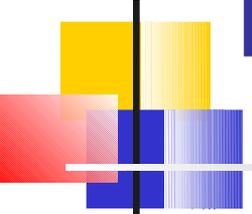


# Perchè questo corso?

---

- Fornisce nozioni base di sicurezza
  - utili a tutti
  - di particolare interesse per chi deve/ vuole/ dovrà sviluppare strumenti software di vasto utilizzo
- Non di laboratorio ma “pratico”
- Basato sui top 20 security controls
- Parte fissa (le nozioni) e parte variabile (esempi e strumenti)

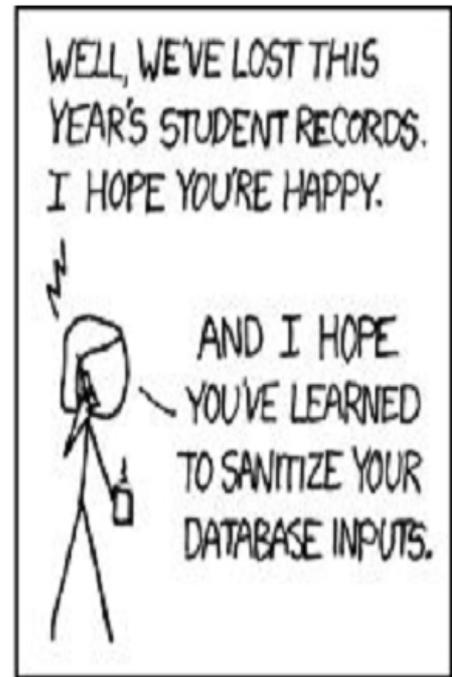
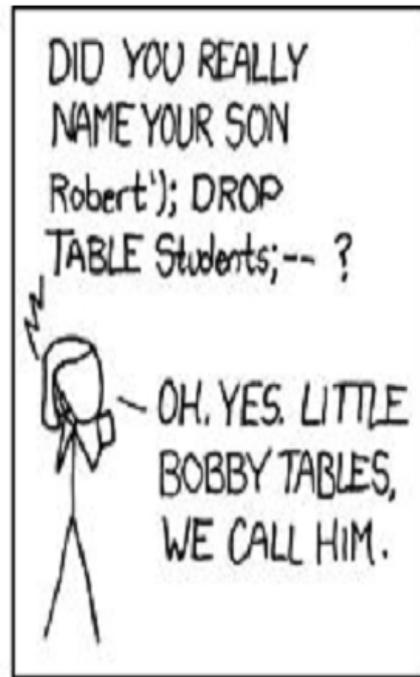


# Top 20 security controls

---

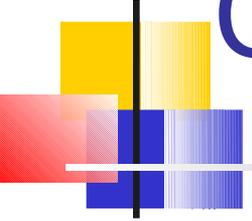
- SANS (associazione mondiale di sistemisti ed amministratori di rete) definisce i 20 controlli più importanti per un sistema ICT
- Controllo = meccanismo per fermare o impedire un attacco
- Il corso vi permette di capire questi controlli ed applicarli
- Approccio empirico ma di grande successo, sono poi state proposte liste analoghe per specifici settori (banche, sanità, ....) e per attacchi, vulnerabilità

# Una delle cose che capirete



# Che in fondo è anche questo ...

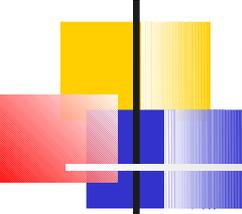




# Organizzazione del corso

---

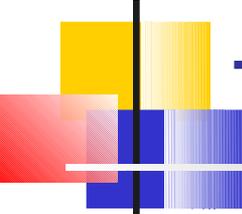
- Materiale didattico = lucidi inviati alla fine di un argomento ([f.baiardi@unipi.it](mailto:f.baiardi@unipi.it)) o messi su pagina web
- Approfondimenti per chi è interessato = libri ed articoli on line
- Esame = orale o seminario su argomento di interesse concordato



# Fondamenti di Sicurezza

---

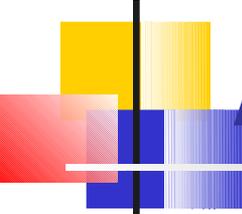
- Da un tema puramente tecnologico sta diventando qualcosa alla frontiera di almeno tre temi
  - Informatica
  - Organizzazione Aziendale
  - Economia
- Il concetto di rischio (= evento caratterizzato da probabilità di occorrenza e che può provocare danno), sua valutazione e gestione stanno diventando sempre più centrali nella sicurezza informatica
- Stiamo passando da sicurezza incondizionale (tipica del mondo militare) a sicurezza condizionale (commerciale)



