



UNIVERSITÀ DI PERUGIA
Dipartimento di Matematica e Informatica



Appunti Cybersecurity Teoria

Autore: Chiara Luchini

Basati su:

- Slides del Prof. Francesco Santini
- Slides del Prof. Stefano Bistarelli

Anno Accademico 2020-2021

Indice

I Appunti parte Prof. Santini	6
1 Alcune terminologie	7
1.1 Security Flaws	7
1.2 Vulnerabilità	7
1.2.1 Le fallo non sono vulnerabilità	8
1.3 Policy	8
1.4 Exploit	8
1.5 Zero-day exploit	8
1.6 Mitigazioni	9
1.6.1 Gestione delle fallo e delle vulnerabilità	9
1.7 Collegamento con CIA	9
2 Linguaggio C	10
2.0.1 CERT C coding standard	10
2.0.2 Array e stringhe	10
2.1 Vulnerabilità delle stringhe ed exploit	13
2.1.1 Valori corrotti	13
2.1.2 Buffer overflow	14
2.2 Memoria	16
2.2.1 Organizzazione di un processo in memoria	16
2.2.2 Puntatori	18
2.3 Stack overflow	19
2.4 Stack smashing	25
2.4.1 Arc Injection	25
2.4.2 Code Injection (Shell Code)	27

3	Return-oriented Programming ROP	30
3.0.1	Funzionamento dell'attacco	30
3.0.2	Qualche problema teorico	34
3.0.3	Come si fa exploit/previene	34
3.1	Forme di mitigazione	35
3.1.1	Stack-Smashing Protector	35
3.1.2	I canarini	35
3.1.3	Address Space Layout Randomization	36
3.1.4	Stack non-eseguibile	36
3.1.5	W xor X	36
4	Heap Overflows	37
4.0.1	Alcuni problemi dello heap	37
4.1	Heap overflow	37
4.1.1	Dmalloc	37
4.1.2	Tecnica unlink	41
5	Pointer subterfuge	45
5.0.1	Puntatori a funzioni	45
5.0.2	Puntatore a oggetti	46
6	IDManag	49
6.1	intro	49
6.1.1	Identità digitale	49
6.1.2	Autenticazione	49
6.1.3	Autorizzazione	50
6.2	Identy	50
6.2.1	Identifier	50
6.2.2	Identità	50
6.2.3	Account	51
6.2.4	Separazione tra ID e account	51
6.2.5	Non Human Identifier	51
6.2.6	IDM System	52
6.3	Eventi in un ciclo di vita di un'identità	52
6.3.1	Provisioning	52

6.3.2	AUTHORIZATION	52
6.3.3	AUTHENTICATION	52
6.3.4	ACCESS POLICY ENFORCEMENT	53
6.3.5	Sessioni	53
6.3.6	Single Sign-on	53
6.3.7	STRONGER AUTHENTICATION	54
6.3.8	Logout	54
6.3.9	ACCOUNT MANAGEMENT AND RECOVERY	54
6.3.10	DEPROVISIONING	55
II	Appunti parte Prof.Bistarelli	56
1	Cybersecurity	57
1.1	Introduzione alla sicurezza informatica	57
1.1.1	Terminologie	61
1.1.2	Asset di un sistema informatico	62
1.1.3	Vulnerabilità, minacce e attacchi	62
1.1.4	Asset e minacce	67
1.1.5	Attacchi passivi e attivi	69
1.1.6	Requisiti di sicurezza	71

Libri di riferimento

Libri per appunti Prof.Bistarelli

- Computer Security: Principles and Practice, Global Edition [6]

Libri per appunti Prof.Santini

- Secure Coding in C and C++ [1]

Parte I

Appunti parte Prof. Santini

Capitolo 1

Alcune terminologie

In questo Capitolo verranno riportate alcune delle terminologie utilizzate nel campo della Cybersecurity.

1.1 Security Flaws

Una **falla o difetto della sicurezza** è un difetto del software che rappresenta un potenziale rischio per la sicurezza. Quest’ultimo può essere visto come la codifica di un errore umano nel software incluse le omissioni, infatti un difetto del software può originarsi in ogni momento del ciclo di vita di questo. Ovviamente non tutti i difetti nel programma sono delle fallo di sicurezza, quelli che lo sono devono essere rilevati ed eliminati in modo da evitare potenziali attacchi. Questa premessa sottolinea la relazione tra ingegneria del software e sicurezza del programma. Un aumento della qualità nella scrittura del codice porta anche a un aumento della sicurezza. Tuttavia, molte fallo di sicurezza non vengono rilevate perché processi di sviluppo software tradizionali raramente considerano l’esistenza di aggressori.

1.2 Vulnerabilità

Una **vulnerabilità** è un insieme di condizioni che permetto all’attaccante di violare un policy di sicurezza esplicita o implicita. Una falla nella sicurezza di un programma può rendere il software vulnerabile agli attachi quando i dati in input superano un limite di sicurezza.

Ciò può verificarsi quando un programma contenente una falla è installato con privilegi di esecuzione maggiori di quelli della persona che lo esegue o è utilizzato da un servizio di rete dove i dati in input arrivano tramite la connessione a internet.

1.2.1 Le falle non sono vulnerabilità

Una falla può esistere senza tutte le precondizioni necessarie a creare una vulnerabilità. Viceversa una vulnerabilità può esistere senza una falla. Poichè essendo la sicurezza un attributo di qualità che deve essere scambiato con altri come ad esempio le performance e l'usabilità, i designer di software possono intenzionalmente lasciare i loro prodotto vulnerabile per alcune forme di exploit. Ciò significa che il designer ha accettato il rischio per conto del consumatore.

1.3 Policy

Una **policy di sicurezza** è un insieme di regole e pratiche che specificano o regolano come un sistema o un'organizzazione fornisce dei servizi di sicurezza per proteggere delle risorse di sistema sensibili e critiche. Coloro che sono documentate, ben conosciute, e visibilmente imposte possono aiutare a stabilire il comportamento dell'utente.

1.4 Exploit

L'**exploit** è una tecnica che trae vantaggio di una vulnerabilità di sicurezza per violare un policy di sicurezza esplicita o implicita. Le vulnerabilità sono soggette a sfruttamento, gli exploit possono essere di diverso tipo dai worms ai virus fino ad arrivare ai trojan. Una buona compresione di come il programma può essere sfruttato è un valido strumento da utilizzare per sviluppare un software sicuro.

1.5 Zero-day exploit

Una vulnerabilità **zero-day** o **0-day** è una vulnerabilità del software-computer che è sconosciuta a coloro che sarebbero interessati nel mitigare quest'ultima, incluso il fornitore del software. Un exploit mirato a un zero-day è chiamato **zero-day exploit** o **zero-day attack**.

1.6 Mitigazioni

La **mitigazione** racchiude i metodi, tecniche, processi, strumenti, o librerie di runtime che possono prevenire o limitare gli exploit contro le vulnerabilità. Una mitigazione o contromisura è una soluzione per le fallo o una soluzione alternativa che può essere applicata per prevenire l'exploit. Esempi:

- A livello di codice sorgente: una mitigazione può essere una semplice sostituzione di un'operazione di string copy illimitata con un limitata;
- A livello di sistema o network: una mitigazione può coinvolgere lo spegnimento di una porta o il filtraggio del traffico per prevenire un attacco.

1.6.1 Gestione delle fallo e delle vulnerabilità

Il metodo preferito per eliminare una falla è di trovare il difetto e correggerlo. Tuttavia, in alcuni casi è più conveniente eliminare le fallo prevenendo l'utilizzo di input dannosi nel difetto. Un altro modo di gestire le vulnerabilità consiste nell'isolamento di quest'ultima, ovviamente affrontandola operativamente ne aumenta il costo della mitigazione poichè esso viene spostato dallo sviluppatore agli amministratori di sistema e agli utenti finali.

1.7 Collegamento con CIA

Una risorsa fisica o logica può avere una o più vulnerabilità che possono essere sfruttate da un attaccante. Il risultato può compromettere potenzialmente la **confidenzialità**, l'**integrità** o la **disponibilità** delle risorse.

Capitolo 2

Linguaggio C

Il linguaggio di programmazione C è conosciuto per essere un linguaggio leggero con un numero di "tracce" ridotto. Alcuni programmatore abituati ad usare altri linguaggi come Java, Pascal o Ada credono che il linguaggio li protegga di più rispetto a quello che effettivamente fa. Queste false ipotesi hanno portato i programmatore a fallire nel prevenire la scrittura di un array oltre i limiti, fallire nel rilevare overflow di interi e anche nel chiamare funzioni con un numero errato di parametri.

2.0.1 CERT C coding standard

Il SEI CERT C Coding Standard è una codifica standard per i software per il linguaggio di programmazione C, sviluppato da il CERT Coordination Center per migliorare la sicurezza, l'affidabilità del sistema di software.

2.0.2 Array e stringhe

Problemi con gli array. Il CERT C Secure Coding Standard include il fatto di non applicare l'operatore *sizeof* a un puntatore quando si vuole la grandezza dell'array. Esempio:

```

01 void clear(int array[]) {
02     for (size_t i = 0; i < sizeof(array) / sizeof(array[0]); ++i) {
03         array[i] = 0;
04     }
05 }
06
07 void dowork(void) {           When passed as a parameter, an
08     int dis[12];             array name is a pointer
09
10     clear(dis);             sizeof(int *) == 8 always
11     /* ... */
12 }
```

Figura 2.1: Esempio *sizeof(array)* .

Stringhe. Le stringhe sono un concetto fondamentale nell'ingegneria del software, ma essi non sono un tipo integrato in C o C++. Infatti la libreria standard di C supporta le stringhe di tipo `char` e le stringhe larghe di tipo `wchar_t`.

```

01 #include <stdio.h>
02 #include <stdlib.h>
03
04 void get_y_or_n(void) {
05     char response[8];
06     puts("Continue? [y] n: ");
07     gets(response);
08     if (response[0] == 'n')
09         exit(0);
10     return;
11 }
```

- 01 char *gets(char *dest) {
- 02 int c = getchar();
- 03 char *p = dest;
- 04 while (c != EOF && c != '\n') {
- 05 *p++ = c;
- 06 c = getchar();
- 07 }
- 08 *p = '\0';
- 09 return dest;
- 10 }

Figura 2.2: Esempio *gets* .

Il problema principale con la funzione `gets()` è che non fornisce alcun modo di specificare un limite sul numero dei caratteri da leggere. Quest'ultima risulta deprecata e eliminata, il CERT consiglia di non utilizzare funzioni deprecate o obsolete.

Lettura da `stdin()`. La lettura dei dati da una fonte illimitata come `stdin()` crea dei problemi per il programmatore poiché non è possibile conoscere in anticipo

quanti caratteri un utente utilizzerà, infatti è impossibile pre-allocare un array di sufficiente lunghezza. Una soluzione comune è di allocare staticamente un array più grande rispetto al necessario. Questo approccio funziona con gli utenti "amichevoli", ma con gli attaccanti un array di caratteri di lunghezza fissa può essere facilmente superato. Infatti questo metodo è proibito dal CERT il quale afferma che non si può copiare dati da una fonte illimitata in un array con lunghezza fissata.

"STR35-C. Do not copy data from an unbounded source to a fixed-length array."

Parametri dei programmi. Le vulnerabilità possono occorrere quando uno spazio indeguato è allocato per copiare un input di un programma come un argomento della riga di comando.

```
1 int main(int argc, char *argv[]) {  
2     /* ... */  
3     char prog_name[128];  
4     strcpy(prog_name, argv[0]);  
5     /* ... */  
6 }
```

Figura 2.3: Esempio vulnerabilità con parametri dei programmi .

Nonostante `argv[0]` contiene il nome del programma, un attaccante può controllare il contenuto di `argv[0]` per causare una vulnerabilità fornendo una stringa con più di 128 bytes.

Null terminating string. In Figura 2.4 possiamo vedere che il risultato di `strcpy()` su `c` porta l'utente a poter scrivere oltre i limiti dell'array poichè la stringa salvata in `a[]` non terminata correttamente (null-terminating).

```

1 int main(void) {
2     char a[16];
3     char b[16];
4     char c[16];
5     strncpy(a, "0123456789abcdef", sizeof(a));
6     strncpy(b, "0123456789abcdef", sizeof(b));
7     strcpy(c, a);
8     /* ... */
9 }
```

Figura 2.4: Esempio null terminating string .

“STR32-C. Null-terminate byte strings as required.”

Funzionamento esempio 2.4 Abbiamo 3 array lunghi 16 caratteri, copiamo la stringa "0123456789abcdef" in **a** e in **b**, come si può notare questa stringa ha una lunghezza di 16 caratteri. Successivamente si copia la stringa contenuta in **a** nell'array **c** e questo porta ad avere dei problemi ovvero:

- non viene salvato sia in **a** che in **b** il carattere di fine stringa "\0" allora essendo che la stringa non termina e che l'array **b** è memorizzato subito dopo **a** si ha di conseguenza che

$$a = [0123456789abcde0123456789abcde\0]$$

- quindi quando si va a copiare **a** dentro **c** si copiano 32 caratteri più "\0" e non 16 sforando così la memoria allocata inizialmente.

2.1 Vulnerabilità delle stringhe ed exploit

2.1.1 Valori corrotti

Precedentemente abbiamo visto alcuni degli errori più comuni nel manipolare il linguaggio C o C++. Questi errori sono pericolosi nel momento in cui il codice opera

con dati esterni non fidati come gli argomenti delle linee di comando, variabili d'ambiente, input della console, file di testo e connessione a internet. **Infatti è meglio considerare tutti i dati esterni al codice come non fidati.** Nell'analisi della sicurezza dei software, un valore è chiamato **corrotto (tainted)** quando viene da una risorsa non fidata, al di fuori del controllo del programma, e non è stato sanificato per assicurarsi che sia conforme a qualsiasi vincolo posto su di esso.

```
01 bool IsPasswordOK(void) {
02     char Password[12];
03
04     gets(Password);
05     return 0 == strcmp(Password, "goodpass");
06 }
07
08 int main(void) {
09     bool PwStatus;
10
11     puts("Enter password:");
12     PwStatus = IsPasswordOK();
13     if (PwStatus == false) {
14         puts("Access denied");
15         exit(-1);
16     }
17 }
```

Figura 2.5: Esempio password.

2.1.2 Buffer overflow.

In questo caso la falla sta nella funzione `IsPasswordOk()` poichè permette ad un attaccante di guadagnare l'accesso non autorizzato causato dalla funzione `gets()`. Questa tipologia di attacco che permette una scrittura oltre i limiti viene definita **buffer overflow**.

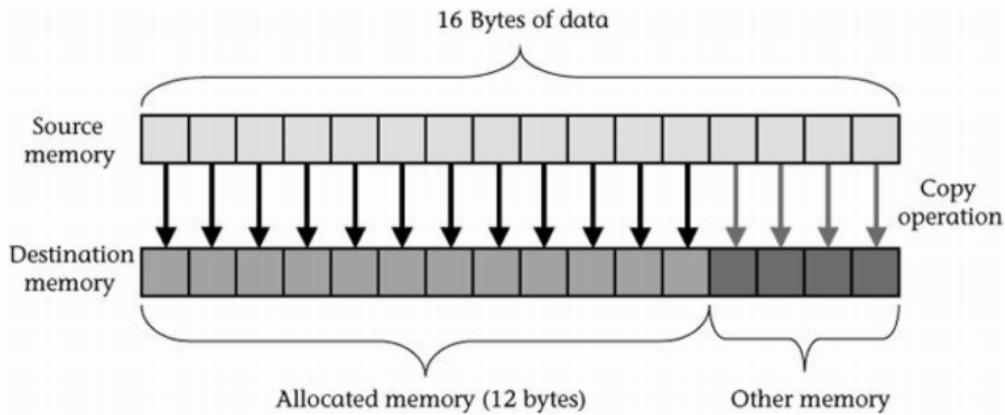


Figura 2.6: Allocazione della memoria.

Oltre questo è presente anche un altro problema ovvero che il programma `IsPasswordOk` non controlla lo stato di ritorno di `gets`.

“FIO04- C. Detect and handle input and output errors.”

Quando si fallisce l'inserimento della password non viene controllato da nessuno perciò il contenuto del buffer `Password` è indeterminato, nei programmi reali esso potrebbe contenere la password del precedente utente. Il buffer overflow occorre quando un dato è scritto oltre i limiti della memoria allocata in una particolare struttura. Il linguaggio C e C++ sono suscettibili a questi attacchi poichè:

- Definisce le stringhe come array di caratteri con terminazione null;
- Non usano dei metodi per il controllo implicito dei limiti;
- Fornisce funzioni per le stringhe che non applicano dei controlli.

