Meccanica per la corruzione della memoria

CSP009_S_E

PRESENTAZIONE DA PARTE DI:
ELIAS ATHANASOPOULOS

Elias Athanasopoulos

Profilo del formatore

- Professore associato, Università di Cipro
- Ricerca sulla sicurezza dei sistemi e sulla privacy
- https://elathan.github.io/srec/

Argomento-1: Introduzione alle vulnerabilità del software

Tratteremo queste competenze

- Come viene sviluppato il software ?
- Qual è la differenza tra i sistemi di programmazione memory safe e memory unsafe?
- Come presentano le vulnerabilità a prova di memoria?

Argomento-2: Sfruttamento Vulnerabilità nel software nativo

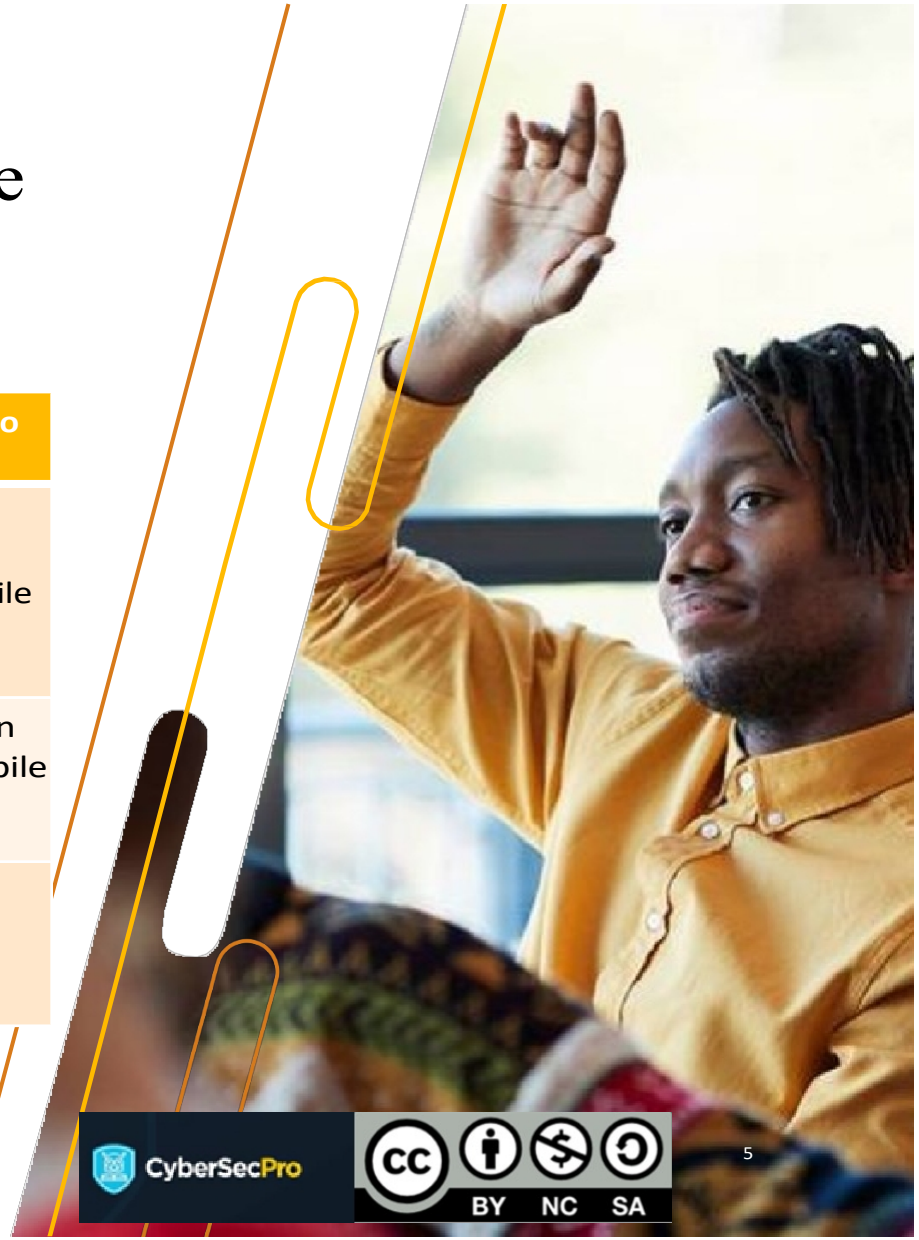
Tratteremo queste competenze

- La meccanica di un attacco a flusso di controllo
- Come si può introdurre nuovo codice in un programma vulnerabile?
- Come funziona moderno software sfruttamento

Esercitazioni pratiche di formazione

Esercizi pratici che richiedono un impegno personale e di squadra.

	Titolo	Obiettivo dell'esercizio fisico
Esercizio-1 (Giorno-1)	Attacco al flusso di controllo	Dirottare il flusso di controllo di un programma vulnerabile
Esercizio-2 (Giorno-1)	Iniezione di codice	Iniettare codice in un programma vulnerabile
Esercizio-3 (Giorno-1)	Attacco ROP	Compromettere un programma vulnerabile



Metodo di valutazione

Delineare gli elementi di valutazione e il processo di valutazione

Elemento di valutazione	Come	Note
Compiti applicati (individuali)	Soluzione del problema da presentare in seguito	
Lavoro di squadra	Progetto di squadra da presentare in seguito	
Discussione di gruppo	Durante il workshop	

Conoscenze di base e prerequisiti

Conoscenze di base:

Conoscenza di base della programmazione C

Conoscenza di base dei sistemi operativi

Prerequisiti:

Nessuno

Strumenti tecnici e altri requisiti

Strumenti tecnici

Sistema operativo basato su Linux con una catena di strumenti per il compilatore C

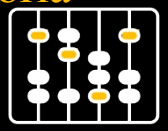
Registrazione: Come iscriversi e altre informazioni pratiche

La procedura di registrazione specifica per la formazione Cybersecurity Essentials e Management può variare a seconda dell'ente di formazione o dell'istituzione. Tuttavia, le fasi generali sono generalmente semplici e possono essere completate online o di persona.

