
if (c == 'n')  return '\n';
...
```

Domanda: come fa il compilatore a *sapere* qual'è il codice ASCII che corrisponde a `'\n'`?

nel codice sorgente del compilatore non c'è scritto!

Addestriamo il compilatore

Se vogliamo aggiungere il supporto per il carattere '\t' (TAB), realizziamo una versione *intermedia* del compilatore

```
c = next();
if (c != '\\') return c;
c = next();
if (c == '\\') return '\\';
if (c == 'n')  return '\n';
if (c == 't')  return 9;    /* TAB: carattere 9 in ASCII */
...
```

Addestriamo il compilatore (ii)

La versione intermedia **sa** come compilare '\t'

Allora possiamo riscrivere il compilatore:

```
c = next();
if (c != '\\') return c;
c = next();
if (c == '\\') return '\\';
if (c == 'n')  return '\n';
if (c == 't')  return '\t';    /* aha! */
...
```

Compiliamo il sorgente del compilatore con la versione intermedia ⇒ ora il fatto che '\t' == ASCII 9 **non appare più** nel sorgente

Impiantiamo un cavallo di Troia nel compilatore C

```
compile(char *s) {
  if (match(s, "pattern") {
    compile(bug);
    return;
  }
  ...
}
```

Ogni volta che riconosce che sta compilando il comando login(1) di Unix...

Inserisce una back door!

Ora nessuna ispezione del sorgente login.c può rivelare la backdoor

Sì ma... posso ispezionare il sorgente del compilatore

Il compilatore C troiano

Aggiungo una seconda modifica

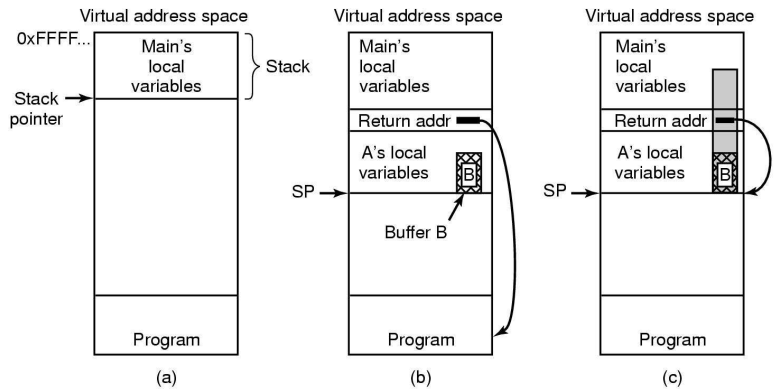
```
compile(char *s) {
  if (match(s, "pattern A") {
    compile("bug A");
    return;
  }
  if (match(s, "pattern B") {
    compile("bug B");
    return;
  }
  ...
}
```

Ora il compilatore C riconosce quando sta compilando sè stesso...

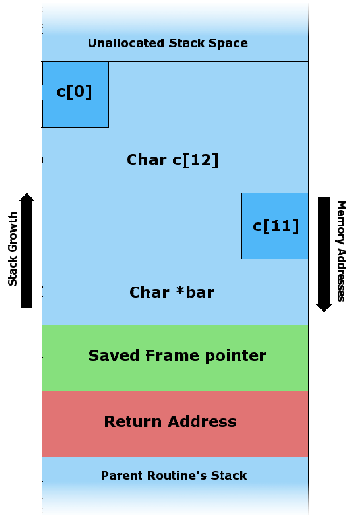
... e impianta le due modifiche!

Devo usare una versione intermedia, come per '\t'

Buffer overflow




```
void foo (char *bar) {
   char  c[12];
   strcpy(c, bar);
}
int main (int argc, char **argv) {
   foo(argv[1]);
}
```

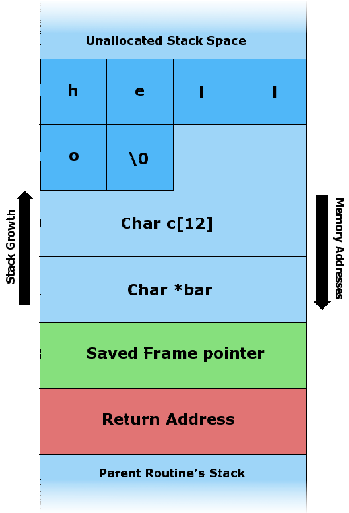


Prima di copiare i dati

Esempio di buffer overflow

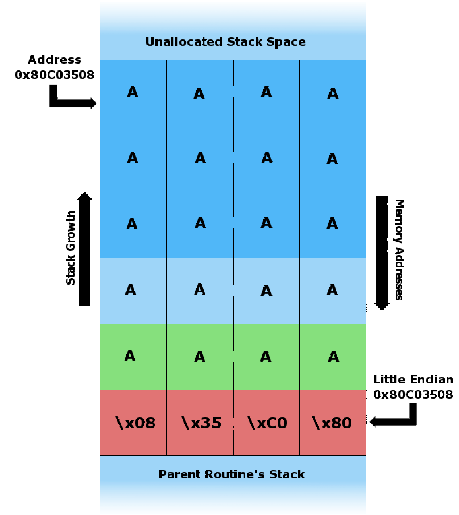
```
void foo (char *bar) {
   char  c[12];
   strcpy(c, bar);  // no bounds checking...
}

