

# IL PROBLEMA DELLA SICUREZZA INFORMATICA

## Distinzione

La sicurezza informatica consiste in due diversi ordini di problemi:

- ✚ il problema generale – che chiameremo di **sicurezza** – di assicurarsi che i file (o in genere i dati utilizzati in un sistema informatico) non siano letti o manomessi da persone non autorizzate; questo problema coinvolge aspetti tecnici, manageriali, legali e politici, oltre che, ovviamente, di privacy;
- ✚ il problema degli specifici meccanismi e regole – che chiameremo di **protezione** – che il sistema operativo usa per salvaguardare la sicurezza delle informazioni contenute nel computer.

Il confine tra “sicurezza” e “protezione” non è ben chiaro.

La **sicurezza** ha molti aspetti: due dei più importanti sono la **perdita di dati** (per **eventi accidentali** quali incendi, terremoti, tumulti o ... topi; per **errori hardware e software**; per **errori umani** quali immissione di dati non corretti, smarrimento di supporti di memorizzazione, errata esecuzione di programmi) e le **intrusioni** (**passive**, quando ascrivibili a soggetti che vogliono “semplicemente” leggere dati che non sono abilitati a leggere; **attive**, quando sono effettuate da individui più maliziosi, che intendono apportare modifiche non autorizzate ai dati).

La **protezione** indica i meccanismi, definiti dal sistema operativo, per controllare l’accesso alle risorse da parte di programmi, processi o utenti. Tali meccanismi consistono nei mezzi per specificare i controlli e in quelli per applicare i controlli stessi. La protezione va fornita per molte ragioni: impedire violazioni casuali o, peggio, intenzionali delle restrizioni di accesso a specifiche risorse.

## IL PROBLEMA DELLA PROTEZIONE: Meccanismi e Regole

In generale è bene distinguere fra:

➡ **regole di protezione**: cosa va fatto;

➡ **meccanismi di protezione**: come va fatto, cioè accorgimenti e meccanismi specifici usati in un sistema operativo per garantire la sicurezza.

Un particolare e molto diffuso meccanismo di protezione è il **dominio di protezione**.

Un dominio di protezione, o semplicemente dominio, è un insieme di coppie (oggetto, diritti) dove:

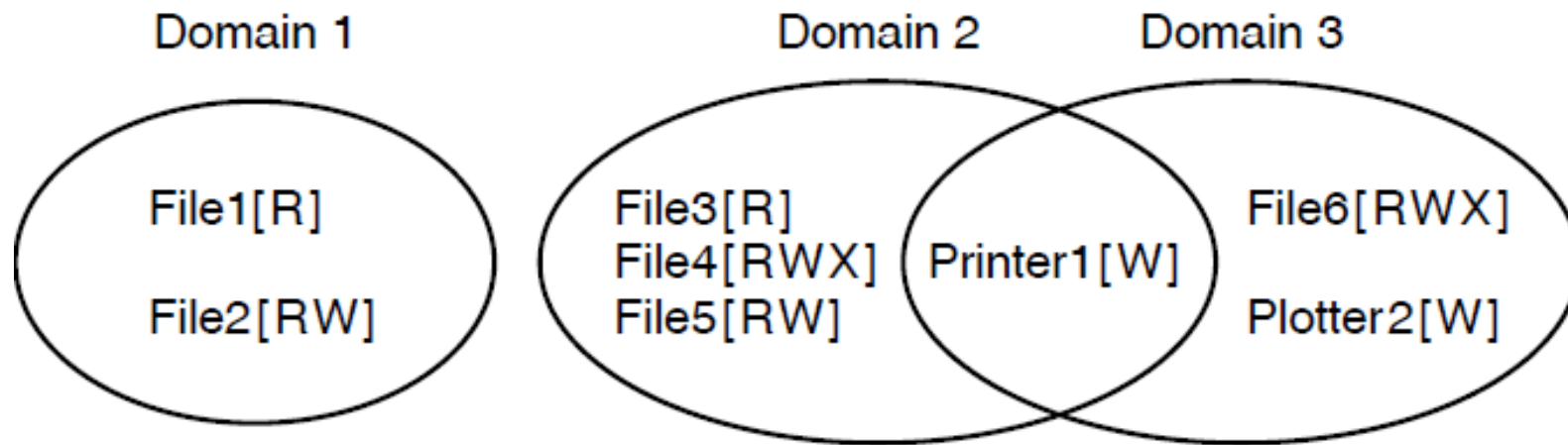
👍 **un oggetto**: indica una risorsa hardware (es.: CPU, Segmento di memoria, stampante, disco, ecc.) o software (es.: file, programma, semaforo, ecc.);

👍 **i diritti**: indicano le operazioni che possono essere effettuate sull'oggetto.