Non tutti i buffer overflow portano a delle vulnerabilità, solo nel caso in cui un attaccante può manipolare gli input controllati dall'utente per sfruttare una falla di sicurezza.

Il buffer overflow può essere sfruttato sia nella memoria **heap** che in quella **statica** sovrascrivendo strutture di memoria adiacenti. Per aiutare nel verificare la presenza di un buffer overflow esistono dei programmi che a tempo di compilazione (**staticamente**) o in modo **dinamico** (eseguono il programma passando delle stringhe dove viene richiesto) controllano la presenza o meno di quest'ultimi.

2.2 Memoria

2.2.1 Organizzazione di un processo in memoria

Nell'immagine sottostante possiamo vedere come è organizzata la memoria in 3 differenti casi, il primo è il caso generale che andremo a prendere in considerazione, il secondo riguarda la memoria in Linux e l'ultimo in Windows.

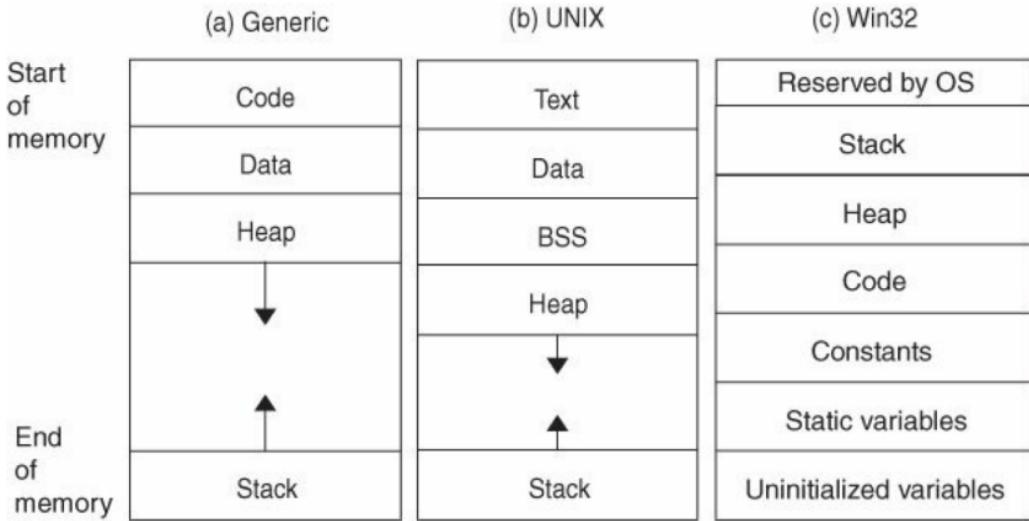


Figura 2.7: Organizzazione della memoria.

Studiamo le varie parti della memoria nel caso generale:

- **Code:** zona in cui viene salvato il codice del programma;
- **Data:** memoria in cui sono salvate le variabili globali e le variabili locali statiche, inizializzate e non;
- **Heap:** memoria per allocazione dinamica dei processi, gli indirizzi di memoria aumentano in modo crescente;
- **Stack:** zona LIFO (pila) in cui vengono salvate le variabili locali alle funzioni (supporto per l'esecuzione dei processi), gli indirizzi di memoria aumentano in modo decrescente .

Noi vedremo degli esempi di buffer overflow nello stack, è molto importante ricordarsi come funziona e come vengono salvate le variabili, se si alloca dello spazio si va da un numero più alto a uno più basso e viceversa (come una fisarmonica). La memoria è struttura in questo modo così la parte dinamica (heap) non si incontra con la parte statica (stack) a meno che non viene riempita completamente.

La memoria UNIX è molto simili alla memoria generale, l'unica differenza sta nel fatto che la parte **Data** nella memoria generale viene suddivisa in due in quella UNIX:

- **Data:** vengono salvate delle variabili globali/statiche inizializzate;
- **BSS(Block Started by Symbol):** vengono salvate le variabili globali che non sono state inizializzate e vengono di default inizializzate a 0.

Il segmento **Text** è l'equivalente del segmento **Code**, entrambi includono le istruzioni e i dati read-only.

Esecuzione di un programma. Vediamo un primo esempio di esecuzione di un programma. In questo caso possiamo vedere due funzioni `main()` e `fun(...)`, si parte salvando in memoria le variabili locali presenti nella funzione `main()` la quale richiama l'altra. In Figura 2.8 possiamo vedere che le variabili vengono salvate in sequenza dal basso verso l'alto in base a come sono state inizializzate, successivamente con la chiamata di `fun()` si salvano i parametri che essa prende in input, l'indirizzo di ritorno (**return address**¹) e la variabile `res`. Il segmento che contiene la variabile del `main` si chiama **stack/frame main**, stessa definizione per il segmento di `fun`. Al termine dell'esecuzione di `fun` si cancella dalla memoria tutto quello riguardante quest'ultima e si ritorna all'istruzione successiva alla chiamata di funzione in `main`. Per riportare il controllo nella posizione corretta, deve essere salvato l'indirizzo di ritorno. Lo stack è adatto a mantenere queste informazioni perché è una struttura dati dinamica in grado di supportare qualsiasi livello di annidamento all'interno vincoli di memoria.

¹indica l'indirizzo di ritorno dopo che è stata eseguita la funzione in questione, in questo caso indica l'indirizzo del `main`.

```

int fun(int p1, int p2, int p3) {
    int res= 0;
    res= p1 + p2 + p3;
    return res;
}

```

```

int main() {
    int a= 4, b= 5, c= 7;
    a= fun(a,b,c);
}

```

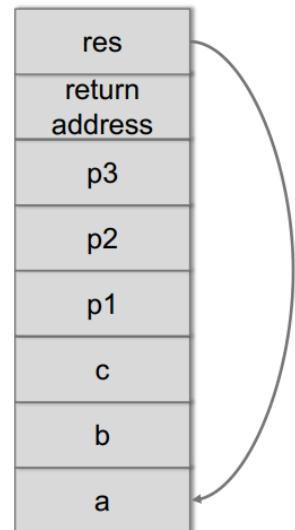


Figura 2.8: Esempio esecuzione processo.

2.2.2 Puntatori

Extended base pointer (EBP)

L'indirizzo del frame corrente è salvato all'interno del frame o nel **base pointer register**. Il registro **extended base pointer (ebp)** è usato per questo scopo, esso è usato anche come punto fisso di riferimento all'interno dello stack. Quando una subroutine è chiamata, il puntatore del frame per la routine di chiamata viene anch'esso inserito nello stack in modo che può essere ripristinato quando la subroutine termina.

Instruction pointer

Il **puntatore di istruzione (eip)** punta all'istruzione successiva da eseguire, quando si esegue una sequenza di istruzioni esso è aumentato automaticamente dalla grandezza di ogni istruzione. Eip non può essere modificato direttamente ma deve essere modificato indirettamente con delle istruzioni come :

- jump;
- call;
- return.

Extended stack pointer (ESP)

L'extended stack pointer (esp) è il puntatore corrente allo stack, esso punta alla parte superiore della pila.

2.3 Stack overflow

Secondo esempio In questo secondo esempio possiamo vedere che ci sono 3 funzioni (`main`, `f1` e `f2`) che si chiamano a vicenda. Quando viene eseguito il programma viene inserito nello stack il frame del `main`, supponiamo che esso inizi dall'indirizzo 1000, il quale viene salvato all'interno del ebp e del registro della CPU. Successivamente quando viene chiamata la funzione `f1` l'ebp all'interno del registro della CPU cambia con quello dell'inizio del frame di `f1`, ovvero 1008. All'interno del frame di `f1` vengono salvati oltre alle variabili locali anche il valore del ebp corrente(1008) e il valore di ritorno, il quale è uguale all'epb del `main` poiché appena terminata `f1` la CPU deve tornare a eseguire le istruzioni successive alla chiamata di funzione nel frame `main`.

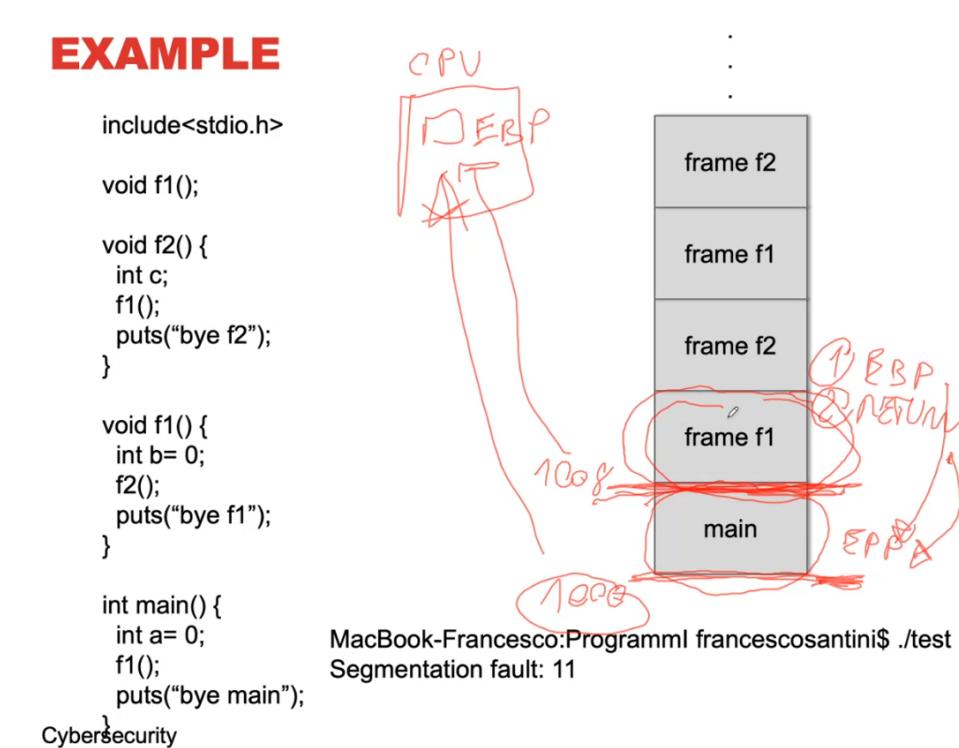


Figura 2.9: Esempio esecuzione processo con spiegazione.

Esempio disassembly in intel. Adesso vediamo cosa succede a livello assembly² quando si chiama una funzione. La funzione `main` alloca due variabili `MyInt` (intero, 4 byte) e `*MyStrPtr` (puntatore a carattere, 4 byte), poi chiama la funzione `foo` a cui vengono passati come parametri le variabili definite prima. Ci sono degli errori fra la spiegazione del professore e i commenti nell'esempio, ovvero `MyInt` come parametro viene allocato per primo nello stack secondo il professore mentre nel commento a riga 4 viene allocato come secondo rispetto a `MyStrPtr` poiché gli argomenti delle funzioni vengono allocati da destra verso sinistra, per comodità seguiamo la spiegazione delle slides. A livello assembly quando viene chiamata la funzione vengono effettuate le seguenti operazioni,:

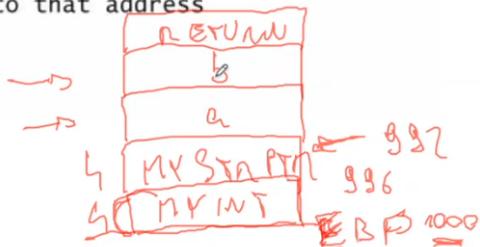
1. `mov eax, [ebp- 4]`: si sposta quello che si trova in `ebp-4` (`MyStrPtr`) in `eax`, dove `ebp = 1000` e `ebp-4 = 996`;
2. `push eax`: si inserisce `eax` (`MyStrPtr`) nello stack;
3. `mov ecx, [ebp- 8]`: si sposta quello che si trova in `ebp-8` (`MyInt`) in `ecx`, dove `ebp = 1000` e `ebp-8= 992`;
4. `push ecx`: si inserisce `ecx` (`MyStrPtr`) nello stack;
5. `call foo`: chiamiamo la funzione `foo` e inseriamo nello stack il return address in cima al frame di `foo`;
6. `add esp, 8`: aggiunge 8 byte al puntatore `esp` dopo il return di `foo`.

²Il linguaggio assembly (detto anche linguaggio assemblativo o linguaggio assemblatore o semplicemente assembly) è un linguaggio di programmazione molto simile al linguaggio macchina, pur essendo differente rispetto a quest'ultimo, [2].

DISASSEMBLY IN INTEL

```

01 void foo(int, char *); // function prototype
02
03 int main(void) {
04     int MyInt=1; // stack variable located at ebp-8 4 BYTES
05     char *MyStrPtr="MyString"; // stack var at ebp-4 4 BYTES
06     /* ... */
07     foo(MyInt, MyStrPtr); // call foo function
08     mov eax, [ebp-4]           # Push 2nd argument on stack
09     push eax
10     mov ecx, [ebp-8]           # Push 1st argument on stack
11     push ecx
12     call foo                 # Push the return address on stack and
13     add esp, 8                # jump to that address
14     /* ... */
15 }
16 }
```



Cybersecurity

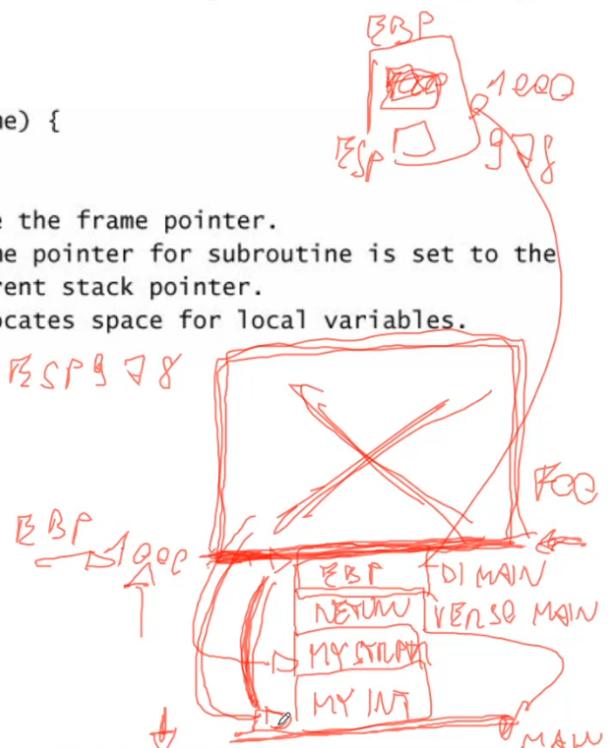
Figura 2.10: Esempio disassembly main in intel.

Vediamo ora come si comporta la funzione `foo` quando viene chiamata, Figura 2.11. Dopo aver salvato il return address salviamo anche l'ebp della funzione `main` in modo da ricordarsi dove inizia il suo frame quando la chiamata di funzione `foo` termina. Dopo ebp di `main` inizia il frame di `foo` dove verranno salvate tutte le sue variabili locali, esso è grande 28 byte.

1. `push ebp`: salva il puntatore al frame corrente ovvero ebp di `main`;
2. `mov ebp, esp`: salviamo il puntatore alla fine del frame di `main` esp nel ebp, in modo da iniziare il frame di `foo` alla fine di quello del `main`;
3. `sub esp, 28`: allochiamo 28 byte di spazio per `foo` con `esp - 28`.

DISASSEMBLY OF FOO (PROLOGUE)

```
1 void foo(int i, char *name) {  
2     char LocalChar[24];  
3     int LocalInt;   
4     push ebp      # Save the frame pointer.  
5     mov ebp, esp  # Frame pointer for subroutine is set to the  
6                  # current stack pointer.  
7     sub esp, 28    # Allocates space for local variables.  
8     /* ... */
```



Cybersecurity

Figura 2.11: Esempio disassembly foo in intel.

Come si può vedere anche nell’immagine sottostante, il frame di `foo` è costituito dai seguenti elementi in questo ordine:

1. Primo parametro della funzione `foo`: `toMyString` (4 byte);
2. Secondo argomento: `MyInt` (4 byte);
3. Return address del `main` (4 byte);
4. `Ebp` del `main` (4 byte);
5. Prima variabile locale `LocalChar` (24 byte);
6. Seconda variabile locale `LocalInt` (4 byte).