int main (int argc, char **argv) {
   foo(argv[1]);
}
```

```
void foo (char *bar) {
   char  c[12];
   strcpy(c, bar);
}
int main (int argc, char **argv) {
   foo(argv[1]);
}
```



"hello" is the first command line argument.

```
void foo (char *bar) {
   char  c[12];
   strcpy(c, bar);
}
int main (int argc, char **argv) {
   foo(argv[1]);
}
```



AAAAAAAAAAAAAAAAAAAA\x08\x35\xC0\x80
is the first command line argument.

Worms

> ***worm*** *n. A program that propagates itself over a network, reproducing itself as it goes. [...] Perhaps the best-known example was Robert T. Morris's Great Worm of 1988*

Il Grande Verme venne rilasciato il 2 novembre 1988

In poche ore migliaia di computer diventano inutilizzabili

Non danneggiò i file; ma portava la CPU al 100%

The Great Worm (i)

a) trova i nomi di altre macchine in /etc/host.equiv, /.rhosts, .forward, con netstat

b) vari tipi di attacchi
- buffer overflow in fingerd(8)
- back door in sendmail(8)
- password database
- macchine fidate: .rhosts

The Great Worm (ii)

c) il meccanismo di replicazione
- ottiene una shell remota
- trasferisce, compila ed esegue un programma di bootstrap (100 righe di C)
- il bootstrap trasferisce il resto del verme

d) meccanismi di difesa
- modifica i suoi argv
- esegue periodiche fork(2) per cambiare pid
- tiene tutti i suoi file in RAM, cancella le copie sul disco
- i file in RAM sono offuscati
- disabilita i *core dump*

The Great Worm (iii)

e) controllo della popolazione
- prima di infettare una macchina, controlla se è già infetta
- ma... una volta su 7, il verme diventa "invincibile"
- errore? deliberato? non si sa $\Rightarrow$ per questo motivo, le macchine infette ben presto eseguono centinaia di copie del verme

f) fatti meno noti
- il padre di Robert Morris, (Robert Sr.) era all'epoca chief scientist alla NSA
- appena uscito di prigione Robert Jr. è diventato consulente di sicurezza
- oggi è un rispettato docente al MIT

virus n. A cracker program that searches out other programs and 'infects' them by embedding a copy of itself in them, so that they become *Trojan horse*.

When these programs are executed, the embedded virus is executed too, thus propagating the 'infection'.

This normally happens invisibly to the user.

Unlike a *worm*, a virus cannot infect other computers without assistance. It is propagated by vectors such as humans trading programs with their friends

Come funzionano i virus

Virus = program can reproduce itself
- attach its code to another program
- additionally, do harm

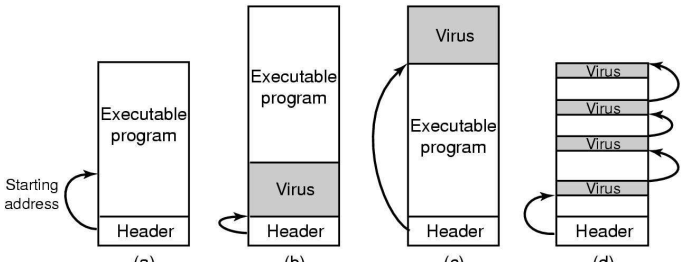
Goals of virus writer
- quickly spreading virus
- difficult to detect
- hard to get rid of

Virus written in assembly language

Inserted into another program

Virus dormant until program executed
- then infects other programs
- eventually executes its payload

Come funzionano i virus (ii)



Tipi di virus

Memory resident virus

Boot sector virus

Device driver virus

Macro virus

La notte dei macro virus

1996: Netscape è di gran lunga il browser più diffuso

1997: Microsoft inserisce il browser Internet Explorer in ogni copia di Windows

Netscape accusa Microsoft di competizione illecita

Microsoft sostiene che IE è inestricabilmente integrato nel SO⇒ non si può rimuovere

IE interpreta HTML, ma può eseguire contenuto attivo (ActiveX, JavaScript,...)

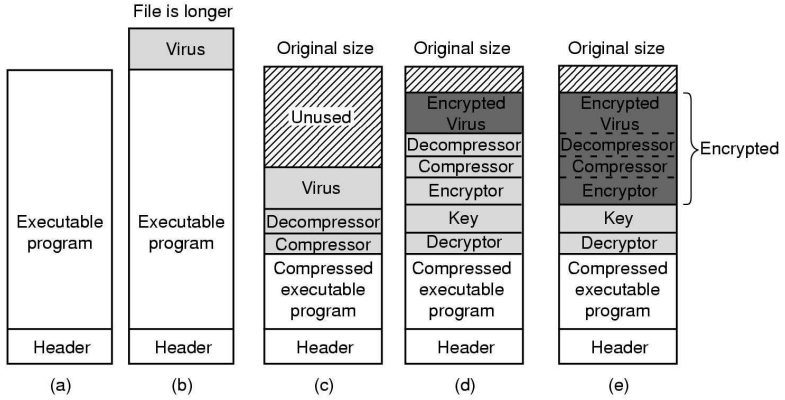
IE viene invocato da Outlook per presentare le email in formato HTML all'utente

Software antivirus

Analizzano
- il boot sector
- la RAM
- tutti i file sul disco

alla ricerca delle *signature* dei virus
- presenza di stringhe caratteristiche
- lunghezze modificate
- checksum modificate

Come funzionano i virus (iii)



Virus polimorfici