1. Registrazione online
2. Registrazione di persona
3. Ulteriori informazioni pratiche

Attacchi di corruzione della memoria

Sicurezza della memoria

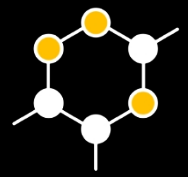


Sistemi di programmazione memory-safe vs memory- unsafe

Attacchi al flusso di controllo

Come può essere violata la sicurezza della memoria

Attacchi moderni



Come funzionano gli attacchi moderni?

Progressi del corso

01. Sistemi di programmazione a prova di memoria

02. Vulnerabilità

03. Attacchi al flusso di controllo

04. Iniezione di codice

05. Programmazione orientata al ritorno (ROP)

Software

I programmi sono scritti in un linguaggio di alto livello

- C, C++, Java, C#, Ruby, Python

Vengono compilati per l'esecuzione

- Codice macchina (codice non gestito e non sicuro)
- Codice di macchina virtuale, come la JVM (codice gestito e sicuro)

Le diverse architetture presentano proprietà diverse nell'esecuzione del software.

- Alcuni concetti generici si applicano a tutti

Sistemi sicuri e non sicuri

Sistemi di programmazione sicuri

- Esecuzione di codice gestito in una macchina virtuale (ad esempio, Java)
- Eseguire l'analisi statica e in fase di compilazione (rifiutare il codice non sicuro) e utilizzare controlli in fase di esecuzione (ad esempio, Rust).
- Accesso limitato alla memoria

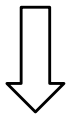
Sistemi di programmazione non sicuri

- Accesso illimitato alla memoria (ad esempio, C/C++)
- Migliori prestazioni
- Accesso a basso livello (ad esempio, driver)
- Codice ereditario

Sistemi di programmazione non sicuri contro sistemi di programmazione non sicuri

Non sicuro (ad esempio, C/C++)

```
a[i]= j;
```



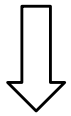
Codice macchina

`a` è solo un indirizzo di partenza

Non c'è modo di verificare se `i` si trova nei limiti di `un`

Sicuro (ad esempio, Java)

```
a[i]= j;
```



Codice della macchina virtuale

`a` è un'entità ben definita

La VM conosce il tipo di `a`, i limiti, il numero di riferimenti, ecc.

Progressi del corso

01. Sistemi di programmazione a prova di memoria

02. Vulnerabilità

03. Attacchi al flusso di controllo

04. Iniezione di codice

05. Programmazione orientata al ritorno (ROP)

Esempio di programma C

```
#include <stdio.h>

int main(int argc, char *argv[]) {

        int ary[5]= {1, 2, 3, 4, 5};

        fprintf(stderr, "Il quinto numero di ary è: %d\n", ary[5]);

        ritorno 1;
}
```

Accesso fuori dai limiti!

```
#include <stdio.h>

int main(int argc, char *argv[]) {

        int ary[5]= {1, 2, 3, 4, 5};

        fprintf(stderr, "Il quinto numero di ary è: %d\n", ary[5]);

        ritorno 1;
}
```

Vocabolario

Vulnerabilità

- Un errore del software (noto anche come bug) che può potenzialmente consentire a qualcuno di trarre vantaggio dal programma vulnerabile.

Sfruttare

- Il processo di controllo di un programma una o più vulnerabilità.
- Non tutte le vulnerabilità possono essere sfruttate

Vocabolario

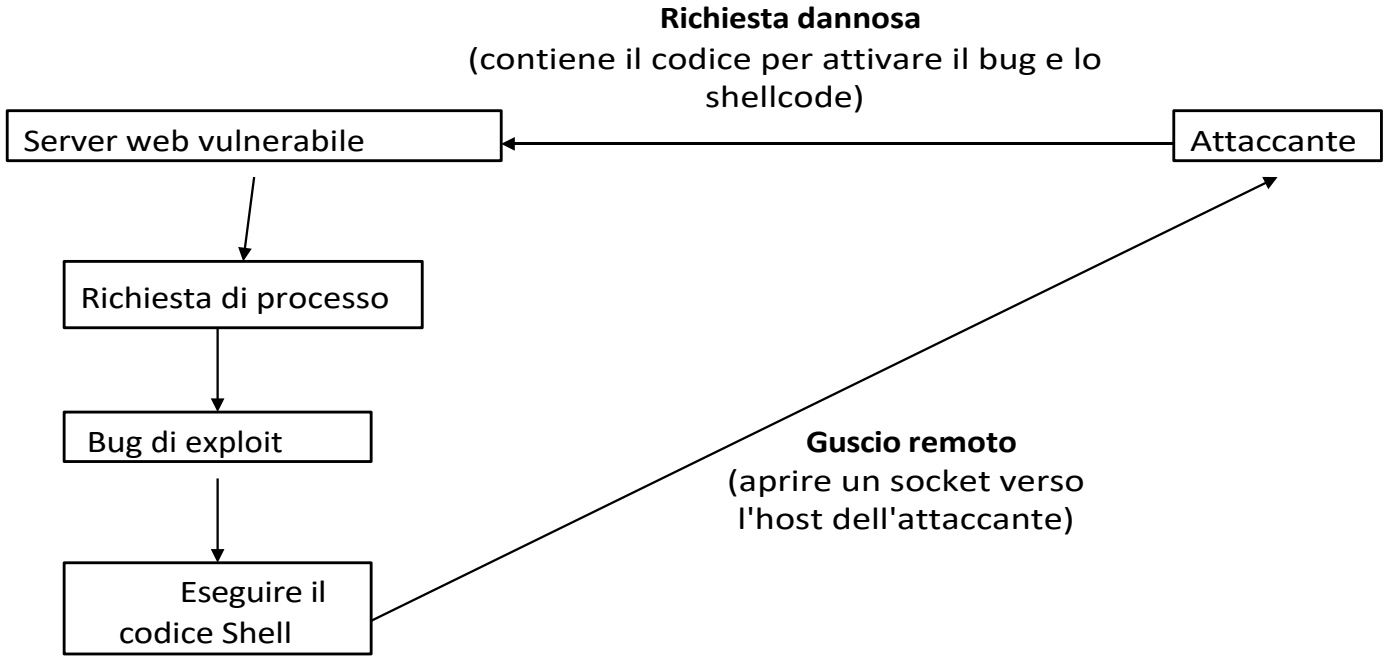
Esecuzione di codice arbitrario

- Lo stato di un exploit in cui un attaccante può eseguire un programma a sua scelta.