Address	Value	Description	Length
0x0012FF4C	?	Last local variable—integer: LocalInt	4
0x0012FF50	?	First local variable—string: LocalChar	24
0x0012FF68	0x12FF80	Calling frame of calling function: main()	4
0x0012FF6C	0x401040	Return address of calling function: main()	4
0x0012FF70	1	First argument: MyInt (int)	4
0x0012FF74	0x40703C	Second argument: pointer toMyString (char *)	4

Figura 2.12: Frame di foo.

Infine vediamo cosa succede quando facciamo il return di foo, in generale lo scopo è di disallocare tutto lo spazio di memoria della funzione foo e di tornare all’istruzione successiva del main. Esso è composto da più comandi in assembly che sono:

1. `mov esp,ebp`: allo stack pointer esp inserisco il valore del ebp del frame corrente, ovvero si passa ad esempio da `esp = 900 → 1000`.
2. `pop ebp`: ristabilisce il frame pointer ebp con quello della funzione chiamante (main), ovvero disallocando i 28 byte del frame di foo il primo elemento che troviamo in memoria è proprio il valore del ebp del main che ci eravamo salvati in precedenza, facendo il pop lo eliminiamo dalla pila e lo reinseriamo nel registro della CPU;
3. `ret`: fa il pop del return address dallo stack, lo mette nel eip (prossima istruzione da fare ovvero tornare al punto di chiamata) e trasferisce il controllo a quel frame.

Dopo aver fatto il return si esegue il prossimo comando che è appunto `add esp, 8` in Figura 2.10, il quale aggiunge all’esp 8 byte perché dalla memoria devono essere eliminati ancora i parametri passati alla funzione chiamata che sono `MyStrPtr` e `MyInt`. In questo modo si torna l’esp combacia con la parte alta del frame del main e l’ebp con l’inizio di quest’ultimo.

DISASSEMBLING FOO (EPILOGUE)

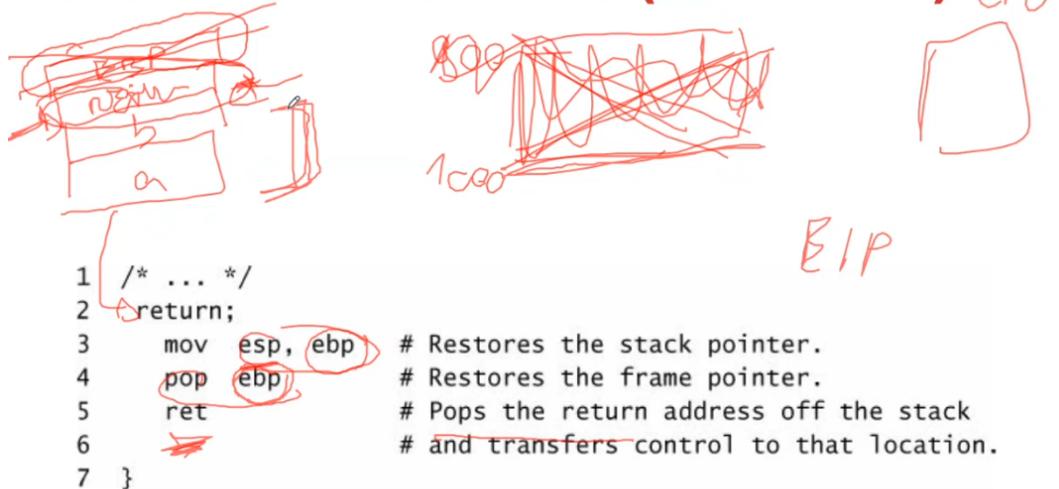


Figura 2.13: Return di foo.

Valori di ritorno. Se ci sono dei valori di ritorno questi sono salvati in eax dalla funzione chiamata prima che ritorni al main. In questo modo la funzione chiamante sa dove si trovano i valori di ritorno e li può usare.

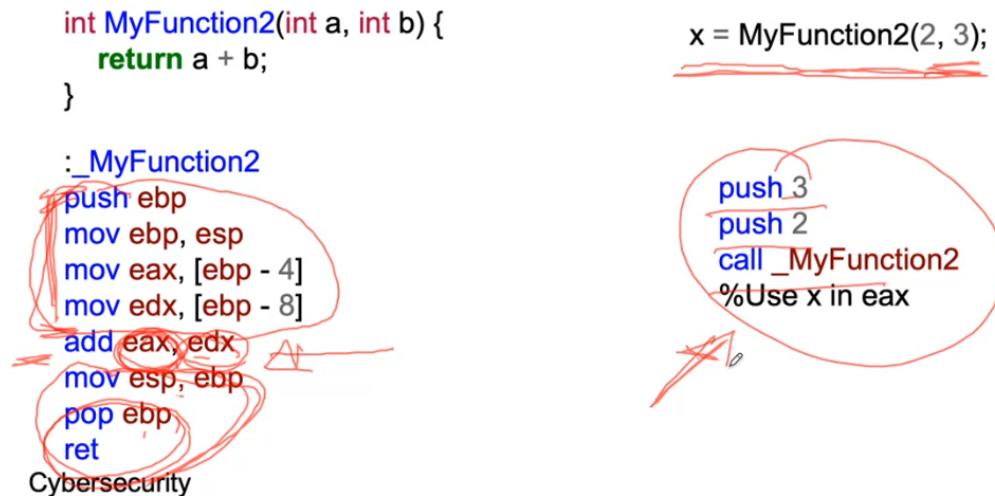


Figura 2.14: Valori di ritorno.

2.4 Stack smashing

Stack smashing è quando un attaccante intenzionalmente fa overflow di un buffer nello stack per ottenere l'accesso a viste regioni della memoria del computer. Questo avviene quando si riscrivono dei dati all'interno della memoria allocata all'esecuzione dello stack. Si possono avere delle serie conseguenze come ad esempio la modifica di valori delle **variabili automatiche** o l'esecuzione di codice arbitrario. Un esempio comune è sovrascrivere il return address che si trova nello stack.

2.4.1 Arc Injection

Esempio *IsPasswordOk*: Arc Injection

Nell'esempio in Figura 2.5 il programma è vulnerabile a stack smashing. Questo difetto può essere facilmente dimostrato inserendo una password di 20 caratteri “1234567890123456 W>*!” che porta a far saltare il programma in modo imprevisto.

```
01 bool IsPasswordOK(void) {
02     char Password[12];
03
04     gets(Password);
05     return 0 == strcmp(Password, "goodpass");
06 }
07
08 int main (void) {
09     bool PwStatus;
10     puts("Enter Password: ");
11     PwStatus=IsPasswordOK();
12     if (!PwStatus) {
13         puts("Access denied");
14         exit(-1);
15     }
16     else
17         puts("Access granted");
}
```

Figura 2.15: IsPasswordOk nuova.

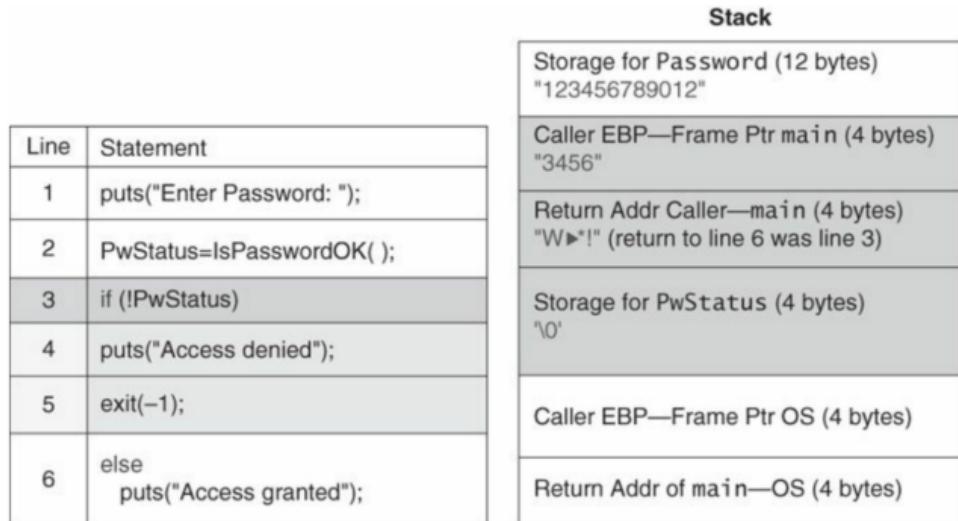


Figura 2.16: Funzionamento stack smashing.

In memoria l'ultima sequenza di quattro caratteri W▶*! corrisponde a un indirizzo di 4 byte che sovrascrive l'indirizzo di ritorno nello stack, quindi invece di tornare all'istruzione subito dopo la chiamata in `main()` ritorna al ramo "Access granted" bypassando così il controllo della password e autorizzando ad avere accesso al sistema. Un metodo per capire dove è salvato il return address di una funzione nello stack è l'utilizzo del comando `void * __builtin_return_address (unsigned int level)` il quale permette di con un valore di 0 di avere il return address della funzione corrente e con un valore di 1 ritorna il valore della funzione chiamante. Questa tecnica di buffer overflow si chiama **Arc Injection**.

La tecnica di **Arc Injection** (a volte chiamata **return into-libc**) prevede il trasferimento del controllo al codice esiste già nella memoria di processo. Questi exploit sono chiamati così perché inseriscono un nuovo arco (trasferimento del flusso di controllo) nel flusso di controllo del programma invece di iniettare nuovo codice. Questa tecnica è preferita rispetto al code injection per vari motivi:

- Si utilizza del codice che si trova già in memoria nel sistema preso di mira, in questo modo le **impronte lasciate dall'attaccante sono significativamente minori**;
- Essendo che questo metodo si basa su codice esistente, **non può essere bloccato da schemi di protezione basati sulla memoria** come la creazione di

segmenti di memoria non eseguibile.

2.4.2 Code Injection (Shell Code)

Quando l'indirizzo di ritorno viene sovrascritto a causa di un software difetto, raramente indica istruzioni valide. Per un attaccante è possibile creare delle stringhe speciali che contengono **un puntatore a qualche codice malevolo**, sempre creato dall'attaccante. Questo implica che quando il return address sovrascritto viene invocato il controllo è trasferito al codice iniettato, il quale viene eseguito con i permessi del programma. Per questo i programmi che vengono eseguiti con permessi di **root** o dei **privilegi elevati** sono il target di certi attacchi, infatti frequentemente il codice malevolo apre una shell remota (**shellcode**) nella macchina compromessa permettendo all'attaccante di inserire dei comandi.

Esempio *IsPasswordOk*: Code Injection.

Prendendo sempre come esempio il programma "IsPasswordOk" 2.15.

INJECTION

```
01 /* buf[12] */
02 00 00 00 00
03 00 00 00 00
04 00 00 00 00
05
06 /* %ebp */
07 00 00 00 00
08
09 /* return address */
10 78 fd ff bf
11
12 /* "/usr/bin/cal" */
13 2f 75 73 72
14 2f 62 69 6e
15 2f 63 61 6c
16 00 00 00 00
17
18 /* null pointer */
19 74 fd ff bf
20
21 /* NULL */
22 00 00 00 00
23
24 /* exploit code */
25 b0 0b      /* mov $0xb, %eax */
26 8d 1c 24    /* lea (%esp), %ebx */
27 8d 4c 24 f0 /* lea -0x10(%esp), %ecx */
28 8b 54 24 ec /* mov -0x14(%esp), %edx */
29 cd 50      /* int $0x50 */
```

% ./BufferOverflow < exploit.bin
(exploit.bin is the "payload")

Cybersecurity

Figura 2.17: Esempio con Code Injection

Anche in questo caso cerchiamo di sovrascrivere il segmento del return address in modo da farlo puntare al nostro codice malevolo che possiamo vedere al commento "exploit code". Il nostro payload, ovvero il nostro codice, è tutto scritto in esadecimale per comodità invece che in binario. Partiamo inserendo 12 caratteri nella variabile "Password" successivamente sovrascriviamo il puntatore ebp con 4 interi casuali, non ci interessa questa zona. Arriviamo infine al punto che ci interessa ovvero il segmento del return address nel quale inseriremo come valore l'esadecimale del segmento dove salviamo il codice di exploit in modo da eseguirlo. Prima di eseguire quest'ultimo salviamo 3 parametri che passeremo successivamente alle funzioni del exploit:

- **"/usr/bin/cal"**: il primo esadecimale si traduce con il comando `/usr/bin/cal`.
- **"null pointer"**: il secondo è un puntatore a null.
- **"NULL"**: caratteri nulli.

Spiegazione exploit code. Vediamo nel dettaglio cosa fanno i comandi scritti in esadecimale del nostro exploit code.

- `mov $0xb, %eax`: assegna 0xB, il quale identifica il numero della chiamata di sistema `execve()`, al registro `%eax`;
`int execve(const char *filename, char *const argv[], char *const envp[]);`
- `lea (%esp), %ebx`: "load effective address", calcola l'indirizzo effettivo del secondo operando (operando sorgente) e lo salva nel primo (operando destinatario). L'operando sorgente è un indirizzo di memoria (parte offset) specificato con una delle modalità di indirizzamento del processore mentre l'operando di destinazione è un registro di uso generale. In questo caso vengono inseriti i tre parametri, salvati in precedenza agli indirizzi `(%esp)`, `-0x10(%esp)` (`esp-16`) e `-0x14(%esp)` (`esp-20`), nei registri ebx, ecx ed edx.
- `int 0x50`: chiamata vera e propria per invocare `execve()`, che comporta l'esecuzione del programma di calendario Linux.

The screenshot shows a terminal window titled "root@localhost:~/exploit". The window has a menu bar with "File", "Edit", "Settings", and "Help". The main area displays the command "[root@localhost exploit]# ./BufferOverflow < exploit.bin" followed by the prompt "Enter Password:". Below the password prompt is a calendar for February 2005, showing days from 1 to 28. The terminal window has scroll bars on the right side.

```
[root@localhost exploit]# ./BufferOverflow < exploit.bin
Enter Password:
February 2005
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28
```

Figura 2.18: Risultato esempio Code Injection.

Capitolo 3

Return-oriented Programming ROP

ROP è una tipologia di attacco a basso livello, assembly. Quando si compila un codice, viene caricata in quasi tutti i programmi Unix anche la libreria standard **libc**, la quale contiene delle routine utili per un attaccante. Per l'attacco vengono utilizzate solo piccole sequenze di codice, esse sono lunghe solo due o tre istruzioni (**gadget**). Alcune di esse sono presenti in **libc** come risultato delle scelte fatte dal compilatore durante la generazione del codice.

3.0.1 Funzionamento dell'attacco

La ROP è una tecnica simile all'arc injection, ma invece di ritornare alle funzioni, il codice di exploit ritorna a sequenze di istruzioni seguite da un return (**ret**). Ogni sequenza di istruzioni utile è chiamata **gadget**, ognuno di essi specifica determinati valori da inserire nello stack che permettono di usare queste sequenze di istruzioni. Essi sono delle istruzioni come load, add o jump. Questa tecnica permette all'attaccante di eseguire codice in presenza di difese di sicurezza come *executable space protection*¹ e *code signing*²

Esempio gadget. Il gadget che vediamo in questo esempio è `pop %ebx; ret;`, costituito da due istruzioni. A sinistra possiamo vedere il suo corrispettivo nel linguaggio assembly il quale copia il valore costante `$0xdeadbeef` nel registro ebx e poi

¹Nella sicurezza del computer, la protezione dello spazio eseguibile contrassegna le regioni di memoria come non eseguibili, in modo tale che un tentativo di eseguire codice macchina in queste regioni causerà un'eccezione, [5]

²La firma del codice è il processo di firma digitale di eseguibili e script per confermare l'autore del software e garantire che il codice non sia stato alterato o danneggiato da quando è stato firmato,[4].

passa all’istruzione successiva grazie al puntatore eip, mentre la parte destra mostra il gadget equivalente.

EXAMPLE

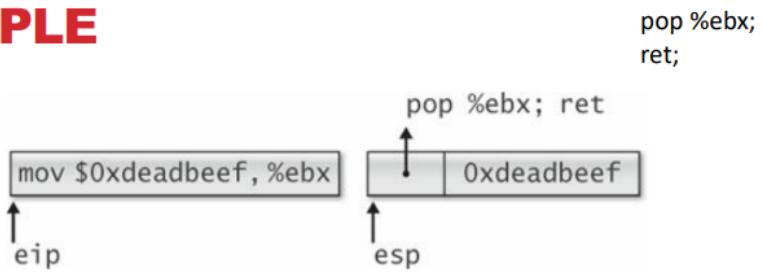


Figura 3.1: Esempio gadget.