# Temi trattati nel corso

---

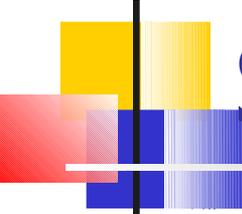
- Vulnerabilità, minacce
  - Come riconoscerle
- Attacchi, attacchi automatizzati
  - Effetti e pericolosità
- Contromisure
  - Sicurezza e virtualizzazione
  - Strumenti informatici per la difesa
- Assurance



# Attenzione a sicurezza

---

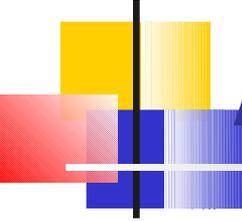
- un ente pubblico o privato è e sarà sempre più dipendente
  - dal suo sistema informativo
  - dal sistema informativo dei partner
  - dai sistemi informatici/telematici che li collegano
- Deve essere convinto e poter provare ad altri la propria sicurezza
- certificare il rispetto di un qualunque standard o modello (anche non di sicurezza) non è possibile se trascuriamo questa dipendenza



# Sicurezza Informatica

---

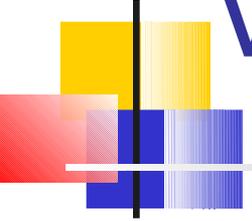
- **Confidenzialità**
  - Le informazioni possono essere lette solo da chi ne ha diritto
- **Integrità**
  - Le informazioni possono essere modificate solo da chi ne ha diritto
- **Disponibilità**
  - Le informazioni possono essere lette/scritte quando necessario
  - Le risorse devono poter essere usate solo da chi ne ha diritto



# Altre proprietà

---

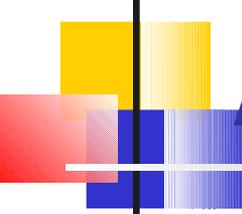
- Tracciabilità
  - individuazione di chi ha invocato una operazione
- Accountability
  - addebitare l'uso delle risorse
- Auditability
  - Poter verificare l'efficacia dei meccanismi utilizzati
- Forensics
  - Poter provare che certi attacchi hanno avuto luogo
- Privacy
  - Chi/come/se può usare le proprie informazioni personali



# Vulnerabilità

---

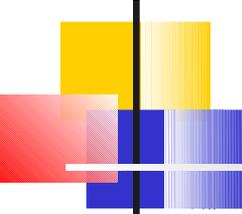
- Un difetto (bug, errore, ...) che permette comportamenti che violano le proprietà di sicurezza
- Tutte le vulnerabilità sono difetti, non tutti i difetti sono vulnerabilità



# Attacco ad un SI

---

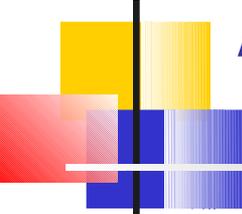
- Un attacco è una sequenza di azioni eseguite per ottenere il controllo di un SI
- L'attacco sfrutta una o più vulnerabilità del SI
- Attacco automatizzato = le azioni possono essere codificate in un programma (exploit)
- Quando l'attaccante controlla il SI subentrando al proprietario (owner) è in grado di
  - raccogliere informazioni
  - modificare informazioni
  - impedire ad altri di accedere alle informazioni



# Punto di vista del corso

---

- Attacco centrico = focalizzato sulla difesa cost effective da attacchi ad un sistema informativo
- Cosa rende possibile un attacco
- Come si possono gestire (impedire, limitare, scoprire) gli attacchi
- Costi, ricavi, sicurezza attesa ...



# Approcci alla sicurezza

---

- Sicurezza incondizionale
  - Si assume che qualsiasi sia la vulnerabilità nel sistema esista qualcuno interessato ed in grado di sfruttarla
- Sicurezza condizionale (analisi e gestione del rischio)
  - Considerare vulnerabilità e minacce effettive ed eliminare solo quelle vulnerabilità che qualcuno è interessato a sfruttare per attaccare il sistema

# Sfide per il 21 secolo

## Engineering's Grand Challenges

Click on the engineering challenge you think is the most important:



Make solar energy economical



Provide energy from fusion



Develop carbon sequestration methods



Manage the nitrogen cycle



Provide access to clean water



Restore and improve urban infrastructure



Advance health informatics



Engineer better medicines



Reverse-engineer the brain



Prevent nuclear terror



Secure cyberspace



Enhance virtual reality



Advance personalized learning



Engineer the tools of scientific discovery

## Secure cyberspace

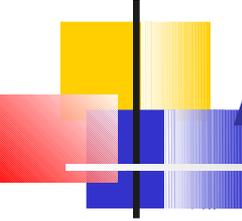
With input from people at this website -- an international community of technological thinkers who have shared their conclusions are revealed.



Watch the video

From urban centers to the depths of the oceans to space, we transcend barriers, overcome challenges, and improve life in our part of the world. In the last century alone, many of these achievements were so commonplace that we now take them for granted.