***Quindi ogni dominio – cioè 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).

## REALIZZAZIONE DI UN DOMINIO

Un dominio può essere realizzato in vari modi:

- ➡ ***ogni utente può essere un dominio***: in questo caso l'insieme di oggetti a cui si può accedere dipende dall'identità dell'utente; *il cambio di dominio* si ha quando si passa da un utente ad un altro;
- ➡ ***ogni processo può essere un dominio***: in questo caso il gruppo di oggetti a cui si può accedere dipende dall'identità del processo; il cambio di dominio si ha quando un processo attiva un altro processo e attende una risposta;
- ➡ ***ogni procedura può essere un dominio***: in questo caso l'insieme di oggetti a cui si può accedere corrisponde alle variabili locali definite nella procedura; si ha un cambio di dominio quando viene chiamata una procedura.

## MATRICI DI PROTEZIONE: LA MATRICE DI ACCESSO

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

		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

Anche *i cambi di dominio* dei processi possono essere modellati tramite matrici di protezione in cui la possibilità di *switch* da un dominio ad un altro viene modellata aggiungendo alle colonne i domini ed evidenziando così l'eventuale diritto di accesso da un dominio ad un altro.

		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			

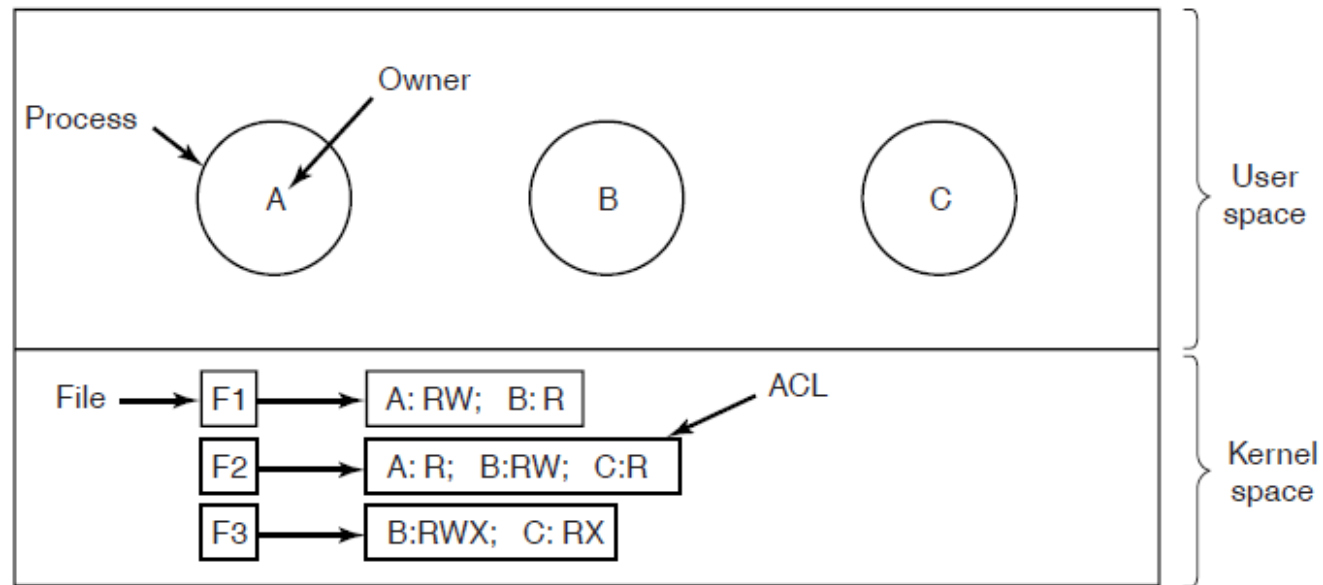
## ACCESS CONTROL LIST (ACL) e CAPABILITY LIST (C-List)

In realtà una matrice di accesso è normalmente talmente sparsa che non viene quasi mai memorizzata, in quanto sarebbe un inaccettabile spreco di memoria.