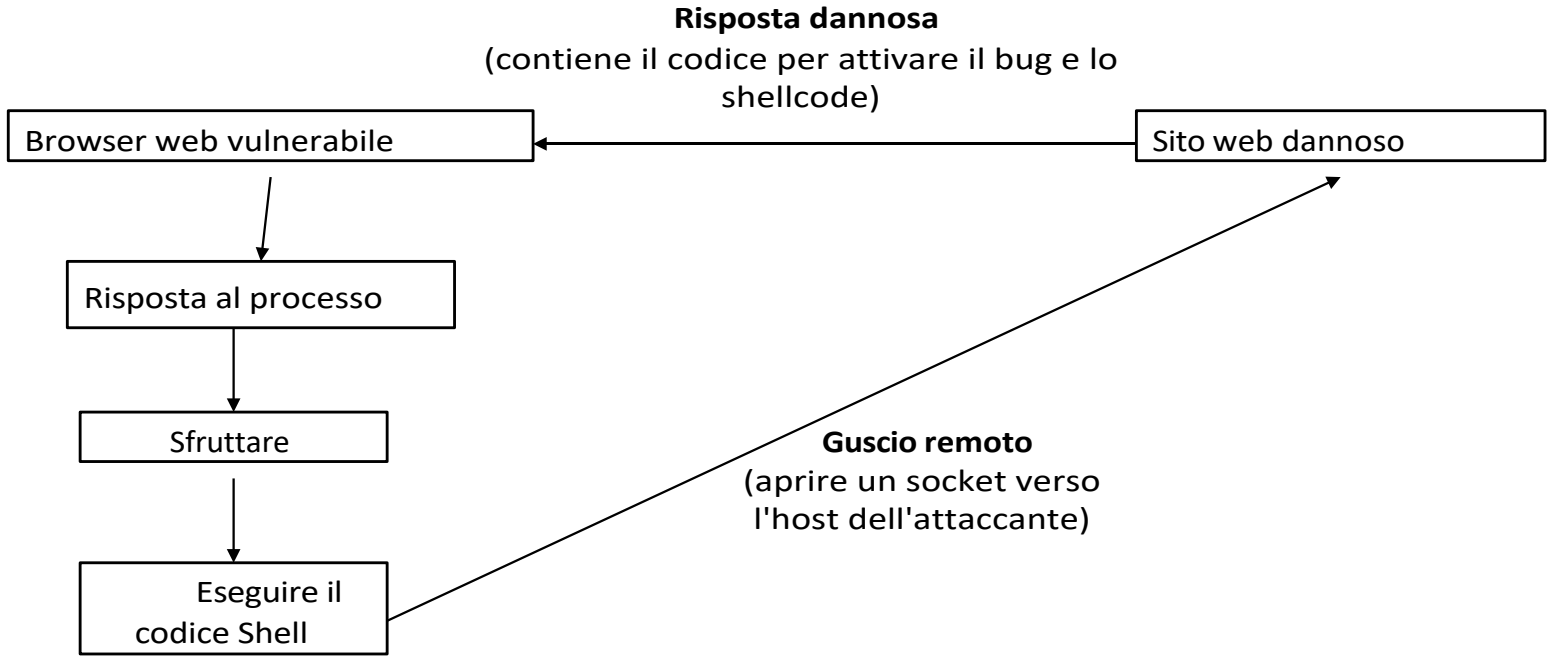
Codice a guscio

- Un codice macchina che un programma vulnerabile esegue e che serve agli scopi dell'attaccante.
- Generare una shell (può essere remota), scaricare malware, creare un account nascosto, manipolare il software, ecc.
- Fortemente dipendente dall'architettura

Idea di alto livello



Idea di alto livello



Progressi del corso

01. Sistemi di programmazione a prova di memoria

02. Vulnerabilità

03. Attacchi al flusso di controllo

04. Iniezione di codice

05. Programmazione orientata al ritorno (ROP)

Funzioni

Il software è composto da diverse funzioni

- `main(), printf(), malloc(), create_user(),` ecc.

Le funzioni consentono il riutilizzo del codice

- Ogni volta che si vuole visualizzare un messaggio è sufficiente chiamare `printf`

In un programma una funzione può chiamare una funzione e poi un'altra funzione.

- Tutto questo concatenamento di funzioni è chiamato **flusso di controllo**

La vita di una funzione

Ogni volta che viene richiamata una funzione, il flusso di controllo del programma viene modificato.

- Dobbiamo farlo in modo trasparente
- Una volta terminata la funzione, il flusso di controllo dovrebbe essere ripreso

Le funzioni possono accettare argomenti

Le funzioni possono restituire dati Le

funzioni possono creare dati locali

Vocabolario

Quando viene chiamata la funzione `foo`

- `pippo` è il chiamante
- L'indirizzo che ha chiamato la funzione è chiamato **sito di chiamata** (o chiamante).

La pila

Le funzioni hanno bisogno di memoria per il loro lavoro

- Questo è lo stack

Questa memoria è per i dati di breve durata

- Una volta terminata la funzione, ci si può liberare dei dati coinvolti

Dipende dall'architettura

- L'idea principale non cambia

La pila può contenere diversi elementi

- Argomenti della funzione, indirizzo di ritorno, puntatore al vecchio frame, argomenti locali

Pila di Intel (32 bit)

Lo stack cresce da indirizzi di memoria più alti a indirizzi di memoria più bassi

- È come se la pila fosse capovolta.