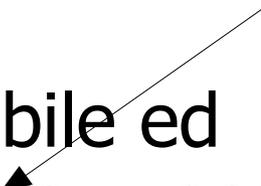
# Analisi del rischio

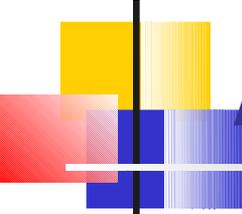
---

E' l'approccio più moderno al problema della sicurezza:

1. Analisi delle risorse da proteggere (asset)
2. Analisi delle vulnerabilità
3. Analisi degli attacchi
4. Analisi degli minacce
5. Analisi delle impatti
6. Individuazione rischio accettabile ed introduzione delle contromisure = gestione del rischio

I controlli

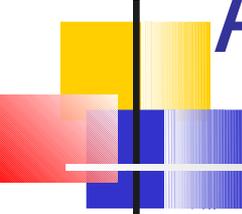




# Analisi del rischio

---

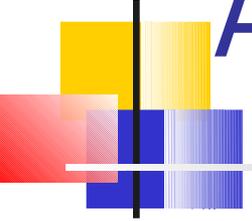
- Non si cerca di ottenere una sicurezza assoluta che è spesso impossibile
- Approccio guidato dall'economia
  - Cosa possiamo difendere
  - Cosa conviene difendere
- Non esistono ancora metodologie assodate esiste una foresta di metodi da semplificare



# Analisi delle risorse da proteggere

---

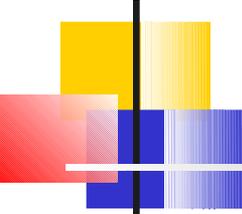
- Individuare un insieme di risorse logiche e fisiche di interesse per l'azienda/ente
- Definire proprietà di queste risorse in termini degli attributi precedenti
  - Chi può leggere alcune informazioni
  - Chi può modificare le informazioni
  - Chi può eseguire certi programmi
  - .....
- Queste proprietà diventano gli obiettivi di sicurezza che il sistema deve garantire



# Analisi delle vulnerabilità

---

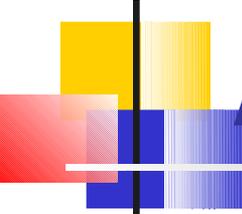
- Quali sono i difetti nelle varie componenti del sistema che permettono ad un attaccante di controllare, in un certo numero di passi gli asset di suo interesse
- I difetti possono essere scoperti in modo automatico o manuale
- Ogni vulnerabilità abilita alcuni attacchi = azioni che permettono di ottenere diritti



# Fasi di un attacco

---

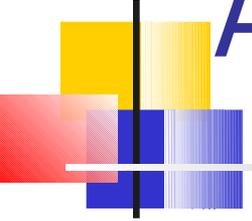
1. Raccolta di informazioni sul sistema
2. Individuazione delle **vulnerabilità** del sistema
3. Ricerca o costruzione di un **programma che sfrutti la vulnerabilità (=exploit)**
4. **Attacco**  $\Leftrightarrow$   
Esecuzione dell'exploit + Eventuali azioni umane
5. Installazione di strumenti per il controllo
6. Cancellazione delle tracce dell'attacco
7. Accesso, modifica, ..., ad un **sottoinsieme** delle informazioni del sistema o **altri attacchi**



# Attacco automatizzabile

---

- In questo caso le azioni umane non sono necessarie, basta eseguire un programma
- Attacco non automatizzabile è strutturalmente meno pericoloso di uno automatizzabile
- Gli attacchi automatizzabili costituiscono il vero carattere irripetibile della sicurezza informatica
  - Tempo di esecuzione dell'attacco
  - Scarse competenze richieste all'attaccante