Il gadget funziona nel seguente modo:

- fa una pop del valore `$0xdeadbeef` che si trova nello stack e lo inserisce nel registro ebx, facendo una pop viene diminuito anche la grandezza dello stack in base a quanto occupava quel valore in memoria. In questo caso si lavora con l’esp perchè si tiene conto della fine del frame che diminuisce;
- infine fa una ret che permette di eseguire il gadget successivo nello stack, come eip. Il nostro scopo è di creare una catena di gadget, composizione di varie operazioni, per l’attacco.

Esempio di attacco. L’obiettivo dell’attacco è di invocare la chiamata di sistema `ssize_t sys_write(unsigned int fd, const char * buf, size_t count)` in modo da stampare a schermo “xxxHACKEDxxx”. Questa istruzione prende in input un *file descriptor* in questo caso un `stdout`, una stringa e la dimensione della stringa da stampare.

```

int main(int argc, char *argv[]){
    char buf[4];
    gets(buf)
    return 0;
}

movl $4, %eax          # 4 is sys_write's id.
movl $1, %ebx          # 1 is stdout's device id
movl str_addr, %ecx   # address of "xxxHACKxxx"
                      # this string will be supplied
                      # by the attack
movl $13, %edx         # length of "xxxHACKxxx"
int  $0x80              # soft-interrupt to OS Kernel
                      # to invoke system call

```

Figura 3.2: Esempio attacco ROP.

Vediamo nel dettaglio come funziona, per inserire il codice nello stack possiamo fare un buffer overflow come abbiamo visto nell'esempio del code injection:

1. `movl $4, %eax`: inserisce l'identificatore della `sys_write` nel registro `eax`;
2. `movl $1, %ebx`: inserisce l'identificatore dello standard output `stdout` nel registro `ebx`;
3. `movl str_addr, %ecx`: copia l'indirizzo della stringa nel registro `ecx`;
4. `movl $13, %edx`: inserisce la grandezza della stringa nel registro `edx`, in questo modo abbiamo caricato nella CPU tutto quello che ci serve. Per prima mette la chiamata di sistema `sys_write` e poi tutti i suoi parametri che abbiamo visto in precedenza;
5. `int $0x80`: int interrompe l'esecuzione della CPU e salta al valore salvato in `eax` che è 4 ovvero la funzione `sys_write`.

Proviamo a fare lo stesso attacco con i gadget.

HOW TO DO IT

Buff has a lower address,
"xxxHACKEDxxx" has a higher address

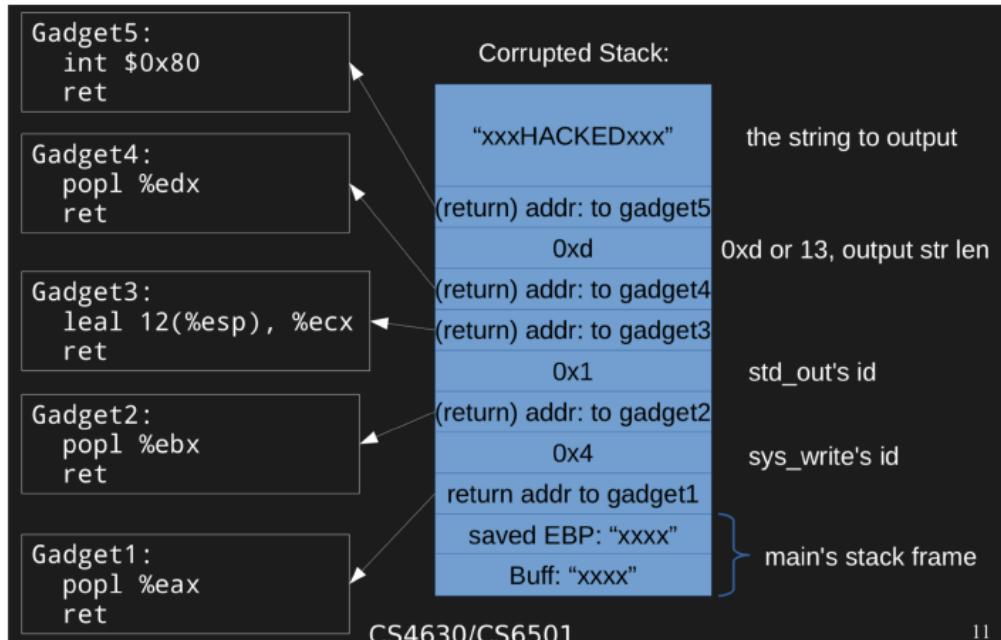


Figura 3.3: Esempio attacco ROP con gadget.

Nel caso di ROP, non c'è bisogno di scrivere un programma ma è necessario andare a trovare, all'interno della **libc** ad esempio, dei gadget equivalenti alle istruzioni viste in precedenza. Come si può vedere in Figura 3.3 partendo dal basso verso l'alto dove il buff ha un indirizzo più basso rispetto alla stringa "xxxHACKEDxxx", abbiamo i primi due spazi di memoria dedicati al main frame con l'ebp e il buffer, successivamente la prima istruzione da effettuare è la **ret** del Gadget1. La ret permette di eliminare tutto quello che ci stava prima di "0x4" compreso il "return addr to gadget1" per poi passare a eseguire la prossima istruzione del gadget1 che è **pop %eax**. Questa operazione è simile al funzionamento di eip ovvero esso contiene l'istruzione successiva da eseguire senza eliminare quello che è salvato prima. Facendo la pop si elimina a sua volta quello che sta contenuto nella porzione di memoria successiva al return addr del gadget1 ovvero 0x4 e lo inserisce nel registro eax. Si continua così per tutti gli altri gadget. Nel Gadget3 si inserisce l'indirizzo di memoria della stringa che vogliamo stampare che è **12(%esp)** ovvero 12 byte sotto rispetto all'esp, il quale punta dopo la ret all'inizio di "return addr: to gadget 4".

3.0.2 Qualche problema teorico

In generale se il file eseguibile è più grande di 3 MB c'è una buona probabilità che si può trovare un insieme di gadget per ogni exploit, più il file è piccolo più diminuisce la probabilità di trovare dei gadget. Inoltre non è sempre necessario usare la ret ma possiamo usare anche il jump o altre, i ROP possono lavorare anche senza **libc** ma utilizzando il codice fornito. La tecnica ROP fornisce un "linguaggio" (Turing complete) completamente funzionale che un utente malintenzionato può utilizzare in modo da far eseguire qualsiasi operazione desiderata a una macchina compromessa.

3.0.3 Come si fa exploit/previene

Il tool *ROPgadget* è uno strumento automatizzato per aiutare ad automatizzare il processo di individuazione dei gadget e costruzione di un attacco contro un file binario. Esso ricerca all'interno del file binario dei potenziali gadget utili e tenta di assemblerli in un payload di attacco che produce una shell. Un altro modo semplice di creare un attacco ROP è tramite il framework CTF(Capture The Flag) **pwnTools**³, scritto in Python è stato progettato per la prototipazione rapida e per rendere la scrittura di exploit il più semplice possibile.

³<http://docs.pwntools.com/en/stable/rop/rop.html>

3.1 Forme di mitigazione

Nella versione 4.1, GCC ha introdotto **Stack-Smashing Funzione Protector (SSP)**, che implementa i canarini derivati da StackGuard.

3.1.1 Stack-Smashing Protector

SSP o ProPolice è un'estensione di GCC per la protezione delle applicazioni scritte in C dalle più comuni forme di exploit di buffer overflow. Esso riordina le variabili locali in modo da mettere i buffer dopo i puntatori e copia i puntatori che si trovano negli argomenti delle funzioni in un'area che precede i buffer delle variabili locali in modo da evitare la corruzione dei puntatori.

3.1.2 I canarini

I canarini consistono in un valore difficile da inserire o falsificare e sono scritti in un indirizzo prima della sezione di a pila da proteggere. Di conseguenza, una scrittura sequenziale dovrebbe sovrascrivere questo valore per arrivare alla regione protetta.

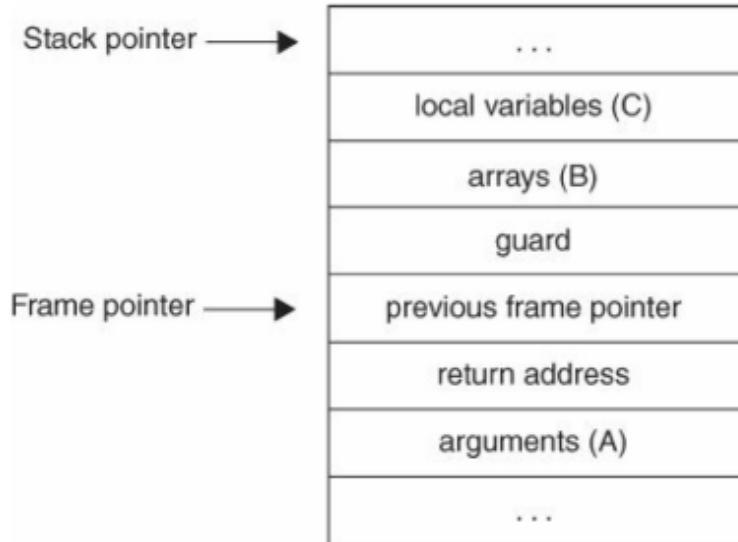


Figura 3.4: Canarino nello stack.

Come si può vedere in Figura 3.4, il canarino viene inizializzato dopo che il return address è salvato ed è verificato immediatamente prima di accedere a quest'ultimo. Un canarino casuale o difficile da falsificare è un numero casuale segreto a 32-bit che cambia ogni volta che il programma viene eseguito. Le opzioni `-fstack-protector` e `-fno-stack-protector` abilitano o disabilitano SSP per la protezione di oggetti vulnerabili.

3.1.3 Address Space Layout Randomization

L'ASLR è una caratteristica della sicurezza di molti sistemi operativi, il suo scopo è di evitare l'esecuzione di codice arbitrario. Essa permette di randomizzare gli indirizzi delle pagine di memoria usate dal programma. Però ASLR non previene la sovrascrittura del return address da parte di un overflow basato sullo stack. In ogni caso esso potrebbe prevenire la predizione corretta da parte degli attaccanti dell'indirizzo dello shellcode, delle funzioni di sistema dei gadget ROP che vogliono invocare.

3.1.4 Stack non-eseguibile

Uno stack non eseguibile è una soluzione runtime (a tempo di esecuzione) che è stata progettata per prevenire l'esecuzione di codice eseguibile nel segmento dello stack. Esso previene il buffer overflow solo sullo stack non sull'heap, inoltre non prevengono il fatto di poter usare l'overflow per un modificare un indirizzo di ritorno, un puntatore ad un oggetto o ad una funzione. Non prevengono la code o arc injection o la ROP.

3.1.5 W xor X

Molti sistemi operativi, come OpenBDS, Windows, Linux e OS X, impone privilegi ridotti nel kernel così che nessuna parte dello spazio degli indirizzi del processo è sia scrivibile che eseguibile. Questa politica è chiamata W xor X ($W \oplus X$) ed è supportato dall'uso di un bit No eXecute (NX) attivo diverse CPU.

Capitolo 4

Heap Overflows

4.0.1 Alcuni problemi dello heap

Il problema riguardante la heap memory occorre quando essa non viene adeguatamente liberata dopo che non è più necessaria. Le memory leaks possono essere problematiche in processi a lungo termine o in attacchi di esaurimento delle risorse. La memoria può essere **esausta** quando un malintenzionato identifica delle azioni esterne che possono allocare memoria ma non liberarla. Di conseguenza le allocazioni successive falliscono e l'applicazione è incapace di processare delle richieste valide dello user senza **crashare**. Inoltre è possibile accedere alla memoria liberata a meno che tutti puntatori che puntano a quella memoria sono settati a NULL o sovrascritti. Perciò quando si libera, bisogna impostare anche il puntatore alla memoria liberata a NULL.

Dereferencing Null or Invalid Pointers. Se l'operando non punta a un oggetto o una funzione, il comportamento dell'operatore unario `*` non è definito.

Double free. Consiste nel liberare lo stesso blocco di memoria più di una volta.

4.1 Heap overflow

4.1.1 Dlmalloc

Nella dlmalloc, i blocchi di memoria (chunks) sono sia allocati a un processo che liberi. I primi 4 byte dei chunk allocati e liberi contengono la dimensione del precedente

blocco adiacente, se è libero, ovvero gli ultimi 4 byte di dati utente del precedente pezzo, se è allocato.

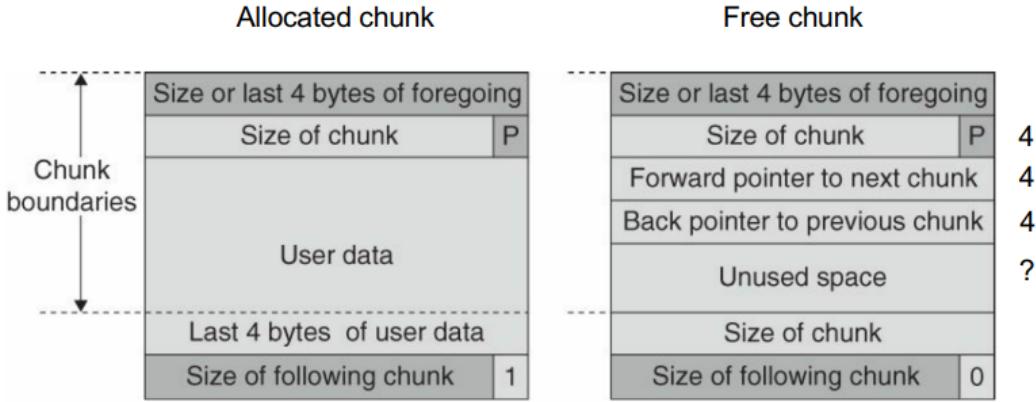


Figura 4.1: Chunck libero e allocato.

Chunck liberi

In dlmalloc, i blocchi liberi sono disposti in linked list¹ circolari a doppio collegamento, detti anche **bin**. Ogni linked list a doppio collegamento ha un'intestazione che contiene un puntatore in avanti e indietro rispettivamente al primo e all'ultimo blocco nella lista. Sia il puntatore in avanti dell'ultimo chunck che quello all'indietro nel primo chunck della lista punta all'elemento testa. Quando la lista è vuota, i puntatori della testa fanno riferimento alla testa stessa.

Bin

Ogni bin ha una *head*(testa) che contiene il puntatore in avanti e indietro che puntano rispettivamente al primo e all'ultimo blocco nella lista. Sia il chunck allocato che quello libero fanno uso di un bit **PREV_INUSE** (rappresentato da P 4.2) che indica se il chunck precedente è allocato o meno.

¹In informatica, una lista concatenata (o linked list) è una struttura dati dinamica, tra quelle fondamentali usate nella programmazione. Consiste di una sequenza di nodi, ognuno contenente campi di dati arbitrari ed uno o due riferimenti ("link") che puntano al nodo successivo e/o precedente, [3].

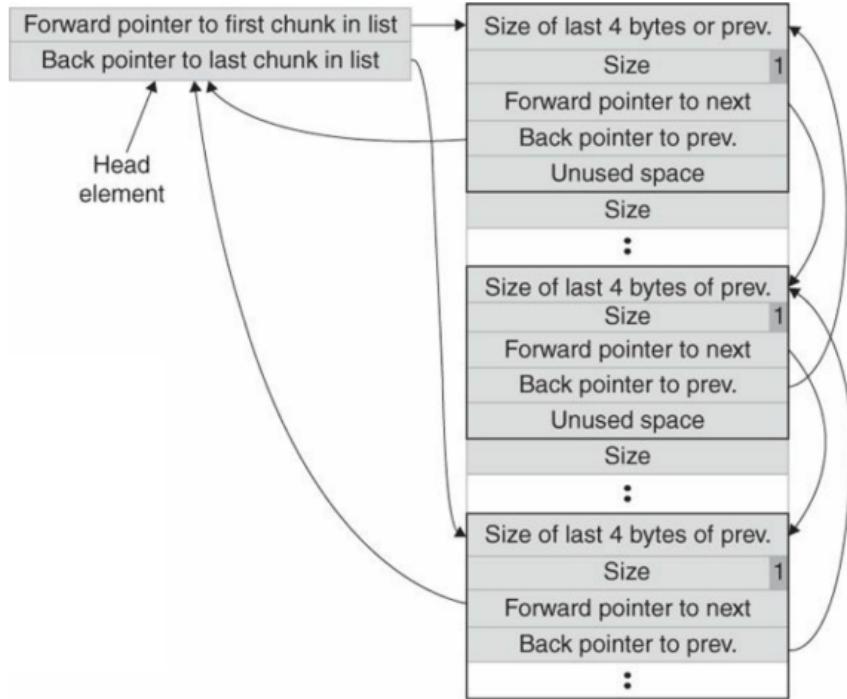


Figura 4.2: Esempio di un bin.