La parte superiore dello stack è sempre conservata in un registro hardware (`%esp`)

Ogni funzione crea un nuovo **stack frame** al momento dell'esecuzione.

- Una porzione virtuale all'interno dello stack
- Lo stack frame viene distrutto una volta terminate le funzioni.

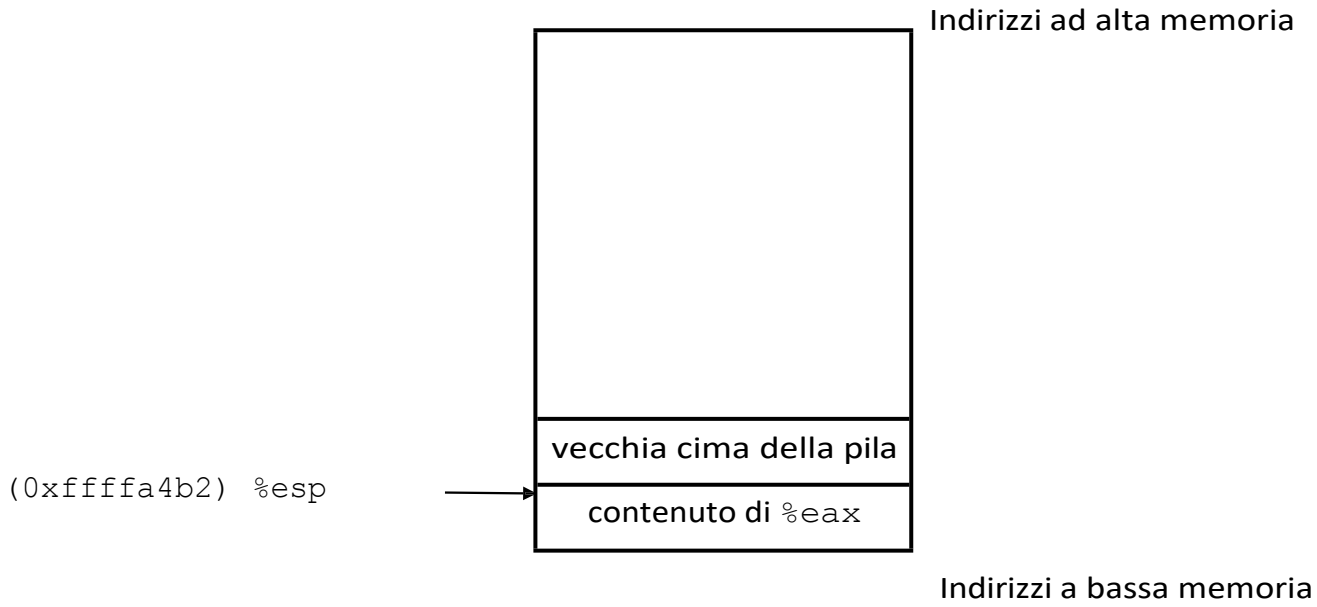
La parte superiore dello stack frame viene conservata in un registro hardware (`%ebp`).

Inserimento in pila

```
spingere %eax
   sub 0x4, %esp mov
   %eax, (%esp)
```

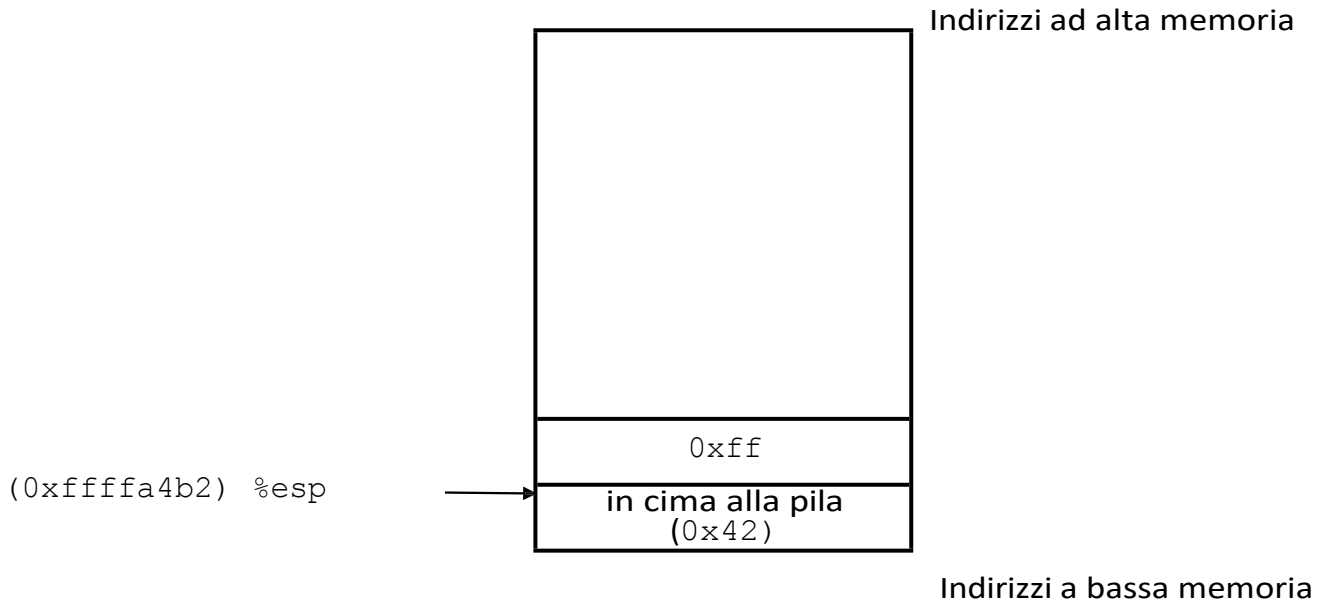
Inserimento in pila

```
spingere %eax
   sub 0x4, %esp mov
   %eax, (%esp)
```