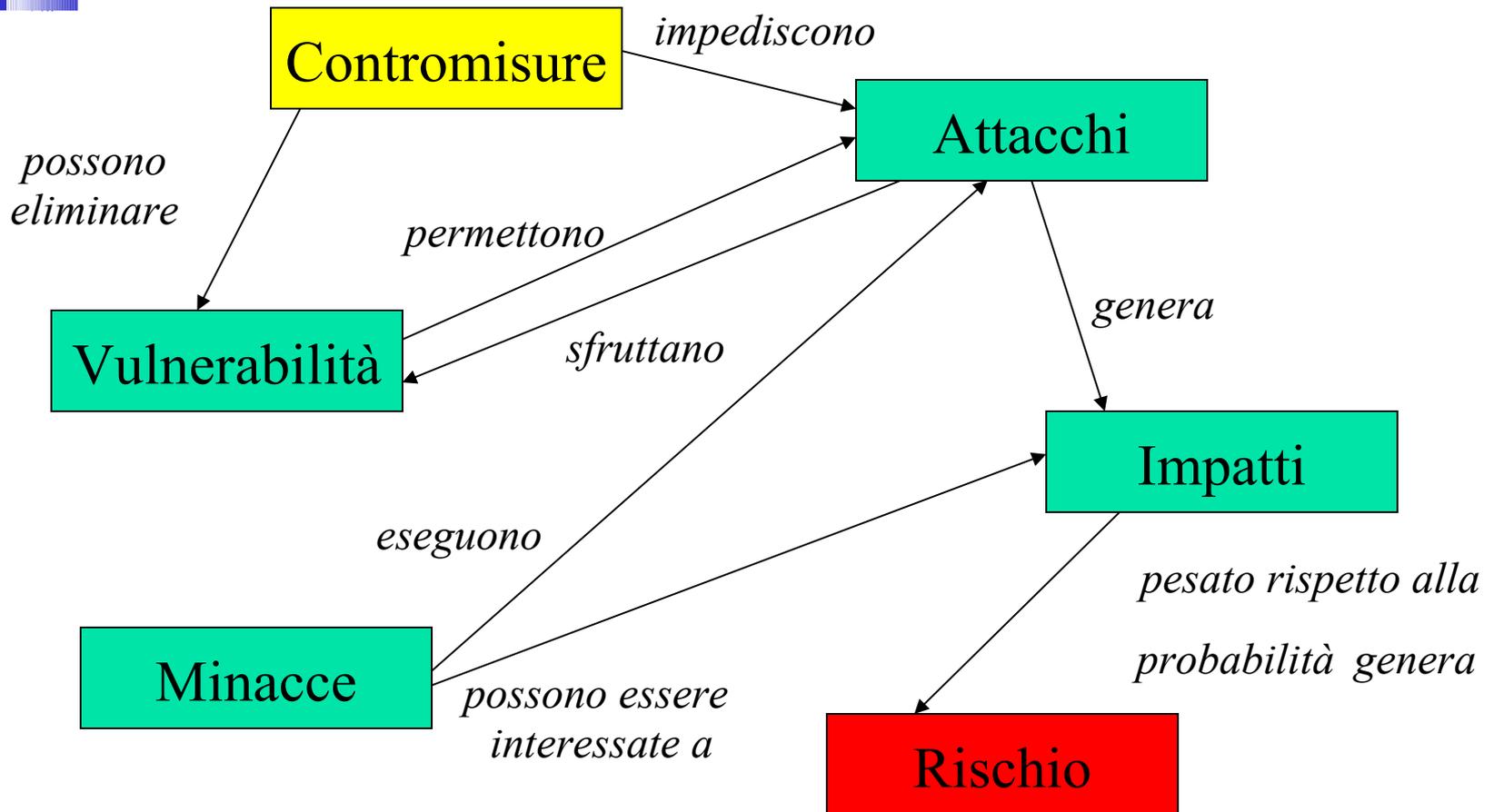


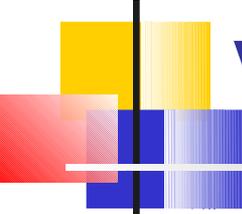
# Attacco locale/remoto

---

- Un attacco che può essere eseguito
  - purchè si disponga di un account su un sistema è detto **locale**
  - senza disporre di un account locale è detto **remoto**
- Un attacco automatizzabile e remoto è il più pericoloso

# Vulnerabilità , ....

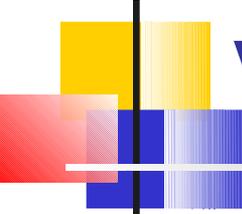




# VISIONI PARZIALI – I

---

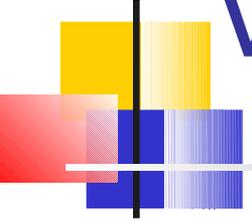
- Sicurezza = Confidenzialità  
⇔ Crittografia
- Un insieme di algoritmi per cifrare informazioni in modo che l'informazione anche se viene letta non viene letta in chiaro
- Contromisura importante ma parziale non permette di garantire disponibilità



# Visione parziale – II

---

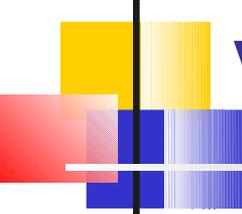
- Per risolvere i vari problemi di sicurezza occorre lavorare sulla tripla  
<utente, risorsa, diritti utente su risorsa>
- Ogni decisione legata alla sicurezza richiede la risoluzione di 3 problemi
  1. Identificazione utente
  2. Identificazione risorsa
  3. Esame dei diritti
- I sistemi di autenticazione (impronte digitali etc) permettono di risolvere il primo problema ma non gli altri due