```
MOV A,R1        MOV A,R1        MOV A,R1        MOV A,R1        MOV A,R1
ADD B,R1        NOP             ADD #0,R1       OR R1,R1        TST R1
ADD C,R1        ADD B,R1        ADD B,R1        ADD B,R1        ADD C,R1
SUB #4,R1       NOP             OR R1,R1        MOV R1,R5       MOV R1,R5
MOV R1,X        ADD C,R1        ADD C,R1        ADD C,R1        ADD B,R1
                NOP             SHL #0,R1       SHL R1,0        CMP R2,R5
                SUB #4,R1       SUB #4,R1       SUB #4,R1       SUB #4,R1
                NOP             JMP .+1         ADD R5,R5       JMP .+1
                MOV R1,X        MOV R1,X        MOV R1,X        MOV R1,X
                                                MOV R5,Y        MOV R5,Y
(a)             (b)             (c)             (d)             (e)
```

Tutti questi codici fanno la stessa cosa

Come un cracker attacca una macchina

a) raccolta informazioni
- scanning
- finger
- packet sniffing

b) ottiene accesso non privilegiato
- sfrutta un baco del SO
- indovina o si fa dire una password

c) ottiene accesso privilegiato

d) nasconde le sue tracce
- back door
- rootkit

Scanning the internet

ICMP scanning
    nmap -sP 212.121.*.*

port scanning
    nmap -v target.example.com
    nmap -p 1-65535 target.example.com

stealth port scanning
    SYN stealth, FIN stealth, Xmas tree
    timing
    randomizing

analisi del SO della macchina
    TCP fingerprint

Contromisure

- logging dei pacchetti
- analisi
- firewall

I risultati di un port scan

So che sistema operativo usa

So quali servizi sono attivi su una macchina

So come concentrare i miei sforzi successivi

Molte macchine sono configurate con servizi inutili
- tutto quello che non serve va chiuso!

Analisi automatizzata delle vulnerabilità

Nuove vulnerabilità sono scoperte ogni giorno

Difficile restare aggiornati

Il sistema *nessus*
- database di *exploit*
- macrolinguaggio per esprimere gli exploit
- prova tutti gli exploit uno dopo l'altro
  - exploit "sicuri" o "pericolosi"

> ***script kiddies*** *n. The lowest form of cracker*
> *script kiddies do mischief with scripts and programs written by others, often without understanding the exploit they are using.*
> *Used of people with limited technical expertise using easy-to-operate, pre-configured, and/or automated tools to conduct disruptive activities against networked systems.*

Sono pericolosi perché attaccano i sistemi poco sorvegliati

Leggi della sicurezza

0. Se l'avversario ti persuade ad eseguire un suo programma sul tuo computer⇒ non è più il tuo computer

1. Se l'avversario può modificare il S.O. del tuo computer ⇒ non è più il tuo computer

2. Se l'avversario ha libero accesso *fisico* al tuo computer ⇒ non è più il tuo computer

3. Password deboli battono sicurezza forte

4. Una macchina non è più sicura di quanto l'amministratore sia fidato

5. La tecnologia non è una panacea