

# Net&System Security 2006

October 17th, 2006

---

## Attacchi a Zone Adiacenti di Memoria dello Stack e Teoria della Complessità delle Vulnerabilità

Angelo P. E. Rosiello  
angelo@rosiello.org

Relatore - Fabrizio Ferrandi  
ferrandi@elet.polimi.it  
**Politecnico di Milano**



# Agenda

---

- Introduzione
- Tassonomia degli Attacchi più Comuni
- Attacchi a zone adiacenti di memoria: the environment
- Attacchi a zone adiacenti di memoria nello Stack: lo stato dell'arte ed un nuovo punto di vista
- Teoria della Complessità delle Vulnerabilità:
  - ▶ Motivazioni
  - ▶ Alcune Dimensioni
- Buffer Overflow e Attacchi a Zone Adiacenti di Memoria a Confronto
- Conclusioni



# Introduzione

---

- Con la diffusione di dispositivi digitali in grado di supportare del software il problema della sicurezza non è più “trascurabile” anche agli occhi dei meno esperti e/o “interessati”.
- Scopi degli attacchi:
  - ▶ Ottenere accesso non autorizzato a sistemi o ad informazioni riservate.
  - ▶ Estorsioni.
  - ▶ Furti d’identità.
  - ▶ Dinieghi di servizio.
  - ▶ Etc.



# Tassonomia degli Attacchi (1/2)

---

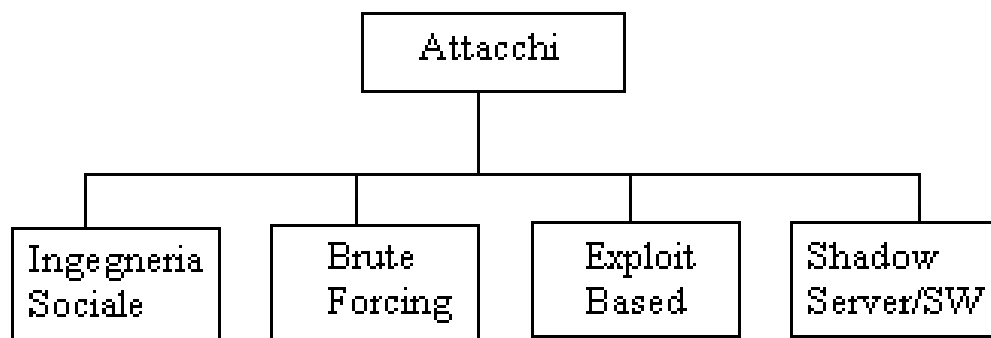
- Stiliamo una prima tassonomia degli attacchi più comuni esistenti in base alla natura dell'attacco stesso:
  - ▶ Ingegneria sociale - l'arte di persuadere la vittima al fine di estorcere informazioni sensibili.
  - ▶ Exploit-based - sfruttamento di vulnerabilità insite nel codice di un programma al fine di prendere il controllo o impedire il corretto funzionamento della macchina target per via remota o locale.



## Tassonomia degli Attacchi (2/2)

---

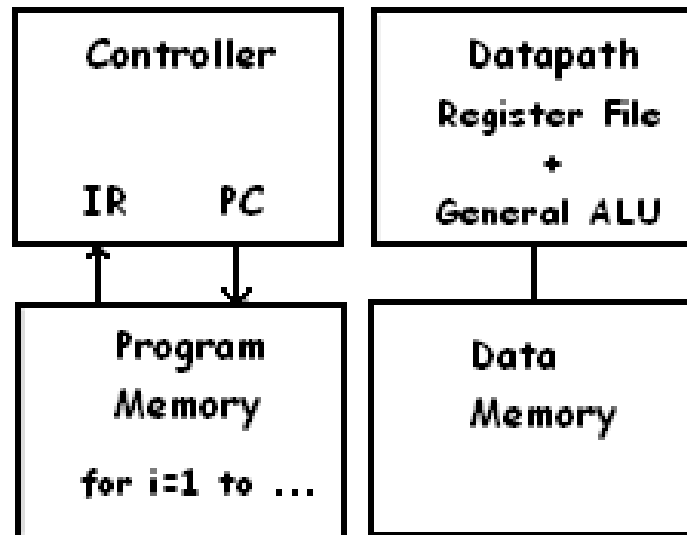
- ▶ Shadow server/software - perfetta copia di un server/software, oggetto dell'attacco, in grado di simulare l'interattività con l'utenza (vittima) al fine di catturare informazioni sensibili della vittima.
- ▶ Brute Forcing - tentativo di indovinare/scoprire i dati dell'utente (password, numero di carta di credito, etc.) provando tutte le possibili combinazioni di dominio in maniera deterministica ed algoritmica.



# Attacchi a zone adiacenti di memoria: *the environment (1/3)*

---

- Tipicamente gli attacchi a zone adiacenti di memoria hanno luogo in general-purpose processors (Intel, Sparc, etc.) o ASIP.