# Visione Parziale - III

---

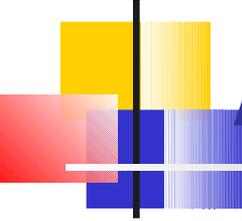
- La sicurezza è diversa dalla affidabilità  
security vs safety
- Un sistema che ha  $10^n - 1$  stati affidabili e 1 stato inaffidabile, ha una probabilità di non funzionare correttamente =  $1/10^n$  = affidabilità cresce con n
- Nel caso di sicurezza abbiamo un avversario intelligente che fa di tutto perchè il sistema entri in uno stato inaffidabile, non conta il numero di stati affidabili ma solo se l'avversario è in grado di far entrare il sistema nello stato



# Visione globale sulla sicurezza

---

- Ogni attacco è sostanzialmente un attacco al sistema operativo del sistema informativo
- La difesa (controlli, contromisure) passa soprattutto sulla difesa del/con/mediante il sistema operativo

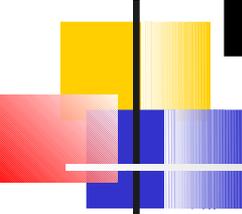


# Alcuni esempi

---

- Vulnerabilità
- Attacco
- Possibili contromisure

Esempio considerato è un attacco molto popolare lo stack overflow che è un caso particolare di buffer overrun



# Buffer overflow

---

Il problema dei *buffer overflow* è sicuramente quello più comune tra tutti i tipi di insicurezza nel codice C, mentre è praticamente assente in linguaggi di alto livello che non permettono al programmatore la gestione della memoria.

I problemi di *buffer overflow* sono stati la principale causa dei problemi di sicurezza riscontrati negli ultimi 10 anni.

La tecnica del buffer overflow consiste nel forzare la scrittura in memoria con una quantità di informazioni superiore a quella accettabile. Se il software è privo di controlli è possibile inserire del codice eseguibile (*bytecode*) in queste stringhe di overflow che consentono ad esempio di eseguire comandi su shell (*shellcode*). Inoltre se il software viene eseguito in modalità *root* un attacco di questo tipo può garantire il pieno possesso di tutte le funzionalità del sistema.

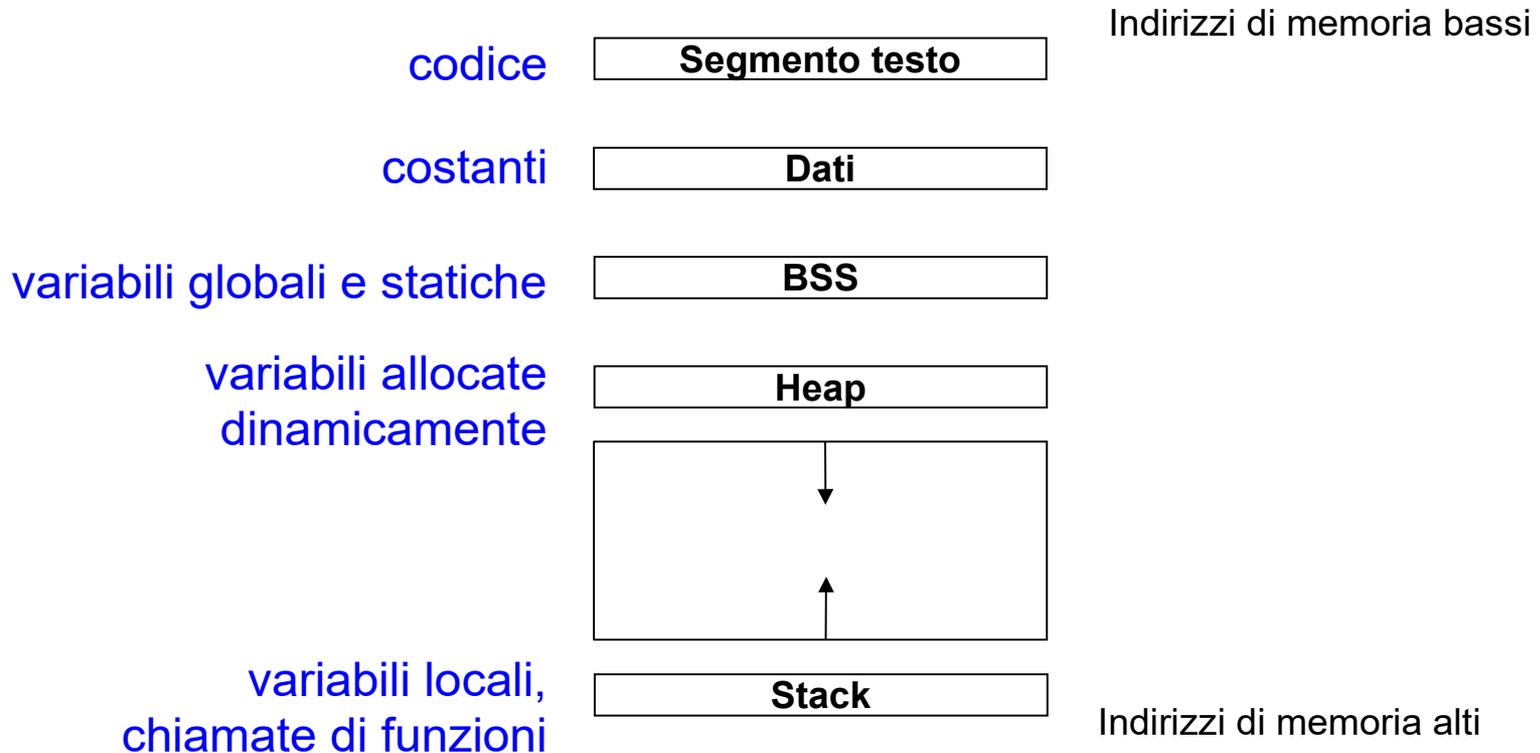
I buffer overflow possono essere eseguiti su diverse zone di memoria: *stack*, *heap* e *bss* (block started by symbol) area di variabili statiche allocate da compilatore.