Sia il puntatore in avanti nell'ultimo chunck che quello indietro nel primo chunck della lista puntano alla testa. Quando la lista è vuota, la testa del puntatore si riferisce direttamente a se stessa. Ogni

UNLINK

`unlink()` è una macro usata per rimuovere un chunck dalla sua lista doppiamente linkata. Essa è usata quando la memoria è consolidata e quando un chunck è tolto della lista libera perché è stato allocato da un utente.

```
#define unlink(P, BK, FD) {
    FD = P -> fd;
    BK = P -> bk;
    FD -> bk = BK;
    BK -> fd = FD;
}
```

Funzionamento macro. Dalla Figura 4.3 si può capire bene il funzionamento della macro, essa prende in input tre puntatori

- **P** puntatore al blocco da rimuovere;
- **BK** puntatore al blocco precedente;
- **FD** puntatore al blocco successivo.

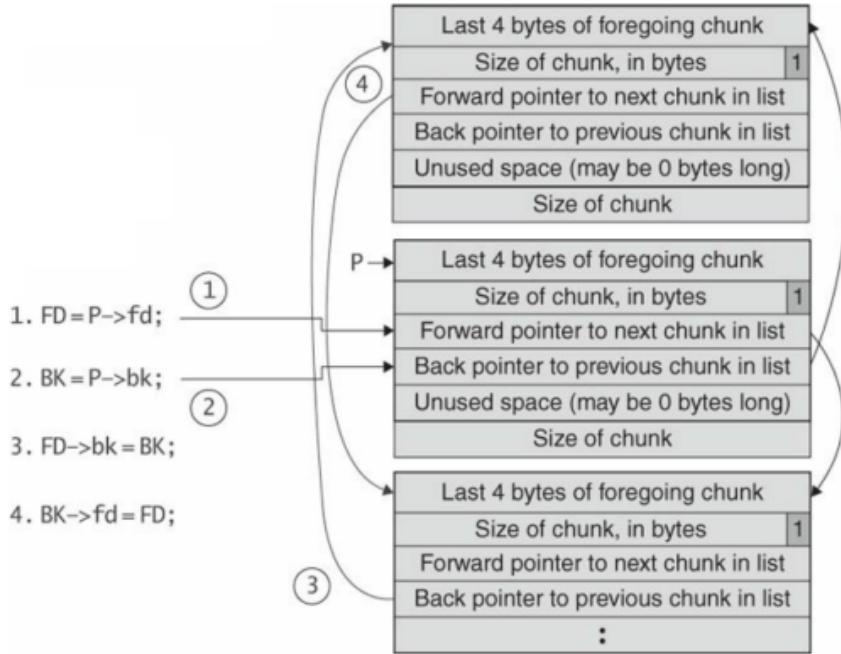


Figura 4.3: Funzionamento macro `unlink`.

In Figura 4.3 possiamo vedere un esempio del funzionamento della macro. Come accennato in precedenza il puntatore P si riferisce al chunck da togliere, esso contiene due puntatori uno che punta al blocco precedente e uno a quello successivo. Nel primo step della `unlink()` si assegna FD in modo da farlo puntare al chunck successivo nella lista rispetto a quello indicato da P. Facciamo la stessa cosa nel secondo step solo che assegniamo a BK il puntatore al chunck precedente. Nel terzo step, il puntatore in avanti (FD) sostituisce il puntatore all'indietro del blocco successivo nella lista con il puntatore al blocco che precede quello che è stato scollegato. Nell'ultimo step il puntatore all'indietro (BK) sostituisce il puntatore in avanti del precedente chunck nella lista con il puntatore al blocco successivo.

4.1.2 Tecnica unlink

La tecnica unlink è stata introdotta la prima volta da Solar Designer e usata con successo contro alcune versioni dei browser di Netscape, traceroute e slocate che utilizzavano dlmalloc. Questa tecnica è usata per fare un buffer overflow in modo da manipolare i tag di confine su un chunck di memoria per ingannare la macro `unlink()` facendole scrivere 4 byte di dati in una zona arbitraria.

```
01 #include <stdlib.h>
02 #include <string.h>
03 int main(int argc, char *argv[]) {
04     char *first, *second, *third;
05     first = malloc(666);
06     second = malloc(12);
07     third = malloc(12);
08     strcpy(first, argv[1]);
09     free(first);
10     free(second);
11     free(third);
12     return(0);
13 }
```

Figura 4.4: Esempio di codice vulnerabile a tecnica unlink.

Il programma vulnerabile alloca 3 chunck di memoria (riga 5-7). Il programma accetta una singola stringa come argomento che è copiata all'interno della malloc *first* (linea 8). Questa operazione strcpy() illimitata è soggetta a un buffer overflow. Il tag di confine può essere sovrascritto da un argomento stringa che supera la lunghezza di *first* perché il tag di confine per il secondo si trova direttamente dopo il primo buffer. Il problema di questo programma accade alla seconda free (riga 10). Vediamo come è strutturato l'heap prima di fare la seconda free.

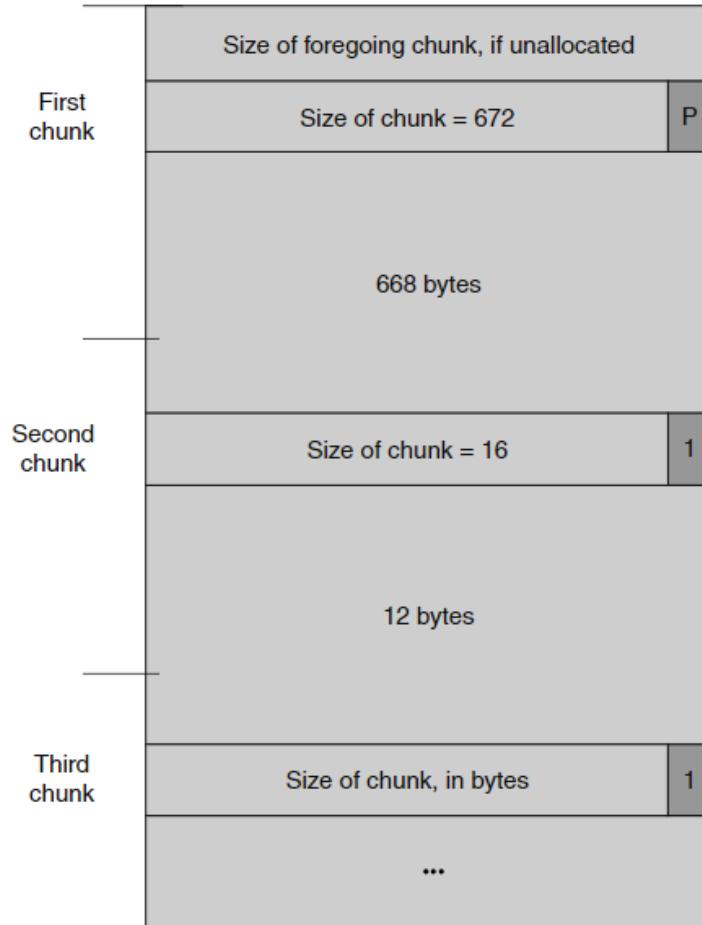


Figura 4.5: Contenuto dell’heap alla prima chiamata di `free()`.

Se il secondo blocco non è allocato, l’operazione `free()` prova a consolidarlo con il primo blocco. Per determinare se il secondo chunck è allocato o meno bisogna guardare il `PREV_INUSE` bit del terzo blocco. La locazione di ogni blocco è determinata aggiungendo la grandezza del blocco all’indirizzo iniziale. Durante le operazioni normali, il bit P del terzo chunck è settato perché il secondo chunck è allocato come si vede in Figura 4.5. Poiché il buffer vulnerabile è allocato nell’heap e non nello stack, l’attaccante non può solamente sovrascrivere l’indirizzo di ritorno per sfruttare la vulnerabilità ed eseguire codice malevolo. L’attaccante può sovrascrivere i boundary tag associati con il secondo chunck della memoria, perché questo tag di confine è collocato immediatamente dopo la fine del primo blocco. La grandezza del primo chunck (672 byte) è il risultato della grandezza richiesta di 666 byte, più 4 byte per la

grandezza, arrotondato al multiplo più vicino a 8 poiché tutti chunck devono essere divisibili per 8 byte.

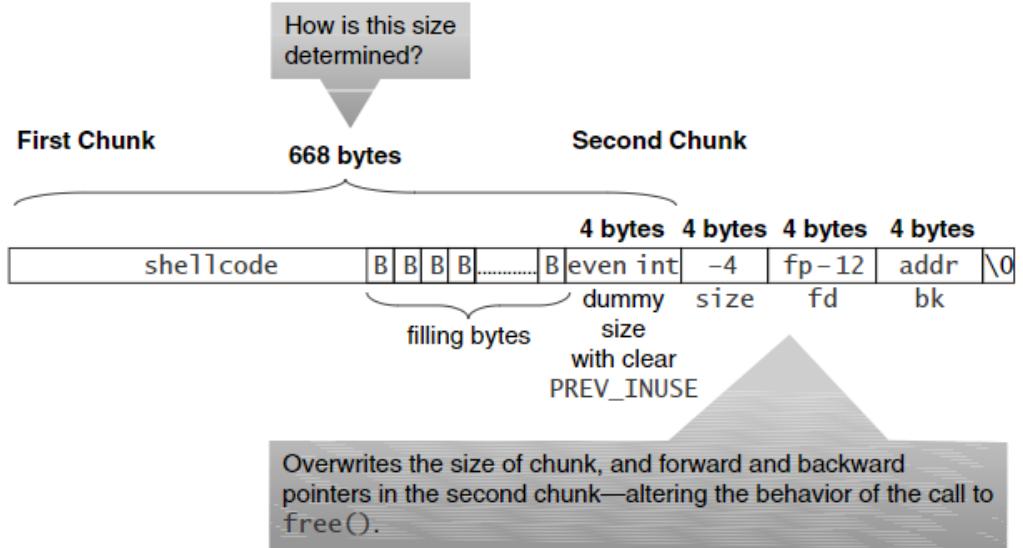


Figura 4.6: Funzionamento tecnica unlink.

Come si vede in Figura 4.6 un argomento malevolo può essere usato per sovrascrivere i tag del secondo chunck. Questo argomento sovrascrive il campo della dimensione del precedente blocco, grandezza del chunck, e i puntatori in avanti e indietro del secondo chunck, alterando così il comportamento della free(). In particolare il campo per la dimensione è modificato inserendo come valore -4 byte, in questo modo quando la free() prova a determinare la locazione del terzo chunck aggiungendo la grandezza appena modificata all'indirizzo iniziale del secondo chunck invece di aggiungere sottrae 4 byte. Così la dlmalloc pensa che l'inizio del successivo chunck è 4 byte prima dell'inizio del secondo chunck. L'argomento malevolo garantisce che la collocazione dove la dlmalloc trova il bit P è libera, ingannando la dlmalloc facendole credere che il secondo chunck non è allocato così l'operazione di free() invoca la unlink macro per consolidare i due blocchi liberi consecutivi. Come si vede in Figura 4.6 viene inserito al posto della grandezza effettiva del chunck un numero pari finto, deve essere pari poiché l'ultimo bit deve essere 0. In fd inseriamo fp-12 che è l'indirizzo dove voglio compiere l'attacco. In bk c'è il dato che vogliamo scrivere all'indirizzo fd. Per come è scritta la unlink, è lei stessa che fa questa cosa quando viene chiamata la

seconda free. Noi scriviamo solo il payload, poi il lavoro lo fanno free e di conseguenza unlink. L'obiettivo è scrivere l'indirizzo addr in fp

Capitolo 5

Pointer subterfuge

Il **pointer subterfuge** è un termine generale per gli exploit che modificano il valore di un puntatore. In C e C++ esistono sia i puntatori a funzioni che i puntatori a variabili.

5.0.1 Puntatori a funzioni

Nel caso dei puntatori a funzioni il valore di quest'ultimi può essere sovrascritto in modo da trasferire il controllo a una shellcode fornita dall'attaccante. Quando un programma esegue una chiamata tramite il puntatore alla funzione, il codice dell'attaccante è eseguito al posto del codice previsto.

```
1 void good_function(const char *str) {...}
2 int main(int argc, char *argv[]) {
3     static char buff[BUFFSIZE];
4     static void (*funcPtr)(const char *str);
5     funcPtr = &good_function;
6     strncpy(buff, argv[1], strlen(argv[1]));
7     (void)(*funcPtr)(argv[2]);
8 }
```

Overflow in the data segment!!!!

Shellcode can be pointed by funcPtr!!

Figura 5.1: Esempio pointer subterfuge su puntatori a funzioni.

Nel caso della Figura 5.1 il problema si trova nelle righe 3-4 poiché non sappiamo la grandezza di `BUFFSIZE` e questo può portare a un buffer overflow sovrascrivendo il puntatore a funzione `*funcPtr`. Abbiamo visto in precedenza il buffer overflow applicato nello stack e nell’heap , in questo caso andiamo a lavorare nella parte Data in cui vengono salvate le variabili sia globali che statiche. Quindi le nostre variabili static `buff[BUFFSIZE]` e `*funcPtr` sono salvate, presumibilmente una sotto l’altra, all’interno della parte Data della memoria. In questo modo effettuando un buffer overflow si va a sovrascrivere il puntatore alla funzione `good_function` (riga 5), in seguito facendo uno `strncpy` inserendo più caratteri di quelli consentiti e richiamando la funzione tramite il suo puntatore (riga 7) non si chiama effettivamente la `good_function` ma quello che abbiamo inserito noi.

5.0.2 Puntatore a oggetti

Lo stesso metodo può essere applicato ai puntatori a variabili.

```

1 void foo(void * arg, size_t len) {
2     char buff[100];
3     long val = ...;
4     long *ptr = ...;
5     memcpy(buff, arg, len);
6     *ptr = val;
7     ...
8     return;
9 }
```

Figura 5.2: Esempio pointer subterfuge su puntatori a variabili.

In questo caso abbiamo il `buff[100]` di grandezza 100 caratteri seguito da una variabile di tipo long e un puntatore sempre long. Nell’esempio di prima andavamo a fare una `strncpy` che portava al buffer overflow, qui invece facciamo una `memcpy(buff, arg, len)`¹ che è sempre una copia non controllata da una zona sorgente a una destinazione. Successivamente a riga 6 assegnando `*ptr = val` scriviamo un

¹`void * memcpy (void * destination, const void * source, size_t num)` Copia i valori dei byte di `num`, indica il numero di byte da copiare, dalla posizione a cui punta `source` direttamente nel blocco di memoria a cui punta `destination`.

valore che vogliamo noi, poiché tramite il buffer overflow su `buff[100]` eccediamo della grandezza massima e andiamo a sovrascrivere i valori delle successive variabili a riga 3-4, in una zona di memoria che vogliamo noi quindi viene eseguita una **scrittura di memoria arbitraria**.

Ultimo esempio Questo ultimo esempio differenzia da quelli precedenti per il modo in cui vengono chiamate le funzioni, esse possono essere chiamate in modo diretto o indiretto.

```
01 void good_function(const char *str) {
02     printf("%s", str);
03 }
04
05 int main(void) {
06     static void (*funcPtr)(const char *str);
07     funcPtr = &good_function;
08     (void)(*funcPtr)("hi ");
09     good_function("there!\n");
10     return 0;
11 }
```

Figura 5.3: Esempio pointer subterfuge.

A riga 9 in Figura 5.3 possiamo vedere un esempio di chiamate diretta, mentre a riga 8 abbiamo una indiretta poiché `good_function` è chiamata tramite il puntatore a quest'ultima. In Figura 5.4 vediamo a livello assembly la differenza fra le due chiamate, la chiamate diretta è più semplice perché viene effettuata, oltre la push per inserire nello stack la stringa da passare alla funzione, direttamente una call a `good_function`. Al contrario la chiamata indiretta è un pò più complessa visto che non si fa una call diretta a `good_function` ma si chiama il puntatore a esse `funcPtr` che contiene l'indirizzo dove è salvata la funzione chiamata. L'istruzione di chiamata, ad esempio, salva l'informazione di ritorno sullo stack e trasferisce il controllo alla chiamata di funzione specificata dall'operando di destinazione (target). Il target specifica l'indirizzo della prima istruzione nella funzione chiamata.