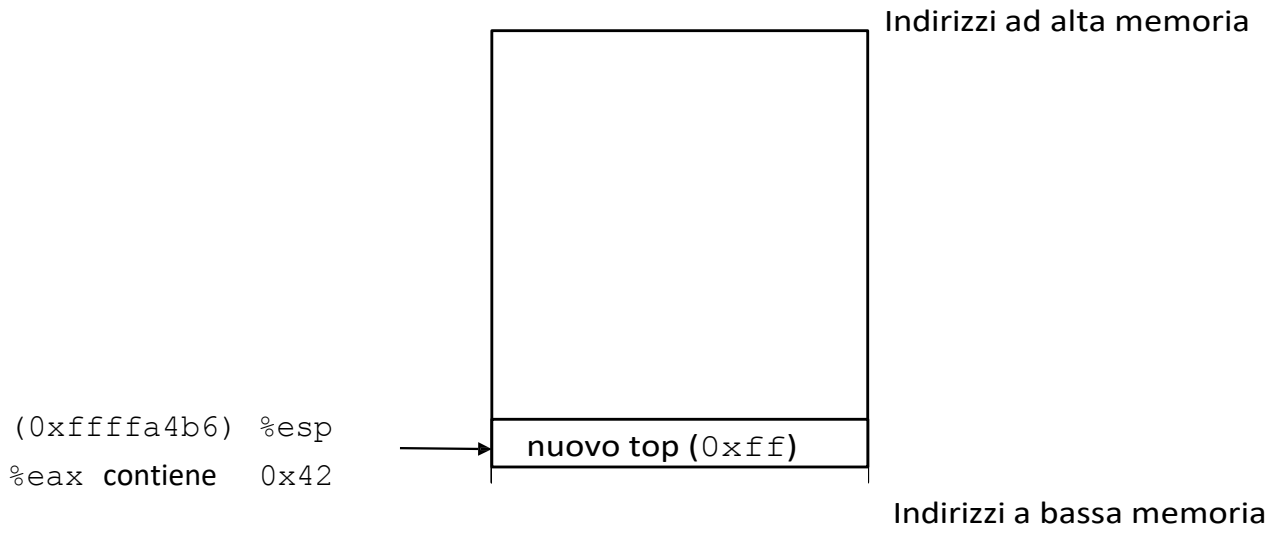
Eliminazione della pila

```
pop %eax
   mov %(esp), %eax add
   0x4, %esp
```

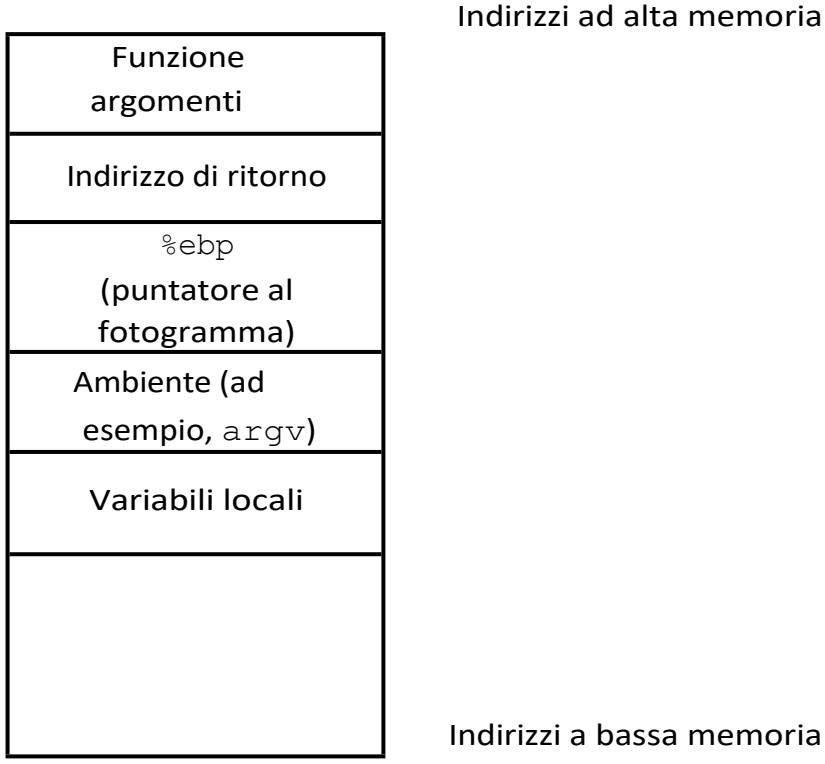


Eliminazione della pila

```
pop %eax
   mov %(esp), %eax add
   0x4, %esp
```



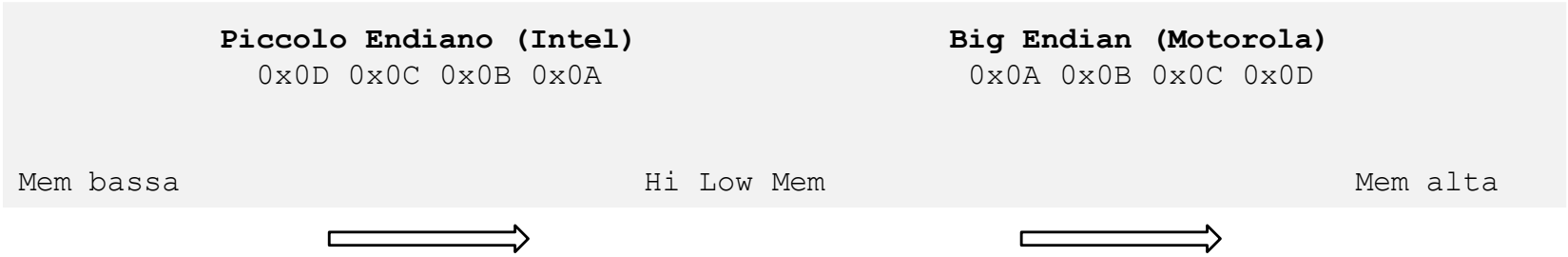
Struttura dello stack in Intel a 32 bit



Endianità

Si supponga che la parola a 32 bit: `0x0A0B0C0D`

Due modi possibili per memorizzarlo



Progressi del corso

01. Sistemi di programmazione a prova di memoria

02. Vulnerabilità

03. Attacchi al flusso di controllo

04. Iniezione di codice

05. Programmazione orientata al ritorno (ROP)

Codice a guscio

Un programma che viene *inserito* in un input dell'utente

- crea una shell (remota), scarica malware, crea un utente, si eleva (diventa root), installa una backdoor, ecc.