# Organizzazione della memoria di un processo

- Per capire la tecnica del buffer overflow è necessario studiare l'organizzazione della memoria di un processo (programma).
- I processi sono divisi, in memoria, in tre regioni: *testo*, *dati* e *stack*. La regione *testo* è fissata, contiene il codice del programma ed è a sola lettura. Qualsiasi tentativo di scrittura provoca una violazione di segmento. La regione *dati* contiene i dati inizializzati e non (variabili statiche e globali) relativi al processo mentre la regione *stack* contiene i dati dinamici (utilizzati nella chiamata di funzioni).



# Organizzazione della memoria di un processo



... a volte si usa il disegno opposto ...

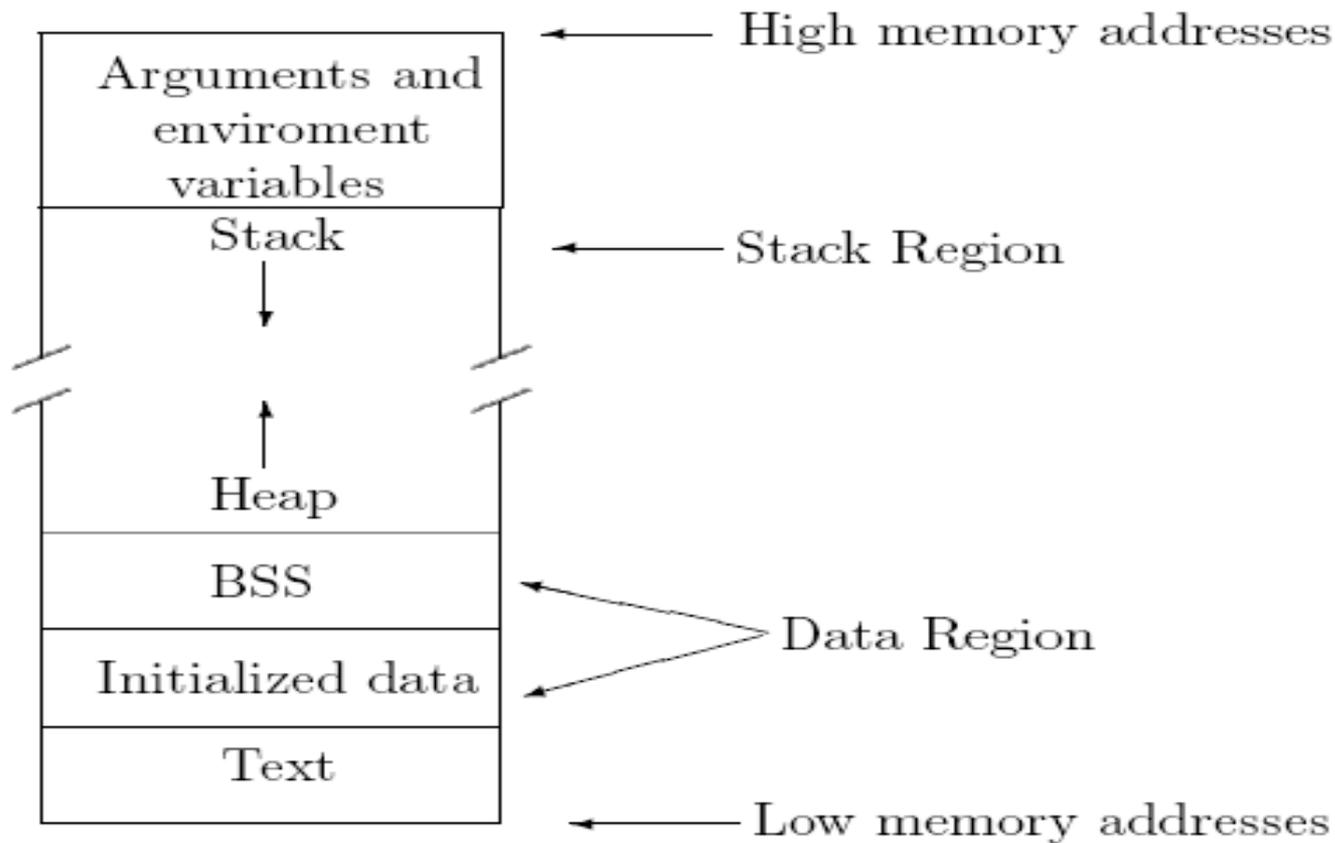
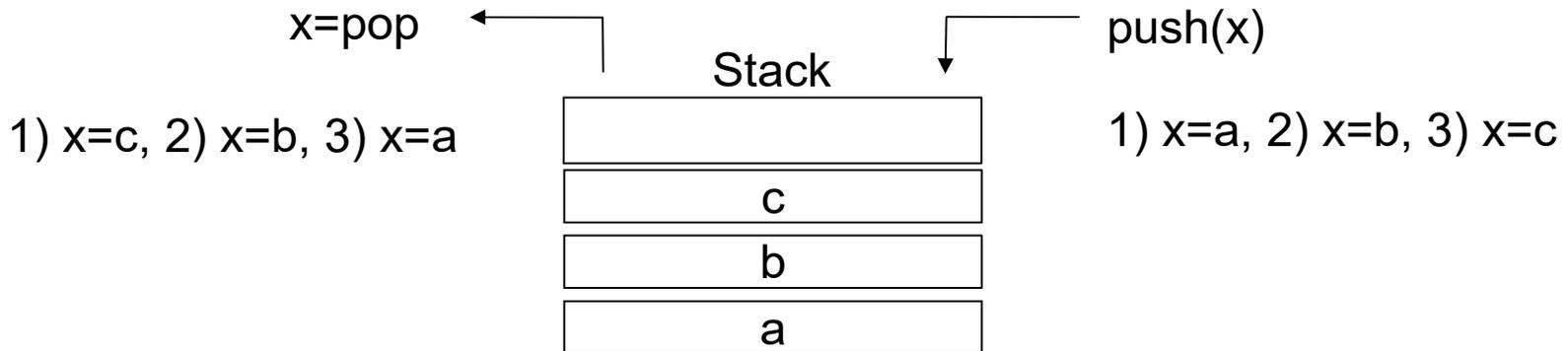
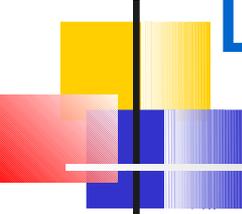