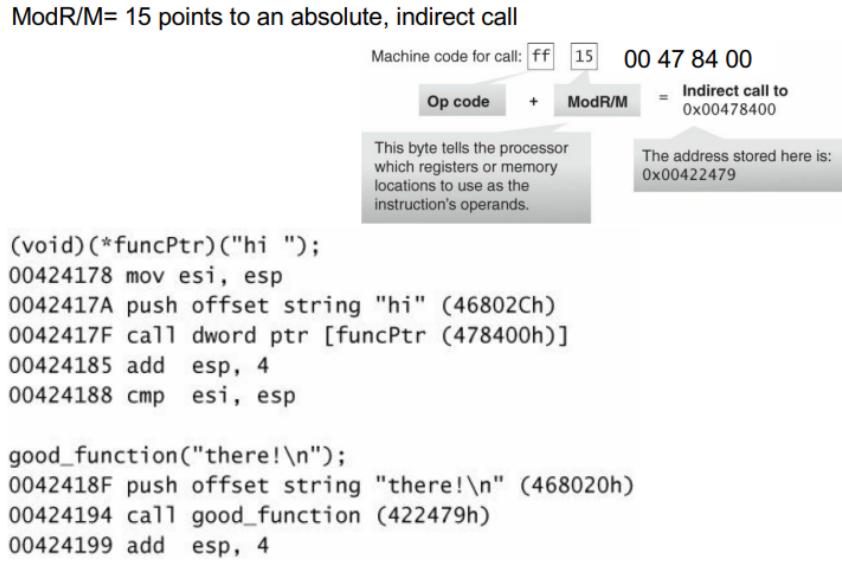


Figura 5.4: Disassembly delle chiamate a **good_function**.

Questo operando può essere un valore immediato, un registro generico, o una posizione in memoria. Queste invocazioni di **good_function()** forniscono esempi di istruzioni di chiamata che possono e non possono essere attaccate. L'invocazione statica utilizza un valore immediato come relativo spostamento, e questo spostamento non può essere sovrascritto perché è nel segmento di codice. La chiamata tramite il puntatore alla funzione utilizza un riferimento indiretto e l'indirizzo nella posizione di riferimento può essere (in genere nel data o nel segmento dello stack) sovrascritto.

Capitolo 6

IDManag

6.1 intro

Il **furto di identità** si verifica quando qualcuno utilizza le informazioni personali di identificazione di un'altra persona, come il nome, il numero di identificazione o il numero di carta di credito, senza il suo permesso, per commettere frodi o altri crimini.

6.1.1 Identità digitale

Un'**identità digitale** è un'informazione su un'entità utilizzata dai sistemi informatici per rappresentare un agente esterno. Tale agente può essere una persona, un'organizzazione, un'applicazione o un dispositivo.

ISO/IEC 24760-1 definisce l'**identità** come "insieme di attributi relativi a un'entità".

Le informazioni contenute in un'**identità digitale** consentono di valutare e autenticare un utente che interagisce con un sistema aziendale sul web/Network, senza il coinvolgimento di operatori umani.

6.1.2 Autenticazione

L'autenticazione è un aspetto fondamentale dell'attribuzione di identità basata sulla fiducia, in quanto fornisce una garanzia codificata dell'identità di un'entità a un'altra.

Le **metodologie di autenticazione** includono la presentazione di un **oggetto univoco** come una carta di credito bancaria, le **informazioni riservate** come una

una password o la risposta a una domanda prestabilita, **la conferma della proprietà di un indirizzo e-mail**, e soluzioni più robuste ma relativamente costose utilizzando metodologie di crittografia.

6.1.3 Autorizzazione

L'autorizzazione è la determinazione di qualsiasi entità che controlla le risorse che l'autenticato può accedere a tali risorse.

L'autorizzazione dipende dall'autenticazione, perché l'autorizzazione richiede la verifica dell'attributo critico.

6.2 Identity

I concetti di **identità**, **identificatore** e **account** sono strettamente correlati ma diversi.

6.2.1 Identifier

Il termine "**identificatore**" si riferisce a un singolo attributo il cui scopo è quello di identificare in modo univoco una persona o un'entità, all'interno di un contesto specifico. (indirizzo email, numero passaporto ecc...)

6.2.2 Identità

Il termine "identità" è definito come un insieme di attributi associati a una specifica persona/entità in un particolare contesto. Un'identità comprende uno o più identificatori e può contenere altri attributi associati a una persona/entità.

Attributi

Le identità umane possono includere attributi come il nome, età, indirizzo, numero di telefono, colore degli occhi e titolo di lavoro. Le identità non umane possono includere attributi come il proprietario, l'indirizzo IP e forse un numero di modello o di versione.

Gli attributi che compongono un'identità possono essere utilizzati per l'autenticazione e l'autorizzazione, oltre che per trasmettere informazioni sull'identità alle applicazioni.

Un'identità online consiste in almeno un identificatore e un insieme di attributi per un utente/entità in un particolare contesto, come un'applicazione o una suite di applicazioni.

6.2.3 Account

Un'identità è associata a un account in ciascuno di questi contesti.

Definiamo un account come un costrutto locale all'interno di una data applicazione o suite di applicazioni che viene utilizzato per eseguire azioni all'interno di quel contesto.

Gli attributi di identità possono essere contenuti account object di un'applicazione, oppure possono essere memorizzati separatamente e referenziati dall'oggetto conto.

6.2.4 Separazione tra ID e account

Un account può avere un proprio identificativo oltre a quello dell'identità ad esso associata. Avere un identificatore dell'account separato dall'identità associato all'account fornisce un grado di separazione.

L'identificativo dell'account può essere utilizzato in altri record dell'applicazione per rendere più facile per gli utenti cambiare il nome utente o altro identificatore associato al proprio account.

Si noti che un account può avere più di un'identità associata ad esso attraverso l'account linking.

6.2.5 Non Human Identifier

Anche gli attori non umani possono certamente avere un'identità. I componenti software che fungono da agenti o bot e i dispositivi intelligenti possono avere un'identità e possono interagire con altri software o dispositivi in modi che richiedono autenticazione e autorizzazione, proprio come gli attori umani

6.2.6 IDM System

Un sistema di gestione delle identità (IdM) è un insieme di servizi che supportano la creazione, la modifica e la rimozione delle identità e degli account associati, nonché l'autenticazione e l'autorizzazione necessarie per accedere alle risorse.

I sistemi di gestione dell'identità sono utilizzati per proteggere risorse online da accessi non autorizzati e costituiscono un parte importante di un modello di sicurezza completo.

6.3 Eventi in un ciclo di vita di un'identità

6.3.1 Provisioning

L'atto di creare un account e le relative informazioni di identità è spesso indicato come provisioning. L'obiettivo della fase di provisioning è quello di stabilire un account con i relativi dati di identità.

Si tratta di ottenere o assegnare un identificativo univoco per l'identità, optionalmente un identificativo univoco per l'account distinto da quello dell'identità, creare un account e associare gli attributi del profilo dell'identità all'account.

6.3.2 AUTHORIZATION

Quando si crea un account, spesso è necessario specificare cosa può fare l'account, sotto forma di privilegi.

Il termine autorizzazione indica la concessione di privilegi che regolano le attività di un account.

L'autorizzazione di un account viene generalmente effettuata al momento della sua creazione e può essere aggiornata nel tempo.

6.3.3 AUTHENTICATION

L'utente fornisce un identificativo per indicare l'account che desidera utilizzare e inserisce le credenziali di accesso per l'account.

Queste vengono convalidate rispetto alle credenziali precedentemente registrate durante la fase di provisioning dell'account.

Le credenziali possono riguardare qualcosa che l’utente conosce, qualcosa che l’utente possiede e/o qualcosa che l’utente è.

Il nome utente indica l’account che l’utente desidera utilizzare e la conoscenza della password dimostra il suo diritto a utilizzare l’account.

6.3.4 ACCESS POLICY ENFORCEMENT

L’autorizzazione specifica ciò che un utente o un’entità può fare e l’applicazione dei criteri di accesso verifica che le azioni richieste da un utente siano consentite dai privilegi che è stato autorizzato a utilizzare.

Per assicurarsi che le azioni intraprese dall’utente siano consentite dai privilegi privilegi che gli sono stati concessi

Un’applicazione potrebbe visualizzare un messaggio che indica che un utente non è autorizzato a visualizzare un particolare servizio.

6.3.5 Sessioni

Alcune applicazioni, in genere le applicazioni Web tradizionali e le applicazioni sensibili, consentono a un utente di rimanere attivo solo per un periodo di tempo limitato prima di richiedere all’utente di autenticarsi nuovamente. (Una sessione tiene traccia delle informazioni)

Le impostazioni di timeout della sessione variano in genere in base alla sensibilità dei dati dell’applicazione.

6.3.6 Single Sign-on

Dopo aver effettuato l’accesso a un’applicazione, l’utente potrebbe voler effettuare un’altra operazione con un’altra applicazione.

Il single sign-on (SSO) è la possibilità di effettuare il login una volta e poi accedere ad altre risorse o applicazioni protette con gli stessi requisiti di autenticazione, senza dover reinserire le credenziali.

Il single sign-on è possibile quando un insieme di applicazioni ha delegato l’autenticazione alla stessa entità.

6.3.7 STRONGER AUTHENTICATION

L'autenticazione step-up è l'atto di elevare una sessione di autenticazione esistente a un livello di garanzia più elevato mediante autenticazione con una forma di autenticazione più forte.

Ad esempio, un utente potrebbe inizialmente accedere con un nome utente e una password per stabilire una sessione di autenticazione.

In seguito, quando accede a una funzione o a un'applicazione più sensibile con requisiti di autenticazione più elevati, all'utente vengono richieste ulteriori credenziali, ad esempio una password una tantum generata sul suo telefono cellulare.

6.3.8 Logout

Come minimo, l'atto di disconnettersi dovrebbe terminare la sessione dell'applicazione dell'utente.

Se l'utente ritorna all'applicazione, dovrà autenticarsi nuovamente prima di poter accedere.

In situazioni in cui si utilizza il single sign-on, potrebbero esserci più sessioni da terminare.

È una decisione di progettazione decidere quali sessioni debbano essere terminate quando l'utente esce da un'applicazione.

6.3.9 ACCOUNT MANAGEMENT AND RECOVERY

Nel corso della vita di un'identità, può essere necessario modificare vari attributi del profilo utente dell'identità.

Ad esempio, un utente potrebbe dover aggiornare il proprio indirizzo e-mail o numero di telefono, la password e il nome. In un'azienda, il profilo di un utente può essere aggiornato per riflettere una nuova posizione, un nuovo indirizzo o nuovi privilegi come i ruoli.

Il recupero dell'account è un meccanismo per convalidare che un utente sia il legittimo proprietario di un account attraverso alcuni mezzi secondari e quindi consentire all'utente di stabilire nuove credenziali.

Ripristino della password smarrita via e-mail

6.3.10 DEPROVISIONING

Può capitare che sia necessario chiudere un account.

In questo caso, l'account dell'utente e le informazioni di identità associate devono essere deprovisionate in modo che non possano più essere utilizzate.

La deprovisioning può consistere nell'eliminazione completa dell'account e delle informazioni di identità associate o nella semplice disabilitazione dell'account, per conservare le informazioni a fini di revisione.

Parte II

Appunti parte Prof.Bistarelli

Capitolo 1

Cybersecurity

1.1 Introduzione alla sicurezza informatica

Il rapporto interno/interagenzia NIST NISTIR 7298 (Glossario di informazioni chiave Termini di sicurezza, maggio 2013) definisce il termine sicurezza informatica come segue:

Misure e controlli che garantiscono riservatezza, integrità, e disponibilità delle risorse del sistema informativo inclusi hardware, software, firmware, e le informazioni che vengono elaborate, archiviate e comunicate.

Questa definizione introduce tre obiettivi chiave che sono al centro della cybersecurity:

- **Confidentiality** (Riservatezza): conservazione delle restrizioni autorizzate all’accesso alle informazioni e divulgazione, compresi i mezzi per proteggere la privacy personale e le proprie informazioni. Una perdita di riservatezza è la divulgazione non autorizzata di informazioni. Questo termine copre due concetti correlati:
 - **Data Confidentiality** : garantisce che le informazioni private o riservate non siano disponibili o divulgare a soggetti non autorizzati.
 - **Privacy**: assicura che le persone controllino o influenzino le informazioni ad essi relativi, esse possono essere raccolte e conservate, inoltre si definisce da chi e a chi possono essere divulgare.

- **Integrity** (Integrità): prevenire la modifica o la distruzione impropria delle informazioni, compresa la garanzia del non ripudio e dell'autenticità delle informazioni. Una perdita di l'integrità è la modifica o la distruzione non autorizzata di informazioni. Questo termine copre due concetti correlati:
 - **Data integrity**: garantisce che le informazioni e i programmi vengano modificati solo in modo determinato e autorizzato.
 - **System integrity**: assicura che un sistema svolga la sua funzione prevista in modo inalterato, libero da intenzionali o involontarie manipolazioni non autorizzate del sistema.
- **Availability**(Disponibilità): garantisce un accesso tempestivo e affidabile nell'utilizzo delle informazioni. Una perdita di disponibilità è l'interruzione dell'accesso o dell'uso di informazioni o un sistema informativo.

Questi tre concetti formano quella che viene spesso definita la triade della CIA. I tre concetti incarnano gli obiettivi di sicurezza fondamentali sia per i dati che per le informazioni e servizi informatici. Ad esempio, lo standard FIPS 199 del NIST (Standards for Security Categorization of Federal Information and Information Systems , febbraio 2004) elenca la riservatezza, integrità e disponibilità come i tre obiettivi di sicurezza per le informazioni e per i sistemi informativi.

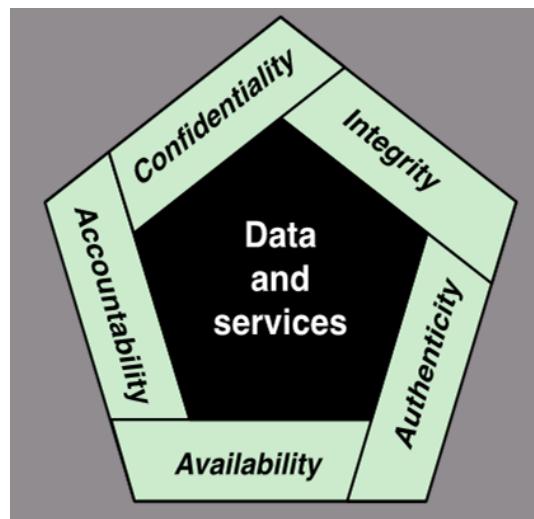


Figura 1.1: Requisiti essenziali in Cybersecurity.

Sebbene l'uso della triade della CIA per definire gli obiettivi di sicurezza sia ben consolidato, alcuni nel campo della sicurezza ritengono che siano necessari concetti aggiuntivi da presentare un quadro completo. Due dei più comunemente citati sono i seguenti:

- **Authenticity** (Autenticità): la proprietà di essere genuini e di poter essere verificati e di essere trusted. Fiducia nella validità di una trasmissione di un messaggio o di un messaggio originatore. Ciò significa verificare che gli utenti siano chi dicono di essere e che ogni input che arriva al sistema proviene da una fonte attendibile.
- **Accountability**(Rendicontabilità): è la capacità di un sistema di identificare un singolo utente, di determinarne le azioni e il comportamento all'interno del sistema stesso. La rendicontabilità è un aspetto del controllo di accesso e si basa sulla concezione che gli individui siano responsabili delle loro azioni all'interno del sistema. Questo supporta il non ripudio, deterrenza, isolamento dei guasti, rilevamento e prevenzione delle intrusioni, il recupero post-azione in concomitanza con l'azione legale . Poiché i sistemi veramente sicuri non sono ancora un obiettivo realizzabile, dobbiamo essere in grado di tracciare una violazione della sicurezza al/ai responsabile/i. I sistemi devono tenere traccia delle loro attività per consentire successive analisi forensi per rintracciare violazioni della sicurezza o per aiutare nelle controversie sulle transazioni.

Si noti che FIPS 199 include l'autenticità sotto integrità.