Il programma è di solito piccolo e preciso

- Soprattutto nei buffer overflow, i buffer vulnerabili possono avere dimensioni limitate.
- Non sono consentite di stringhe, altrimenti le copie di stringhe possono distruggere lo shellcode.

Fortemente dipendente dall'architettura

Programmazione altamente non ortodossa

- Assemblaggio personalizzato

Generazione di un guscio

```
int execve(const char *filename,
                char *const argv[],
                char *const envp[]);
```

Esempio:

- `execve("/bin/sh", NULL, NULL);`
- Ricordate che qui non si fa una vera e propria programmazione!

Chiamate di sistema in Linux

Può essere invocato tramite un interrupt software

- istruzione di montaggio: `int`
- Ad esempio, per `execve` il numero di interrupt è `0x0b` (o 11 in decimale)
- I parametri vengono passati in registri

Il sistema operativo dispone di una tabella di chiamate di sistema

- Ogni numero di chiamata di sistema richiama il codice appropriato
- `/usr/include/asm/unistd_32.h`:

```
#define NR_execve 11
```

`execve` in Linux/IA32

- `execve("/bin/sh", NULL, NULL);`

`%eax`: valore di ritorno

`%ebx`

- primo argomento, l'indirizzo della memoria in cui è memorizzato "/bin/sh"

`%edx` e **`%ecx`**

- Argomento 2nd e 3rd
- Possiamo semplicemente azzerare questi

Hacks

"/bin/sh" è di 7 byte, sarebbe bello se fosse di 8 byte Facile

correzione sporca

- /bin//sh

Nessun 0

- `strcpy` può dividere lo shellcode se contiene degli 0
- Se abbiamo bisogno di azzerare un registro, utilizziamo **`xor`**

Spingere /bin//sh

carbone	h	s	/	/	n	i	b	/
ASCII (esagono)	0x68	0x73	0x2f	0x2f	0x6e	0x69	0x62	0x2f
valore e	**0x68732f2f**				**0x6e69622f**			

Shellcode per `execve("/bin/sh");`

```
.sezione .dati

.sezione .testo

.globl _start


Inizio:

                xor             %eax,%eax

                spingere        %eax                                    # \0

                spingere        $0x68732f2f             # hs//

                spingere        $0x6e69622f             # nib/

                mov             %esp,%ebx

                xor                                             %ecx,%ecx

                xor                                     %edx,%edx

                mov             $0xb,%al

                int             $0x80
```

Progressi del corso

01. Sistemi di programmazione a prova di memoria

02. Vulnerabilità

03. Attacchi al flusso di controllo

04. Iniezione di codice

05. Programmazione orientata al ritorno (ROP)

Memoria non eseguibile

Diversi nomi

- NX-bit (bit non eseguibile)
- DEP (Prevenzione dell'esecuzione dei dati)
- W^X (scrittura XOR esecuzione)

Applicato in hardware (MMU)

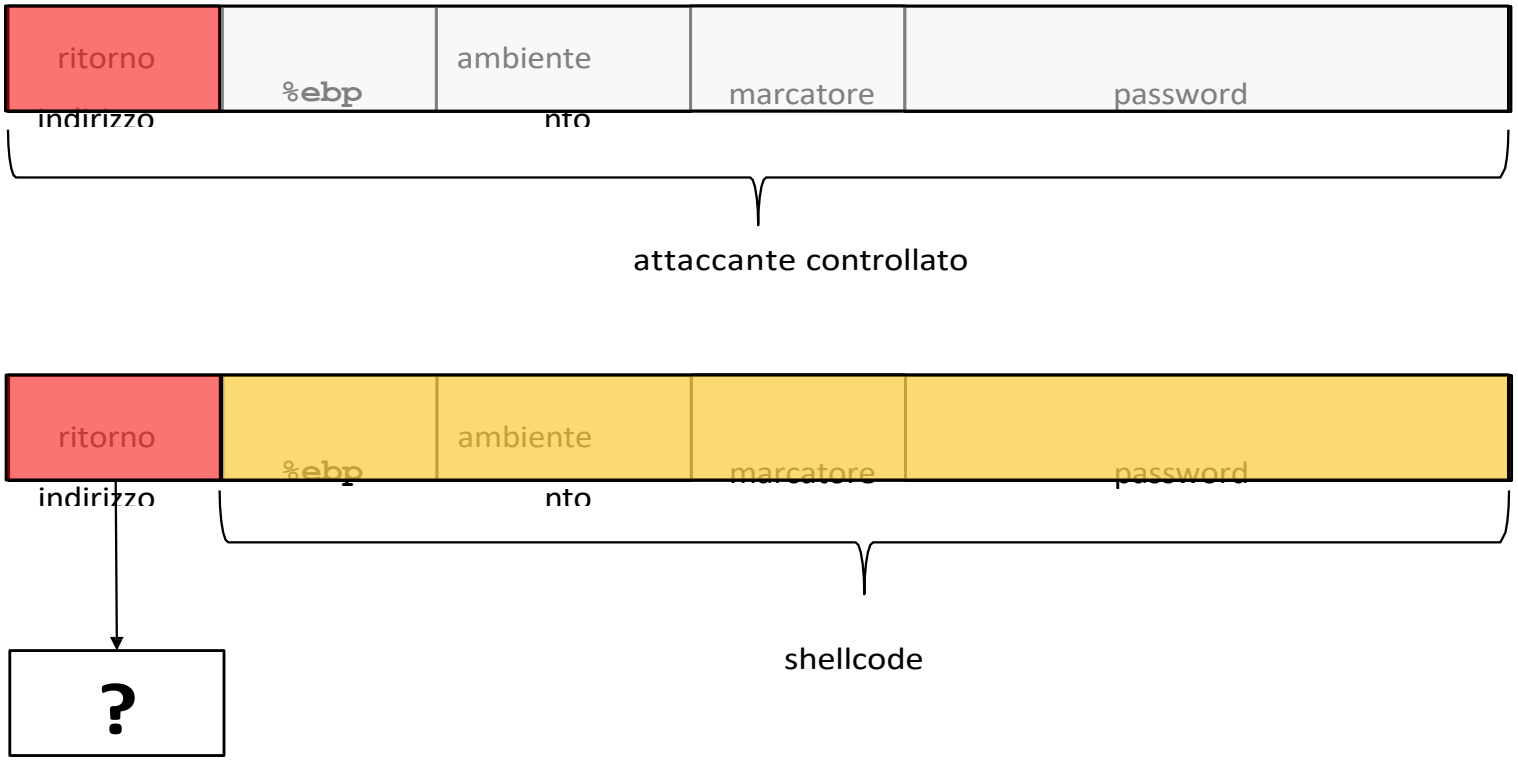
Le pagine di memoria non possono essere eseguibili e scrivibili