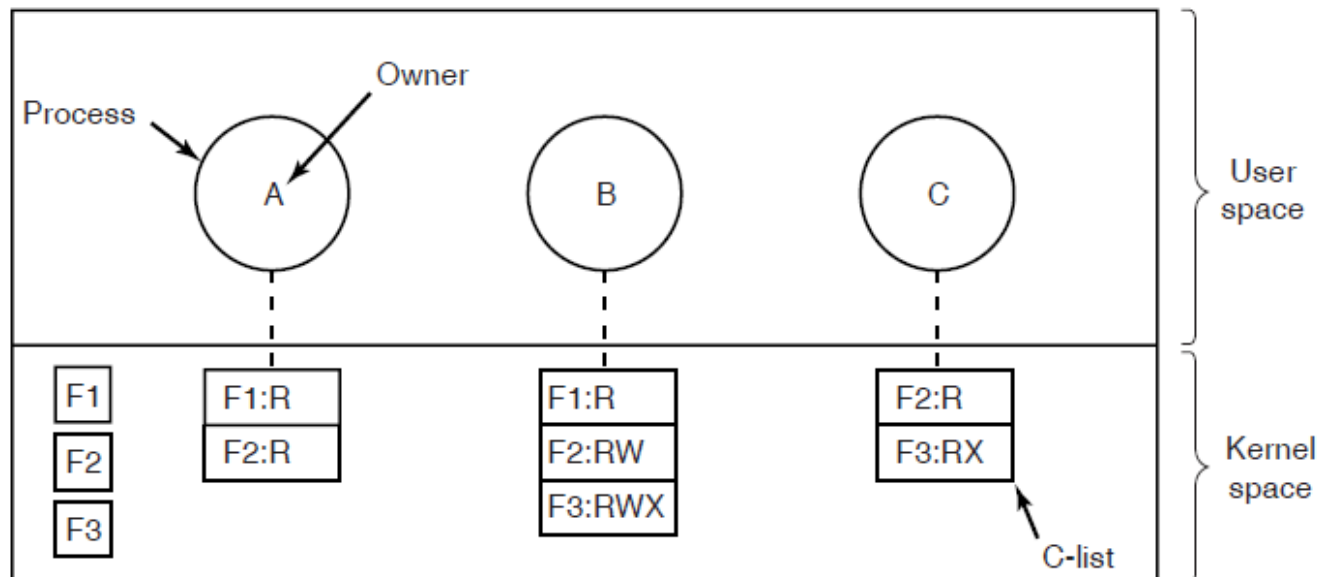
In pratica vi sono **due metodi di rappresentazione/memorizzazione delle matrici di accesso**:

- ➡ **per colonne** (memorizzando solo le celle non vuote); ad ogni oggetto si associa così una lista (ordinata) dei domini a cui è consentito accedervi e delle operazioni lecite.
  - **liste di controllo d'accesso (Access Control List)**,
- ➡ **per righe** (memorizzando solo le celle non vuote); ad ogni processo si associa così una lista di oggetti con le relative operazioni consentite, ovvero, i relativi domini.
  - **liste di capacità (Capability List)**.

## Access Control List (ACL)



## Capability List (C-List)



## CAPABILITIES

Ovviamente è fondamentale che i processi non manipolino a proprio piacimento le capability list.

Esistono tre tecniche principali per impedire che ciò avvenga:

- ➡ *utilizzare una tagged architecture* ovvero un modello di hardware nel quale ogni parola di memoria ha un bit addizionale, detto tag bit (modificabile solo in kernel mode), che dice se la parola contiene una capability o meno (es.: IBM AS/400); il tag bit non viene usato nelle istruzioni aritmetiche, di confronto o in altre istruzioni ordinarie;
- ➡ *mantenere le capability all'interno del kernel*;
- ➡ *mantenere le capability in user space*, ma far uso di tecniche crittografiche per *prevenirne l'alterazione fraudolenta*.



## IL PROBLEMA DELLA SICUREZZA

Come si è in precedenza osservato, quando si parla di sicurezza, bisogna prendere in considerazione l'ambiente esterno in cui il sistema viene a trovarsi in modo da **proteggere il sistema stesso 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 (accessi non autorizzati e conseguenti comportamenti fraudolenti), in realtà **la maggior parte dei problemi deriva da cause accidentali.**

# 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:

- ✓ identificare *qualcosa che l'utente conosce*;
- ✓ identificare *qualcosa che l'utente possiede*;
- ✓ identificare *qualche caratteristica fisica* dell'utente.

Chi cerca di accedere ad un sistema, violando il sistema di autenticazione, viene definito **cracker**.

## AUTENTICAZIONE TRAMITE PASSWORD (1/2)

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

- un nome di *login valido*,
- 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 (2/2)

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

$$P_1 = \underbrace{f(f(\dots f(s) \dots))}_n, \quad P_2 = \underbrace{f(f(\dots f(s) \dots))}_{n-1}, \dots, P_{i-1} = f(P_i);$$

- il server viene inizializzato con  $P_0 = f(P_1)$  (valore memorizzato assieme al nome di login ed all'intero 1);

- 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.