## Attacchi a zone adiacenti di memoria: *the environment (2/3)*

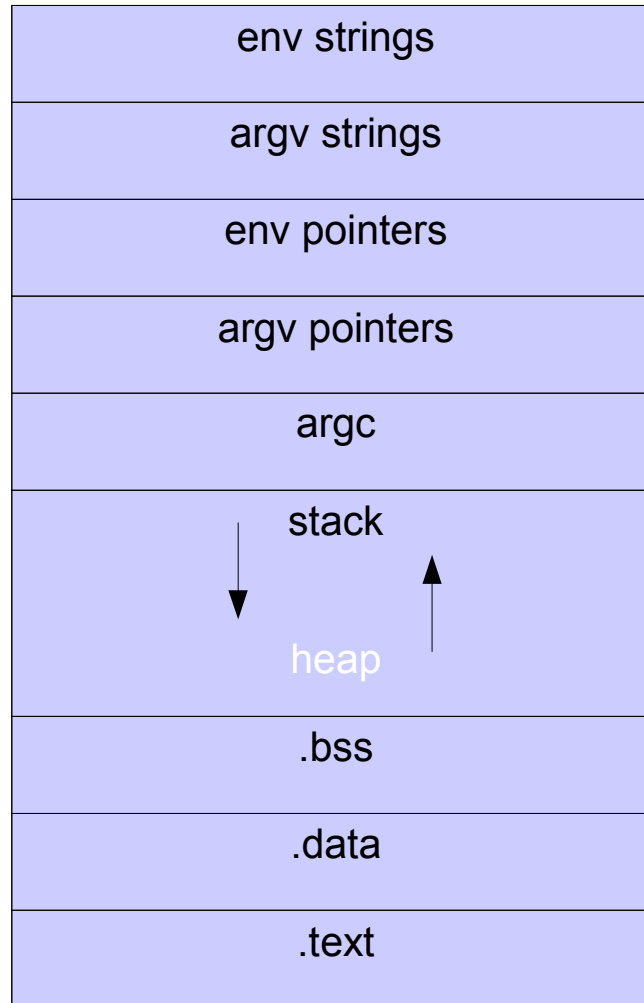
---

- Quando il codice oggetto di un programma viene letto e caricato in memoria per l'esecuzione, esso prende il nome di processo.
- L'intero spazio di memoria allocato per un processo prende il nome di spazio di indirizzamento e consiste di cinque settori principali:
  - ▶ Segmento Codice - contiene il codice eseguibile del programma.
  - ▶ Segmento Dati e BSS - entrambi i settori sono dedicati a memorizzare le variabili globali.
  - ▶ Stack.
  - ▶ Heap.



# Attacchi a zone adiacenti di memoria: *the environment (3/3)*

---



# Attacchi a zone adiacenti di memoria nello Stack: lo stato dell'arte

---

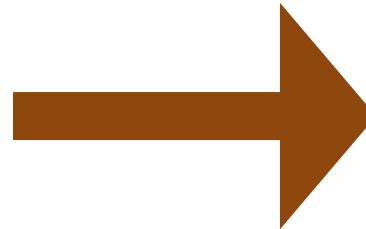
- Negli ultimi anni alcuni articoli [Twitch] hanno trattato gli effetti della concatenazione di stringhe nello stack.
- Pre-condizioni:
  - ▶ Il null-byte terminante un buffer 'X' viene sovrascritto.
  - ▶ Un buffer 'Y' segue 'X'.
- Effetti:
  - ▶ Concatenazione di 'X' ed 'Y' nello stack.



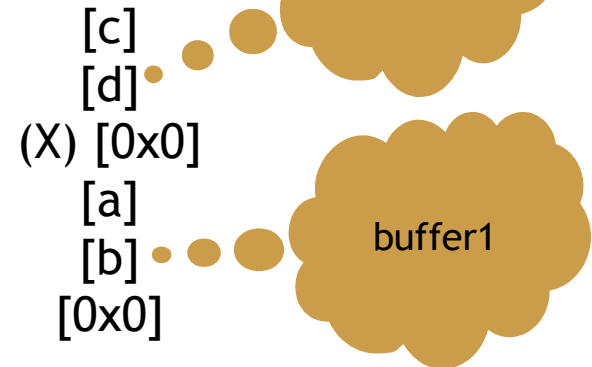
# Dichiarazione di un buffer

Quando un buffer viene dichiarato, esso viene terminato nello stack con un byte terminatore.

```
//Esempio 1
int main( ) {
    char buffer1[]="ab";
    char buffer2[]="cd";
    .....;
    return 0;
}
```



Stack Memory



Se si potesse sovrascrivere in qualche modo il null-byte indicato con (X) i due buffer risulterebbero concatenati.



# Funzioni “Pericolose...”

---

- Molte funzioni standard delle librerie C non terminano automaticamente un buffer con il null-byte.
- Ad esempio la funzione:  

```
char * strncpy(char *dst, const char *src, size_t len)
```
- “The strncpy() function copies at most len characters from src into dst. If src is less than len characters long, the remainder of dst is filled with ‘\0’ characters. Otherwise, dst is not terminated”.
- L’uso scorretto di queste funzioni provoca la concatenazione di buffer nello stack sotto certe condizioni ...