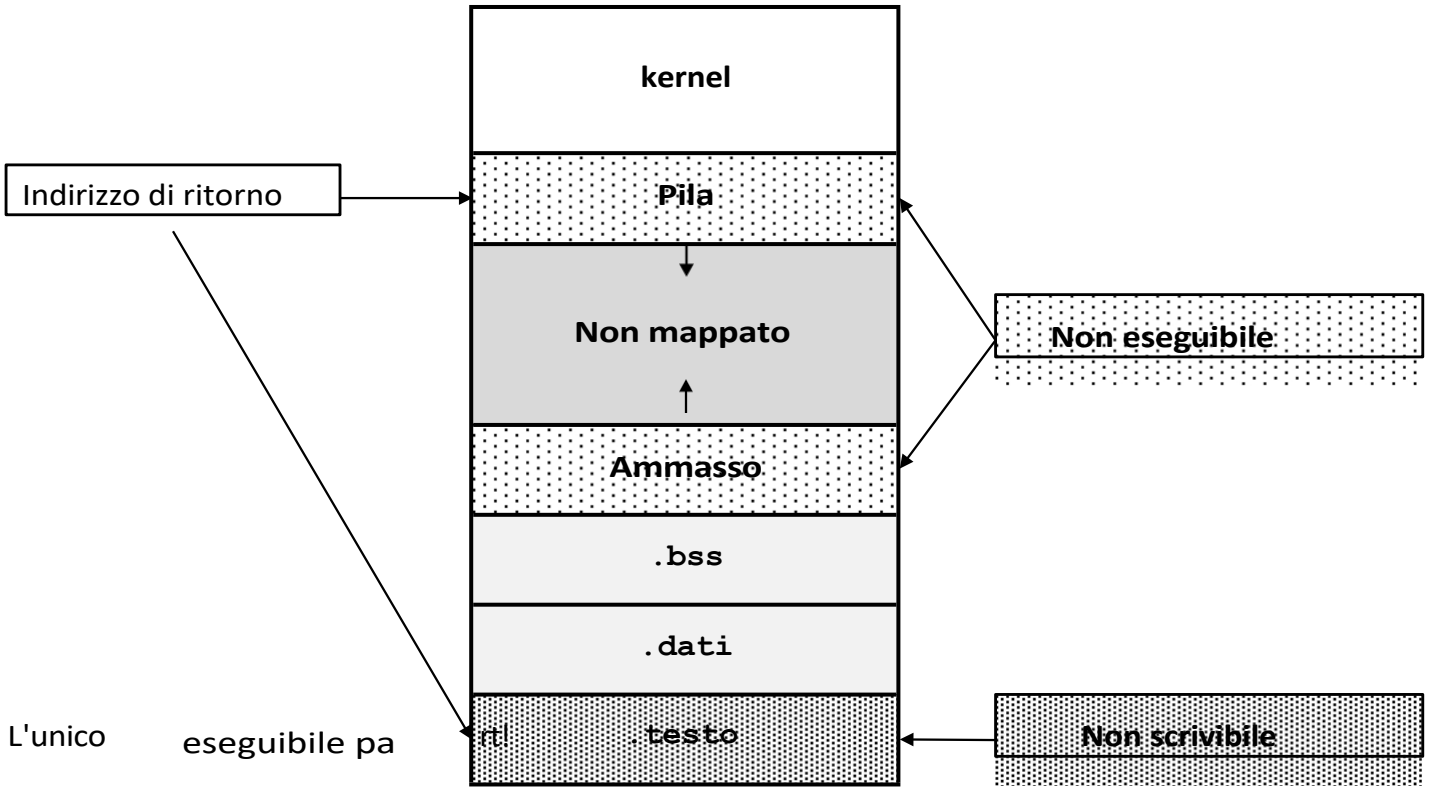
- Stack e heap sono scrivibili ma non eseguibili
- Il codice è eseguibile ma non scrivibile