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 ASPETTI FISICI DELL'UTENTE (*BIOMETRICS*)

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

- ➡ ***misurazione e registrazione in un database*** delle caratteristiche dell'utente al momento della sua registrazione;
- ➡ ***identificazione dell'utente*** tramite rilevamento e confronto delle caratteristiche con i valori registrati;

**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 "opera".
-  Meccanismi di "exploit" che consentono a programmi scritti dagli utenti di essere eseguiti con privilegi di altri utenti.

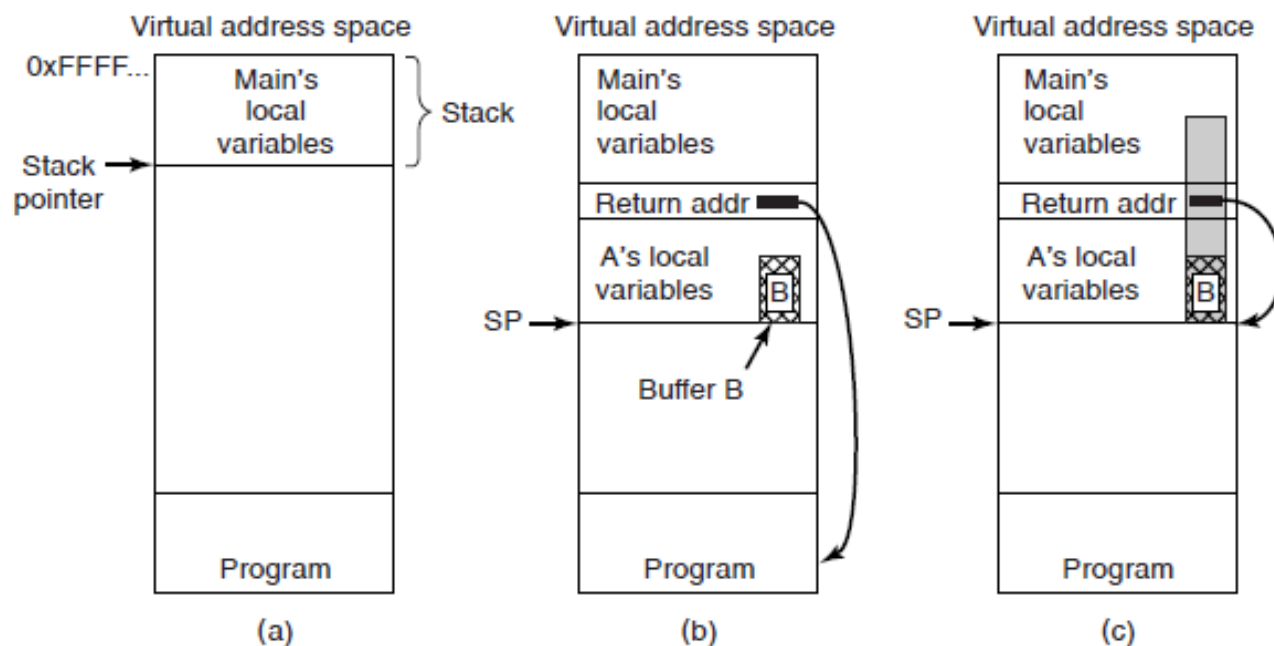
- **Trap Door**

Nome utente o password "speciali" che consentono di eludere le normali procedure di sicurezza. Può essere inclusa in un compilatore.

## BUFFER OVERFLOW (1/2)

- 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;
```



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

## BUFFER OVERFLOW (2/2)

- 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 il *record di attivazione 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.
- Il buffer overflow è riproducibile su sistemi Linux con kernel fino alla serie 2.4.
- Infatti, a partire dalla serie stabile successiva (2.6), è stato introdotto un meccanismo per rendere la vita più difficile a chi intende sfruttare la tecnica del buffer overflow (VA Patch).
- Il meccanismo introdotto consiste nel **rendere variabile (di esecuzione in esecuzione) l'indirizzo di inizio dello stack** (e, di conseguenza, l'indirizzo di inizio dell'ambiente sfruttato).

# TROJAN HORSE

(1/3)

- 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

(2/3)

- 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

(3/3)

- 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 vogliono 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.

## 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 in quello di un altro programma.

✚ Caratteristiche del virus “perfetto”:

- a) è in grado di diffondersi rapidamente,
- b) è difficile da rilevare,
- c) 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**: Vanno in esecuzione quando viene lanciato il programma “compagno”, quindi è sufficiente denominare il virus con il nome di un programma molto utilizzato e dargli come estensione com.
- ➡ **Virus che infettano eseguibili**: si replica iniettando il proprio codice in quello di altri programmi.
- ➡ **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).
- ➡ **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):

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