# Ma... dov'è la minaccia?

---

- Il problema diventa una minaccia per la sicurezza se ad esempio nel codice seguisse una *strcpy()* “ignara” degli effetti della concatenazione...

```
int main {
    char buf1[8];
    char buf2[4];
    fgets(buf1, 8, stdin);
    strncpy(buf2, buf1, 4);
    function(buf2);
}
void function( char buf2[32] ) {
    char buf3[8];
    strcpy( buf3, buf2 );
}
```

- Buf2 in *function()* dovrebbe contenere al più 4 (incluso '\0') caratteri ma post-concatenazione potrebbe presentarne fino a 12 ... stack overflow!



# Un Differente Punto di Vista

---

Cosa accadrebbe se *recv()* ritornasse un errore (-1) nel seguente esempio ?

```
int main( ) {
int i=0;
char buffer1[64];
char buffer2[64];
/* some code here that fills buffer1
   and buffer2 */ i=recv(...);
//controllo dell'overflow...
if(i>63) i=63;
buffer1[i]=i;
.....;
return 0;}
```

Buffer1 e Buffer2 risulterebbero concatenati...  
Analisi complessa se vi sono più funzioni annidate e magari mal documentate.



# Teoria della Complessità delle Vulnerabilità: Motivazioni

---

- Uno dei principali metodi adottati attualmente per “pesare” le vulnerabilità è di associare un livello di severità all’*advisory*.
- Molte software house, sebbene adottino processi di sviluppo consolidati e standardizzati, accompagnati da tecniche di testing/analisi avanzate, sono spesso soggette a vulnerabilità ad alta severità.
- Evidentemente è assente una qualche metrica che ci consenta di pesare le vulnerabilità individuate.



# Teoria della Complessità delle Vulnerabilità: le Dimensioni

---

- Per misurare la complessità di una vulnerabilità sono state prese in considerazione sei dimensioni emblematiche:
  - ▶ Novità Vulnerabilità/attacco.
  - ▶ Immediatezza dell'attacco.
  - ▶ Complessità del codice.
  - ▶ Complessità dell'attacco.
  - ▶ Ubiquità.
  - ▶ Potere dell'attaccante.
- Alla luce delle dimensioni individuate occorrerà costruire una serie storica delle vulnerabilità individuate nel ciclo di vita di un determinato pacchetto software, per valutare la maturità dell'intero processo di sviluppo.



## Le Dimensioni di Complessità (1/2)

---

- Novità Vulnerabilità/attacco - quanto è nuova alla comunità la tipologia di attacco o vulnerabilità identificata?
- Immediatezza dell'attacco - la riuscita dell'attacco è pressoché immediata o richiede più passi?
- Complessità del codice - quanto è latente la vulnerabilità nel codice? Metriche: profondità, immediatezza nell'individuazione, livello di indirezionalità, linee di codice, complex Halestead, etc.
- Complessità dell'attacco - numero di input da manipolare, complessità delle manipolazioni, numero di interfacce a cui accedere e controllare per apportare l'attacco.



## Le Dimensioni di Complessità (2/2)

---

- Ubiquità - numero di versioni, configurazioni, piattaforme affette dalla vulnerabilità.
- Potere dell'attaccante - grado di controllo richiesto all'attaccante sull'ambiente in cui l'attacco ha luogo. E.g.: diritti di accesso, numero di permessi, controllo su risorse esterne, etc.



# Buffer Overflow e Attacchi a Zone Adiacenti di Memoria a Confronto

Dimensioni	Buffer Overflow	Zone Adiacenti
Novità	Molto Conosciuto	Poco Conosciuto
Immediatezza	Immediato	Poco Immediato
Complessità del Codice	Dipende dall'applicazione	Dipende dall'applicazione
Complessità dell'attacco	Medio/Bassa	Medio/Alta
Ubiquità	Dipende dall'applicazione	Dipende dall'applicazione
Potere dell'attaccante	Basso	Medio/Alto



# Conclusioni

---

- Durante l'analisi ed il testing del software è necessario considerare entrambi gli scenari descritti causanti attacchi a zone adiacenti di memoria.
- Grazie alla teoria della complessità delle vulnerabilità ci auguriamo di aver gettato le fondamenta per una nuova metodologia atta a stabilire la qualità del software, ed implicitamente dei processi di sviluppo delle software house, avendo come obiettivo chiave la sicurezza dei prodotti finali.
- Lavori futuri riguarderanno l'analisi delle serie storiche delle vulnerabilità individuate nei pacchetti software soggetti ad analisi.



# Grazie per la Cortese Attenzione

---

## Politecnico di Milano

Angelo P.E. Rosiello

<http://www.rosiello.org>

[angelo@rosiello.org](mailto:angelo@rosiello.org)

## Ringraziamenti

Vivi ringraziamenti a Steve Christey, della “Common Vulnerabilities and Exposures”, per il significativo contributo in merito alla complessità delle vulnerabilità.