I permessi possono essere modificati utilizzando le chiamate di sistema

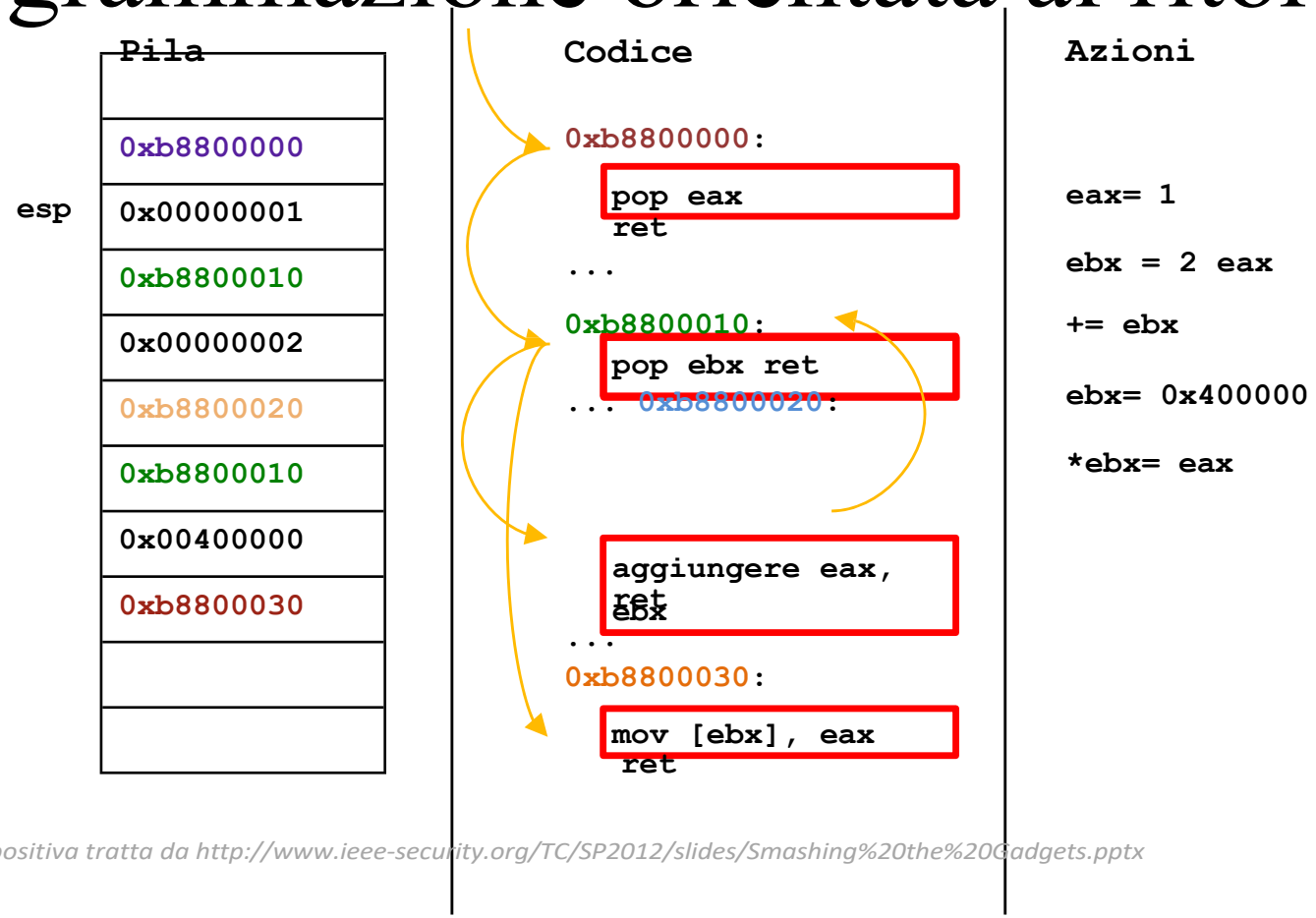
- `mprotect()` per Linux
- `VirtualProtect()` per Windows

Dove dobbiamo saltare?




kernel
Indirizzo di ritorno
Pila
Non mappato
Non eseguibile
Ammasso
.bss
.dati
L'unico
eseguibile pa
rt
.testo
Non scrivibile

Programmazione orientata al ritorno



Diapositiva tratta da http://www.ieee-security.org/TC/SP2012/slides/Smashing%20the%20Gadgets.pptx

Riutilizzo del codice

I programmi includono ampie parti di codice esistente

- Disponibile durante lo sfruttamento

Piccole sequenze di istruzioni che terminano con un `ret`

- Hanno chiamato i gadget
- Eseguono del codice e *ritornano* (cioè leggono lo stack per il gadget successivo).
- Lo stack è controllato dall'attaccante (usa `esp` come contatore del programma)

Concatenazione di più gadget

- Programmazione orientata al ritorno (ROP)
- Turing completo
- In pratica si usa per rendere eseguibile una regione

Gadget

Abbiamo visto gadget come

- `aggiungere eax, ebx; ret`
- `mov [ebx], eax; ret`
- `pop eax; ret`
- `...`

Questo codice è realistico?

Intel e l'architettura CISC

Set di istruzioni denso

- Tutti i valori corrispondono a un'istruzione valida

Istruzioni non allineate Istruzioni

a lunghezza variabile

- Il salto nel mezzo di un'istruzione genera una nuova istruzione.

Difendere la ROP

La ROP si basa sulla conoscenza esatta del layout del codice.

- Gli indirizzi dei codici sono l'inizio dei gadget ROP

Randomizzazione

- Randomizzazione del layout dello spazio degli indirizzi (ASLR) (`/proc/sys/kernel/randomize_va_space`)
- A grana fine

Posizione Codice indipendente

Fughe di informazioni

Bug che consentono di leggere il layout del processo

- Finora gli overflow sono stati utilizzati per sovrascrivere i dati di controllo.

ASLR

- È sufficiente rivelare l'indirizzo in cui è mappata una libreria condivisa
- Tutti i gadget vengono spostati in un nuovo offset.

Canarini da pila

- La rivelazione del contenuto della pila è sufficiente
- Il canarino è conservato nella pila

Grazie

Si prega di inviare tutte le domande a:
athanasopoulos.elias@ucy.ac.cy