La sicurezza informatica è allo stesso tempo affascinante e complessa, alcuni dei motivi sono:

1. *La sicurezza informatica non è così semplice come potrebbe sembrare a un principiante.* I requisiti sembrano essere semplici,in effetti, la maggior parte dei requisiti principali per i servizi di sicurezza possono essere definiti con etichette formate autoesplicative formate da una sola parola: riservatezza, autenticazione, non ripudio e integrità. Ma i meccanismi utilizzati per soddisfare tali requisiti possono essere piuttosto complessi, e capirli può portare a un ragionamento piuttosto sottile.
2. *Nello sviluppo di un particolare meccanismo di sicurezza o algoritmo, bisogna sempre considerare potenziali attacchi a tali funzionalità di sicurezza.* In molti

casi gli attacchi di successo sono progettati guardando un problema in un modo completamente differente, dunque sfruttando una debolezza inaspettata del meccanismo.

3. *A causa del punto 2 , le procedure usate per fornire dei servizi particolari sono spesso controintuitive.* Tipicamente, un meccanismo di sicurezza è complesso e non è ovvio dalle dichiarazioni di una particolare esigenza che tali misure elaborate sono necessarie. Solo quando si prendono in considerazione i vari aspetti della minaccia si elaborano i meccanismi di sicurezza hanno un senso.
4. *I meccanismi di sicurezza in genere coinvolgono più di un particolare algoritmo o protocollo.* Richiedono inoltre che i partecipanti siano in possesso di un'informazione segreta (ad es. una chiave di crittografia), che sollevano domande sulla creazione, distribuzione e protezione di tali informazioni segrete. Potrebbe esserci anche una dipendenza sui protocolli di comunicazione il cui comportamento può complicare il compito di sviluppare il meccanismo di sicurezza. Ad esempio, se il corretto funzionamento del meccanismo di sicurezza richiede la definizione di limiti di tempo per il tempo di transito di un messaggio dal mittente al destinatario, allora qualsiasi protocollo o rete che introduce variabili e/o ritardi imprevedibili può rendere tali termini privi di significato.
5. *La sicurezza informatica è essenzialmente una battaglia di ingegni tra un perpetratore che prova a trovare buchi e il progettista o l'amministratore che tenta di chiuderli.* Il grande vantaggio che l'attaccante ha è che lei o lui ha solo bisogno di trovare una singola vulnerabilità, mentre il progettista deve trovare e eliminare tutte le vulnerabilità per ottenere una sicurezza perfetta.
6. *La sicurezza è ancora troppo spesso un'"aggiunta" (surplus) per essere incorporata in un sistema dopo che il progetto è completo, piuttosto che essere parte integrante del processo di progettazione.*
7. *La sicurezza richiede un monitoraggio regolare, anche costante, e questo è difficile nei tempi attuali.*
8. *C'è una naturale tendenza da parte di utenti e gestori di sistema a percepire pochi vantaggi nell'investimento sulla sicurezza fino a quando non si verifica un problema.*

9. *Molti utenti e persino gli amministratori della sicurezza vedono una sicurezza forte come un ostacolo al funzionamento o all'uso efficiente di un sistema informativo o di un'informazione.*

1.1.1 Terminologie

La maggior parte delle terminologie sono riportate nel Capitolo 1 degli appunti Prof. Santini, di seguito riporto alcuni termini non citati in precedenza.

Risorsa di sistema (Asset)

Una applicazione maggiore, un sistema di supporto generale, un programma ad alto impatto, un impianto fisico, un sistema mission-critical, personale, apparecchiature o un gruppo di sistemi logicamente correlati.

Minaccia

Qualsiasi circostanza o evento che potrebbe avere un impatto negativo sulle operazioni organizzative (inclusi missione, funzioni, immagine o reputazione), risorse organizzative, individui, altre organizzazioni o la Nazione stessa attraverso un sistema informativo tramite accesso, distruzione, divulgazione, modifica non autorizzati delle informazioni , e/o negazione del servizio.

Contromisure

Dispositivo o tecniche che hanno come obiettivo la compromissione dell'efficacia operativa di attività indesiderate o contraddittorie, o la prevenzione di spionaggio, sabotaggio, furto o accesso o utilizzo non autorizzato di informazioni sensibili o di sistemi informativi.

Rischio

Una misura del grado in cui un'entità è minacciata da una potenziale circostanza o evento, e tipicamente una funzione di stima:

1. degli impatti negativi che si verificherebbero se la circostanza o l'evento si verificassero

2. della probabilità che si verifichi.

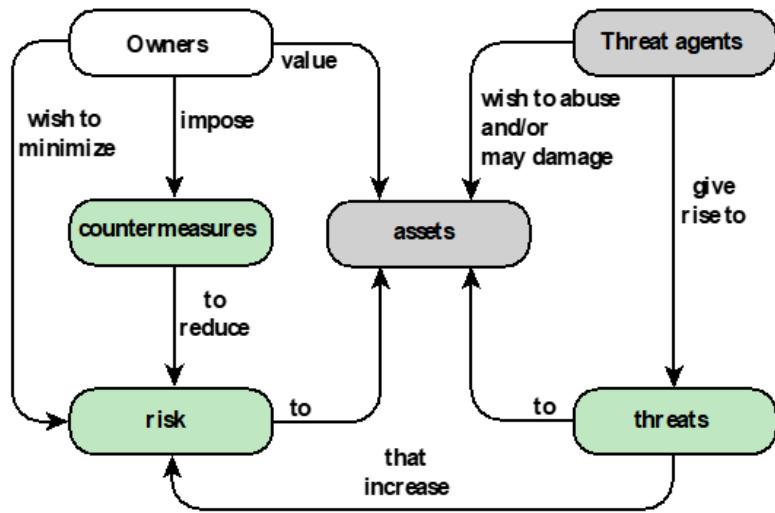


Figura 1.2: Concetti di sicurezza e le loro relazioni.

1.1.2 Asset di un sistema informatico

Gli asset di un sistema informatico possono essere suddivisi come di seguito:

- **Hardware:** compresi i sistemi informatici e altri trattamenti di dati, archiviazione dei dati,e dispositivi di comunicazione dati;
- **Software:** include il sistema operativo, le utilità di sistema e le applicazioni;
- **Data:** inclusi file e database, nonché dati relativi alla sicurezza, ad esempio file di password.
- **Strutture e reti di comunicazione:** rete locale e geografica collegamenti di comunicazione, bridge, router e così via.

1.1.3 Vulnerabilità, minacce e attacchi

Nel contesto della sicurezza, la nostra preoccupazione riguarda le vulnerabilità delle risorse del sistema. [NRC02] elenca le seguenti categorie generali di vulnerabilità di un sistema informatico o di una risorsa di rete:

- Il sistema può essere danneggiato (**corrupted**), quindi fa la cosa sbagliata o dà risposte sbagliate. Ad esempio, i valori dei dati memorizzati possono differire da quello che dovrebbero essere perché sono stati modificati in modo improprio.
- Il sistema avere delle perdite (**be leaky**). Ad esempio, qualcuno che non dovrebbe avere accesso ad alcune o a tutte le informazioni disponibili attraverso la rete ottengono tale accesso.
- Il sistema può diventare non disponibile (**unavailable**) o molto lento. Cioè, usando il sistema o la rete diventa impossibile o impraticabile.

Questi tre tipi generali di vulnerabilità corrispondono ai concetti di integrità, riservatezza e disponibilità, enumerati in precedenza. Una **minaccia** rappresenta un potenziale danno alla sicurezza di una risorsa. Un **attacco** è una minaccia che viene eseguita (azione di minaccia) e, in caso di successo, comporta una violazione indesiderata della sicurezza o a una conseguenza della minaccia. L'agente che effettua l'attacco viene definito **attaccante** o **agente di minaccia**. Possiamo distinguere gli attacchi in due tipi:

- **Attacco attivo**: un tentativo di alterare le risorse del sistema o di influenzare il funzionamento.
- **Attacco passivo**: un tentativo di imparare o di fare use delle informazioni da un sistema che non influenza le risorse di quest'ultimo.

Possiamo classificare gli attacchi in base all'origine di questi:

- **Attacco interno**: iniziato da un entità interna al perimetro di sicurezza (un "insider"). L'insider è autorizzato all'accesso alle risorse del sistema ma le usa in un modo non approvato da coloro che ne garantiscono l'accesso.
- **Attacco esterno**: iniziato fuori dal perimetro, da un utente non autorizzato o illegittimo del sistema (un "outsider"). Su Internet, potenziale aggressori esterni variano dai dilettanti "burloni" a criminali organizzati, internazionali terroristi e governi ostili.

Infine, una contromisura è qualsiasi mezzo adottato per affrontare un attacco alla sicurezza. Idealmente, una contromisura può essere escogitata per prevenire un

particolare tipo di attacco dall'avere successo. Quando la prevenzione non è possibile, o in alcuni casi fallisce, l'obiettivo è rilevare l'attacco e poi riprendersi dagli effetti. Una contromisura stessa può introdurre nuove vulnerabilità. In ogni caso, vulnerabilità residue possono rimanere dopo l'imposizione di contromisure. Tali vulnerabilità possono essere sfruttato da attaccanti che rappresentano un livello di rischio residuo per gli asset. I proprietari dell'asset cercheranno di ridurre al minimo tale rischio dati altri vincoli.

Threat Consequence	Threat Action (Attack)
Unauthorized Disclosure A circumstance or event whereby an entity gains access to data for which the entity is not authorized.	Exposure: Sensitive data are directly released to an unauthorized entity. Interception: An unauthorized entity directly accesses sensitive data traveling between authorized sources and destinations. Inference: A threat action whereby an unauthorized entity indirectly accesses sensitive data (but not necessarily the data contained in the communication) by reasoning from characteristics or by-products of communications. Intrusion: An unauthorized entity gains access to sensitive data by circumventing a system's security protections.
Deception A circumstance or event that may result in an authorized entity receiving false data and believing it to be true.	Masquerade: An unauthorized entity gains access to a system or performs a malicious act by posing as an authorized entity. Falsification: False data deceive an authorized entity. Repudiation: An entity deceives another by falsely denying responsibility for an act.
Disruption A circumstance or event that interrupts or prevents the correct operation of system services and functions.	Incapacitation: Prevents or interrupts system operation by disabling a system component. Corruption: Undesirably alters system operation by adversely modifying system functions or data. Obstruction: A threat action that interrupts delivery of system services by hindering system operation.
Usurpation A circumstance or event that results in control of system services or functions by an unauthorized entity.	Misappropriation: An entity assumes unauthorized logical or physical control of a system resource. Misuse: Causes a system component to perform a function or service that is detrimental to system security.

Figura 1.3: Conseguenze delle minacce e azioni che le causano

La tabella 1.3 basata su RFC 4949, descrive quattro tipi di conseguenze ed elenca i tipi di attacchi che risultano in ciascuna conseguenza.

La divulgazione non autorizzata (Unauthorized disclosure) è una minaccia alla riservatezza. I seguenti tipi di attacchi possono portare a queste conseguenze

- **Esposizione:** quando un insider rilascia intenzionalmente informazioni sensibili a un estraneo, come ad esempio i numeri di carta di credito. Può anche essere il risultato di un errore umano, hardware o software, che si traduce nell'azione da

parte di un'entità di avere l'accesso non autorizzato di dati sensibili. Ce ne sono stati numerosi casi di questo, come le università che pubblicano accidentalmente le informazioni confidenziali degli studenti sul Web.

- **Intercettazione:** l'intercettazione è un attacco comune nel contesto delle comunicazioni. Su una rete locale condivisa (LAN), come una LAN wireless o a broadcast Ethernet, qualsiasi dispositivo collegato alla LAN può ricevere una copia dei pacchetti destinati a un altro dispositivo. Su Internet, un determinato hacker può accedere al traffico di posta elettronica e ad altri trasferimenti di dati. Tutte queste situazioni possono portare all'accesso non autorizzato ai dati.
- **Inferenza:** un esempio di inferenza è noto come analisi del traffico, in cui un avversario è in grado di ottenere informazioni osservando l'andamento del traffico una rete, come la quantità di traffico tra particolari coppie di host sulla rete. Un altro esempio è l'inferenza di informazioni dettagliate da un database di un utente che ha solo un accesso limitato, questo è realizzato da query ripetute i cui risultati combinati consentono l'inferenza.
- **Intrusione:** un esempio di intrusione è un avversario che ottiene l'accesso non autorizzato a dati sensibili superando le protezioni di controllo degli accessi del sistema.

L'inganno (Deception) è una minaccia per l'integrità del sistema o per l'integrità dei dati. I seguenti tipi di attacchi possono portare a queste conseguenze:

- **Masquerade:** un esempio di masquerade è un tentativo di accesso a un sistema da parte di un utente non autorizzato spacciandosi per uno autorizzato, questo può succedere se l'utente non autorizzato conosce l'ID di accesso e la password di un altro utente. Un altro esempio è la logica dannosa (malicious logic), come un cavallo di Troia, che appare per eseguire una funzione utile o desiderabile, ma in realtà ottiene l'accesso non autorizzato alle risorse di sistema o induce un utente a eseguire altre logiche dannose.
- **Falsificazione:** si riferisce all'alterazione o sostituzione di dati validi o all'introduzione di dati falsi in un file o database. Ad esempio, uno studente può alterare i suoi voti su un database scolastico.

- **Ripudio:** in questo caso, un utente nega l'invio di dati o nega di ricevere o possedere i dati.

L'interruzione (Disruption) è una minaccia alla disponibilità o all'integrità del sistema. I seguenti tipi di attacchi possono portare a queste conseguenze:

- **Incapacità:** questo è un attacco alla disponibilità del sistema. Ciò potrebbe verificarsi come risultato della distruzione fisica o del danneggiamento dell'hardware del sistema. Più tipicamente, un software dannoso, come Trojan, virus o worm, potrebbero operare in modo tale da disabilitare un sistema o alcuni dei suoi servizi.
- **Corruzione:** questo è un attacco all'integrità del sistema. Un Software dannoso in questo contesto potrebbe funzionare in modo tale che le risorse di sistema o i servizi funzionino in modo non intenzionale. Oppure un utente potrebbe ottenere l'accesso non autorizzato a un sistema e modificarne alcune funzioni. Un esempio di quest'ultimo è un posizionamento di una logica backdoor (backdoor logic) nel sistema per fornire il successivo accesso al sistema stesso e alle sue risorse con una procedura diversa da quella abituale.
- **Ostruzione:** un modo per ostacolare il funzionamento del sistema è interferire con le comunicazioni disabilitando i collegamenti di comunicazione o alterando la comunicazione delle informazioni di controllo. Un altro modo è sovraccaricare il sistema mettendo un carico in eccesso sul traffico di una comunicazione o sulle risorse di elaborazione.

L'usurpazione (Usurpation) è una minaccia per l'integrità del sistema. I seguenti tipi di attacchi possono portare a queste conseguenze:

- **Appropriazione indebita:** può includere il furto del servizio. Un esempio è un attacco Denial of Service distribuito, quando il software dannoso è installato su degli host da utilizzare come piattaforme per avviare il traffico verso un host di destinazione. In questo caso, il software maligno fa uso non autorizzato delle risorse del processore e del sistema operativo.
- **Uso improprio:** l'uso improprio può verificarsi per mezzo di una malicious logic o di un hacker che ha ottenuto un accesso non autorizzato a un sistema. In entrambi i casi, le funzioni di sicurezza possono essere disabilitate o contrastate.

1.1.4 Asset e minacce

Le risorse di un sistema informatico possono essere classificate come hardware, software, dati, linee e reti di comunicazione. In questa sottosezione li descriviamo brevemente e mettendoli in relazione con i concetti di integrità, riservatezza e disponibilità.

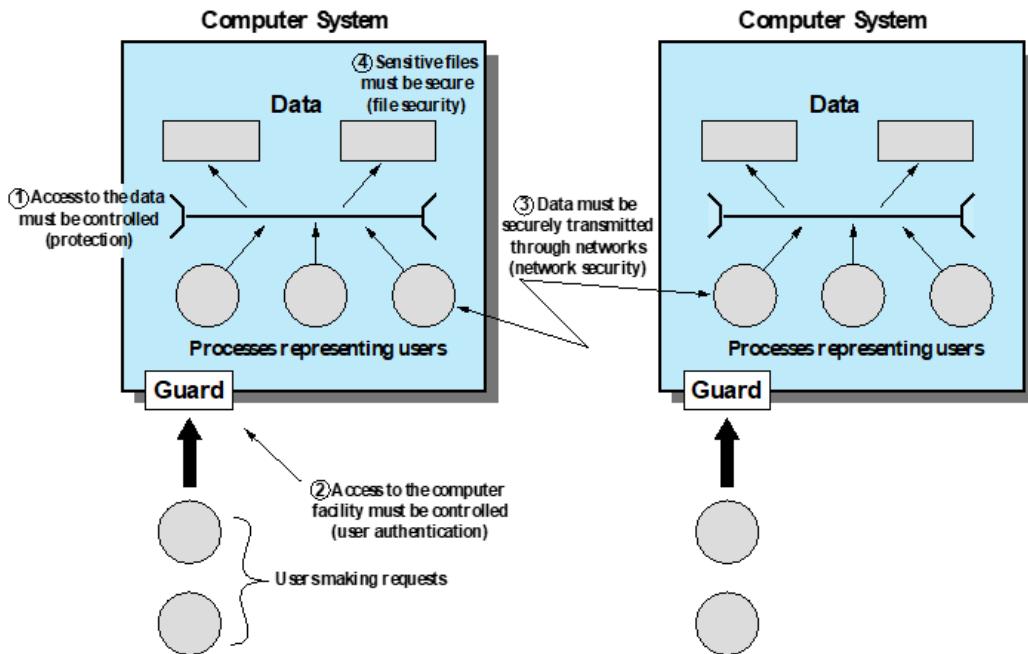


Figura 1.4: Scopo della sicurezza informatica.

Hardware. Una delle principali minacce per l'hardware del computer è la minaccia alla disponibilità. L'hardware è il più vulnerabile agli attacchi e il meno suscettibile ai controlli automatizzati. Le minacce includono danni accidentali e deliberati alle apparecchiature così come il furto. La proliferazione di personal computer e workstation e l'uso diffuso delle LAN aumenta il potenziale di perdite in quest'area. Il furto delle unità USB può portare alla perdita di riservatezza. Le misure di sicurezza fisiche e amministrative sono necessarie per far fronte a queste minacce.

Software. Il software include il sistema operativo, le utilità e l'applicazione programmi. Una delle principali minacce al software è un attacco alla disponibilità. Il Software, in particolare quello applicativo, è spesso facile da eliminare. Il software può anche essere modificato o danneggiato per renderlo inutilizzabile. Un'attenta

gestione della configurazione del software, che include il mantenere dei backup della versione più recente, può mantenere una disponibilità alta. Un problema più difficile da affrontare è la modifica del software che si ha in un programma il quale funziona ancora ma che si comporta in modo diverso rispetto a prima, questa è una minaccia per l'integrità/autenticità. Rientrano in questa categoria i virus informatici e i relativi attacchi. Un ultimo problema è la protezione contro la pirateria del software. Sebbene siano disponibili alcune contromisure, in linea di massima il problema di copie non autorizzate del software non è stata risolta.

Data. Un problema molto più diffuso è la sicurezza dei dati, che coinvolge file e altre forme di dati controllati da individui, gruppi e organizzazioni aziendali.

I problemi di sicurezza relativi ai dati sono ampi e comprendono disponibilità, segretezza e integrità. In caso di disponibilità, la preoccupazione è con la distruzione di file di dati, che può verificarsi accidentalmente o intenzionalmente. Una preoccupazione evidente per la segretezza è la lettura non autorizzata di file di dati o database, e quest'area è stata forse oggetto di ulteriori ricerche e sforzi rispetto a qualsiasi altro settore della sicurezza informatica. Una minaccia meno ovvia alla segretezza comporta l'analisi dei dati e si manifesta nell'utilizzo delle cosiddette banche dati statistiche, che forniscono informazioni di sintesi o aggregate. Presumibilmente, l'esistenza delle informazioni aggregate non minacciano la privacy delle persone coinvolte. Tuttavia, con la crescita dell'uso delle banche dati statistiche, c'è un rischio crescente per la divulgazione di informazioni personali. In sostanza, le caratteristiche di un individuo possono essere identificate attraverso un'analisi attenta. Ad esempio, se una tabella registra l'aggregato dei redditi degli intervistati A, B, C e D e un altro registra l'aggregato dei redditi di A, B, C, D ed E, la differenza tra i due aggregati sarebbe il reddito di E. Questo problema è esasperato dal desiderio crescente di combinare set di dati. In molti casi, abbinando diversi set di dati per coerenza tra i diversi livelli di aggregazione è necessario l'accesso alle singole unità. Pertanto, le singole unità, che sono oggetto di problemi di privacy, sono disponibili in varie fasi del trattamento dei set di dati. Infine, l'integrità dei dati è una delle principali preoccupazioni nella maggior parte delle installazioni. Le modifiche ai file di dati possono avere conseguenze che vanno da minori a disastrose.

1.1.5 Attacchi passivi e attivi

Gli attacchi alla sicurezza della rete possono essere classificati come attacchi passivi e attacchi attivi. Un attacco passivo tenta di imparare o fare uso delle informazioni del sistema, ma non influisce sulle risorse di quest'ultimo. Un attacco attivo tenta di alterare le risorse di sistema o di influenzare il loro funzionamento.

Attacchi passivi

Gli **attacchi passivi** generalmente riguardano l'intercettazione o il monitoraggio di trasmissioni di dati. L'obiettivo dell'attaccante è ottenere le informazioni che vengono trasmesse. Due tipi di attacchi passivi sono il rilascio del contenuto dei messaggi e dell'analisi del traffico.

Rilascio dei contenuti di un messaggio Il **rilascio dei contenuti** del messaggio è facilmente comprensibile. Una conversazione telefonica, un messaggio di posta elettronica e un file trasferito possono contenere dati sensibili o informazioni confidenziali. Vorremmo impedire a un avversario di imparare il contenuto di queste trasmissioni.

Analisi del traffico. Un secondo tipo di attacco passivo, **l'analisi del traffico**, è più sottile. Supponiamo che noi abbiamo un modo per mascherare il contenuto dei messaggi o altre informazioni del traffico di dati, in modo che gli oppositori, anche se hanno catturato il messaggio, non possono estrarre le informazioni dal messaggio. La tecnica comune per mascherare i contenuti è la crittografia. Anche se disponiamo di una protezione crittografica, un avversario potrebbe comunque essere in grado di osservare lo schema di questi messaggi. L'avversario potrebbe determinare la posizione e l'identità degli host nella comunicazione e potrebbe osservare la frequenza e la lunghezza dei messaggi scambiati. Queste informazioni potrebbero essere utili per indovinare la natura della comunicazione che stava avvenendo.

Gli attacchi passivi sono molto difficili da rilevare perché non coinvolgono alterazione dei dati. In genere, il traffico dei messaggi viene inviato e ricevuto in un modo apparentemente normale e né il mittente né il destinatario sono consapevoli che una terza parte ha letto i messaggi o osservato l'andamento del traffico. Tuttavia, è possibile prevenire il successo di questi attacchi, di solito mediante crittografia. Per-

tanto, l'enfasi nell'affrontare gli attacchi passivi è sulla prevenzione piuttosto che il rilevamento.

Attacchi attivi

Gli attacchi attivi comportano alcune modifiche del flusso di dati o la creazione di un falso flusso, esso può essere suddiviso in quattro categorie: replay, masquerade, modifica dei messaggi e denial of service.

Replay. Il replay comporta l'acquisizione passiva di un'unità di dati e la sua successiva ritrasmissione per produrre un effetto non autorizzato.

Masquerade. Una masquerade ha luogo quando un'entità finge di essere un'entità diversa. Un attacco di questo tipo di solito include una delle altre forme di attacco attivo. Per esempio, le sequenze di autenticazione possono essere catturate e riprodotte dopo che è avvenuta una sequenza di autenticazione valida, abilitando così un'entità autorizzata con pochi privilegi a ottenere privilegi extra impersonando un'entità che dispone di tali privilegi.

Modifica di un messaggio. La modifica dei messaggi significa semplicemente che una parte di un legittimo messaggio è alterato, o che i messaggi sono ritardati o riordinati, per produrre un effetto non autorizzato. Ad esempio, un messaggio che afferma: "Consenti a John Smith di leggere dati di file riservati" viene modificato per dire "Consenti a Fred Brown di leggere dati di file riservati".

DOS. La negazione del servizio impedisce o inibisce il normale utilizzo o gestione delle strutture di comunicazione. Questo attacco può avere un obiettivo specifico, per esempio un'entità può sopprimere tutti i messaggi diretti a una particolare destinazione (ad esempio, la sicurezza del servizio di audit). Un'altra forma di rifiuto del servizio è l'interruzione di un'intera rete, o disabilitando la rete o sovraccaricandola di messaggi in modo da degradarne le prestazioni.

Gli attacchi attivi presentano le caratteristiche opposte degli attacchi passivi. Invece gli attacchi passivi sono difficili da rilevare, sono disponibili misure per prevenirli con successo. D'altra parte, è abbastanza difficile prevenire assolutamente gli attacchi attivi, perché per farlo richiederebbe la protezione fisica di tutte le strutture

e i percorsi di comunicazione in ogni momento. Invece, l'obiettivo è rilevarli e riprendersi da qualsiasi disservizio o ritardi da essi causati. Poiché il rilevamento ha un effetto deterrente, esso può anche contribuire alla prevenzione.

1.1.6 Requisiti di sicurezza

Esistono diversi modi per classificare e caratterizzare le contromisure che possono essere utilizzate per ridurre le vulnerabilità e affrontare le minacce alle risorse di sistema. In questa sottosezione, vediamo contromisure in termini di requisiti funzionali, e seguiamo la classificazione definita in FIPS 200. Questo standard enumera 17 aree relative alla sicurezza con riguardo alla protezione della riservatezza, dell'integrità e della disponibilità delle informazioni di sistemi e le informazioni elaborate, archiviate e trasmesse da tali sistemi.

1. **Accesso controllato:** limitare l'accesso al sistema informativo agli utenti autorizzati, ai processi che agiscono per conto degli utenti autorizzati, o ai dispositivi (inclusi altri sistemi informativi) e alle tipologie di transazioni e funzioni che gli utenti autorizzati possono esercitare.
2. **Consapevolezza e Formazione:** garantire che i gestori e gli utenti dei sistemi informativi organizzativi siano consapevole dei rischi per la sicurezza associati alle proprie attività e delle leggi, dei regolamenti e delle politiche applicabili relativi alla sicurezza dei sistemi informativi organizzativi e garantire che il personale sia adeguatamente addestrato a svolgere i compiti e le responsabilità assegnate in materia di sicurezza delle informazioni.
3. **Audit e responsabilità:** creare, proteggere e conservare i record di audit del sistema informativo per consentire il monitoraggio, l'analisi, l'indagine e la segnalazione di atti illeciti, non autorizzati o di attività non appropriate del sistema informativo. Garantire inoltre che le azioni dei singoli individui nel sistema possano essere ricondotte in modo univoco a tali utenti in modo che possano essere ritenuti responsabili di esse.
4. **Certificazione, accreditamento e valutazioni di sicurezza:** valutare periodicamente i controlli di sicurezza nei sistemi informativi organizzativi per determinare se i controlli sono efficaci nella loro applicazione. Sviluppare e attuare

piani d’azione volti a correggere le carenze e ridurre o eliminare le vulnerabilità in questi sistemi. Autorizzare l’esercizio dei sistemi informativi organizzativi ed eventuali connessioni associate a questi. Monitorare continuamente i controlli di sicurezza del sistema informativo per garantire la continua efficacia di essi.

5. **Gestione della configurazione:** stabilire e mantenere le configurazioni di base e gli inventari dei sistemi (inclusi hardware, software, firmware e documentazione) durante i rispettivi cicli di vita di sviluppo del sistema. Stabilire e far rispettare le impostazioni di configurazione di sicurezza per i prodotti informatici utilizzati nei sistemi.
6. **Pianificazione di emergenza:** stabilire, mantenere e implementare piani di risposta alle emergenze, operazioni di backup, e il ripristino post-disastro per i sistemi in modo da garantire la disponibilità di risorse informative critiche e continuità operativa in situazioni di emergenza.
7. **Identificazione e autenticazione:** identificare gli utenti del sistema, i processi che agiscono per conto degli utenti o dei dispositivi e autenticare (o verificare) le identità di tali utenti, processi o dispositivi, come prerequisito per consentire l’accesso ai sistemi.
8. **Risposta all’incidente:** stabilire una capacità operativa di gestione degli incidenti per le informazioni organizzative dei sistemi che includono un’adeguata preparazione, rilevamento, analisi, contenimento, recupero e un controllo delle attività di risposta dell’utente. Tracciare, documentare e segnalare gli incidenti ai funzionari appropriati e/o alle autorità.
9. **Manutenzione:** eseguire la manutenzione periodica e tempestiva dei sistemi, fornire controlli efficaci sugli strumenti, sulle tecniche, sui meccanismi e sul personale utilizzato per condurre una manutenzione del sistema informativo.
10. **Protezione dei media:** proteggere i media dei sistemi, sia cartacei che digitali, limitare l’accesso alle informazioni dei media agli utenti autorizzati e sanificare o distruggere prima i supporti del sistema informativo prima dello smaltimento o del rilascio per il riutilizzo.
11. **Protezione fisica e ambientale:** limitare l’accesso fisico dei soggetti autorizzati ai sistemi informativi, alle apparecchiature e ai rispettivi ambienti opera-

tivi. Proteggere l'impianto fisico e l' infrastruttura di supporto per i sistemi. Fornire utilità di supporto per i sistemi informativi e proteggere quest'ultimi dai rischi ambientali fornendo adeguati controlli ambientali alle strutture che li contengono.

12. **Pianificazione:** sviluppare, documentare, aggiornare periodicamente e implementare piani di sicurezza per le informazioni organizzative dei sistemi che descrivono i controlli di sicurezza esistenti o previsti e le regole di comportamento dei soggetti che accedono ai sistemi.
13. **Sicurezza del personale:** garantire che le persone che occupano posizioni di responsabilità all'interno delle organizzazioni (compresi i fornitori di servizi di terze parti) siano affidabili e soddisfino i criteri di sicurezza stabiliti per quelle posizioni. Garantire che le informazioni organizzative e i sistemi informativi siano protetti durante e dopo le azioni del personale quali licenziamenti e trasferimenti. Applicare sanzioni formali per il personale che non fa rispettare le politiche e le procedure di sicurezza dell'organizzazione.
14. **Valutazione del rischio:** valutare periodicamente il rischio per le operazioni organizzative (inclusi missioni, funzioni, immagine o reputazione), risorse organizzative e individui, risultanti dal funzionamento del sistema e il relativo trattamento, archiviazione o trasmissione di informazioni organizzative.
15. **Acquisizione di sistemi e servizi:** Allocare risorse sufficienti per proteggere adeguatamente l'organizzazione dei sistemi. Impiegare processi del ciclo di vita dello sviluppo del sistema che incorporano considerazioni sulla sicurezza. Imporre limitazioni all'utilizzo e all'installazione del software e garantire che i fornitori di terze parti adottino adeguate misure di sicurezza per proteggere le informazioni, le applicazioni e/o i servizi "esternalizzati" dall'organizzazione.
16. **Protezione del sistema e delle comunicazioni:** monitorare, controllare e proteggere le comunicazioni organizzative (vale a dire, le informazioni trasmesse o ricevute dai sistemi) ai confini esterni e interni. Impiegare progetti "architettonici", sviluppo software tecniche e principi di ingegneria dei sistemi che promuovono un'efficace sicurezza delle informazioni all'interno di dei sistemi organizzativi.

17. Integrità del sistema e delle informazioni: identificare, segnalare e correggere le informazioni e le falte del sistema in modo tempestivo. Fornire protezione da codice dannoso in posizioni appropriate all'interno del sistema organizzativo e monitorare gli avvisi di sicurezza del sistema informativo e adottare le azioni appropriate in risposta.

Bibliografia

- [1] Robert Seacord. *Secure Coding in C and C++*. SEI Series in Software Engineering Paperback. Addison-Wesley Professional, 2 edition, 2013. ISBN 9780321822130.
- [2] Wikipedia. Linguaggio assembly — wikipedia, l'enciclopedia libera, 2022. URL http://it.wikipedia.org/w/index.php?title=Linguaggio_assembly&oldid=125849812.
- [3] Wikipedia. Lista concatenata — wikipedia, l'enciclopedia libera, 2022. URL http://it.wikipedia.org/w/index.php?title=Lista_concatenata&oldid=126467077.
- [4] Wikipedia contributors. Code signing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Code_signing&oldid=1076823321, 2022.
- [5] Wikipedia contributors. Executable space protection — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Executable_space_protection&oldid=1079234448, 2022.
- [6] Lawrie Brown William Stallings. *Computer Security: Principles and Practice*. Pearson Education Limited, 4 edition, 2018.