Figura 1: Semplificazione memory layout

# Lo Stack

- Lo stack (pila) è una struttura dati di tipo LIFO (Last In First Out) che memorizza un numero variabile di informazioni.
- Questa struttura dati viene utilizzata per gestire le chiamate di funzioni (call in assembly). La zona di memoria destinata alla gestione dello stack viene suddivisa logicamente in aree (stack frame) per ogni chiamata di funzione.





# Lo Stack e i registri di sistema

---

L'indirizzo di memoria dell'istruzione da eseguire, in un preciso istante, è contenuto nel registro di sistema **EIP** (Extended Instruction Pointer) a 32 bit (1dword).

# Lo Stack e i registri di sistema

- L'indirizzo di memoria dell'istruzione da eseguire, in un preciso istante, è contenuto nel registro di sistema **EIP** (Extended Instruction Pointer) a 32 bit (1dword)
- Il registro **EBP** (Extended Base Pointer) punta alla base di uno *stack frame* ed il registro **ESP** (Extended Stack Pointer) che punta alla cima dello stack frame.

record di attivazione  
della procedura

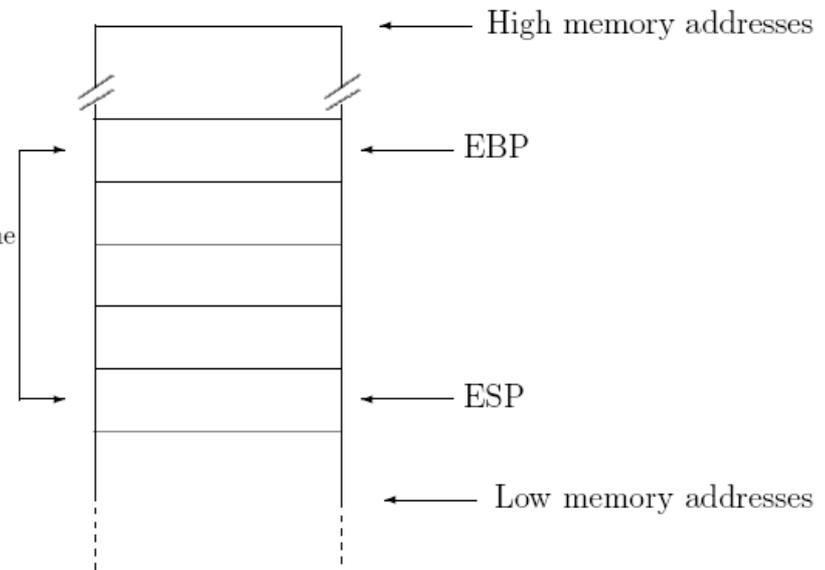
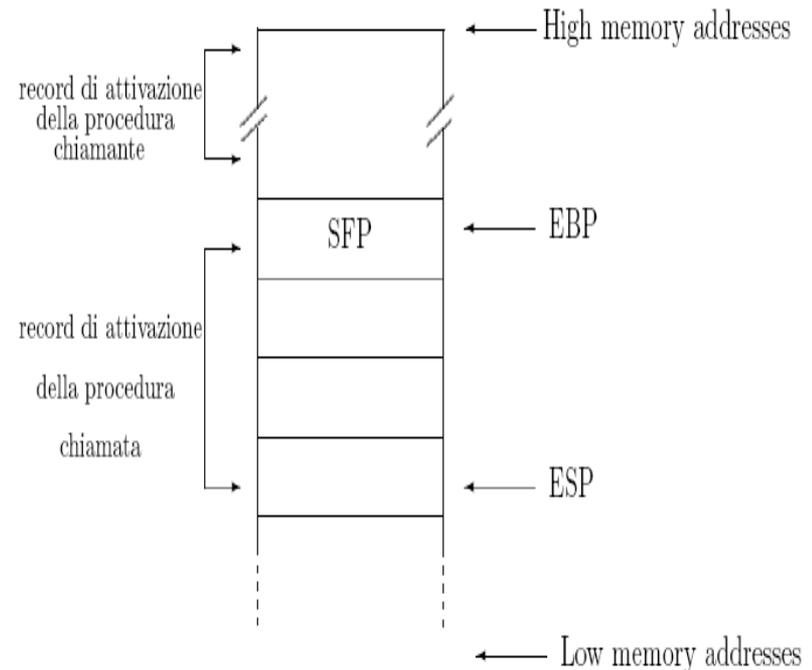
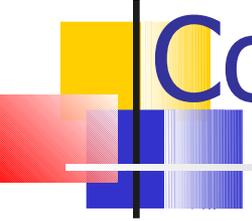


Figura 4: Stack pointer e frame pointer

# Lo Stack e i registri di sistema

Quando viene richiamata una funzione (call) il sistema inserisce nello stack l'indirizzo dell'istruzione successiva, **push(EIP+4)** dove 4 indica 4 byte (4byte=1dword), successivamente inserisce nello stack il puntatore alla base dello stack frame corrente, **push(EBP)** ed infine copia l'**ESP** attuale sull'**EBP** iniziando così il nuovo *stack frame*.





# Complessivamente

---

EIP, EBP, ESP

EIP va in stack

EBP va in stack (chiamato SFP)

$EBP' = ESP$

Settato nuovo  $ESP'$  (lunghezza della  
procedura chiamata)

# Un esempio in C

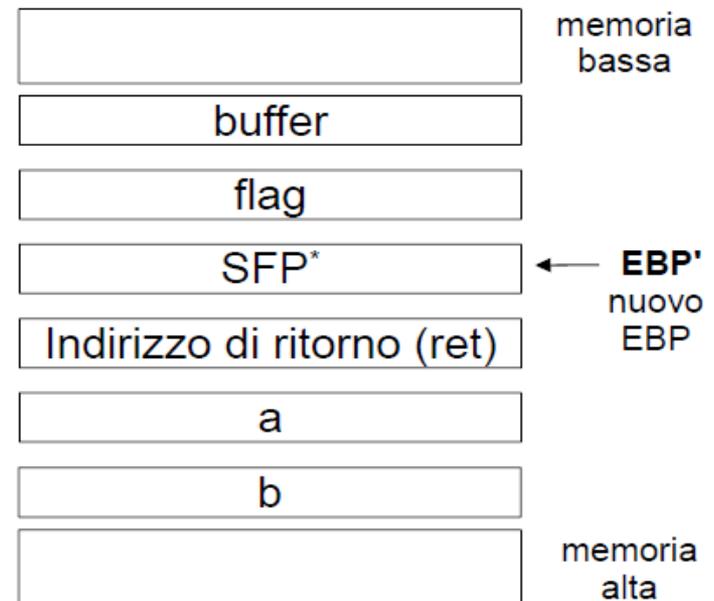
Analizziamo quest'esempio di codice in C per capire meglio l'allocazione dello *stack frame*:

```
void test_function (int a, int b)
{
    char flag;
    char buffer[10];
}
```

```
int main()
{
    test_function (1,2);
    exit(0);
}
```

EIP → Stato dello stack frame →

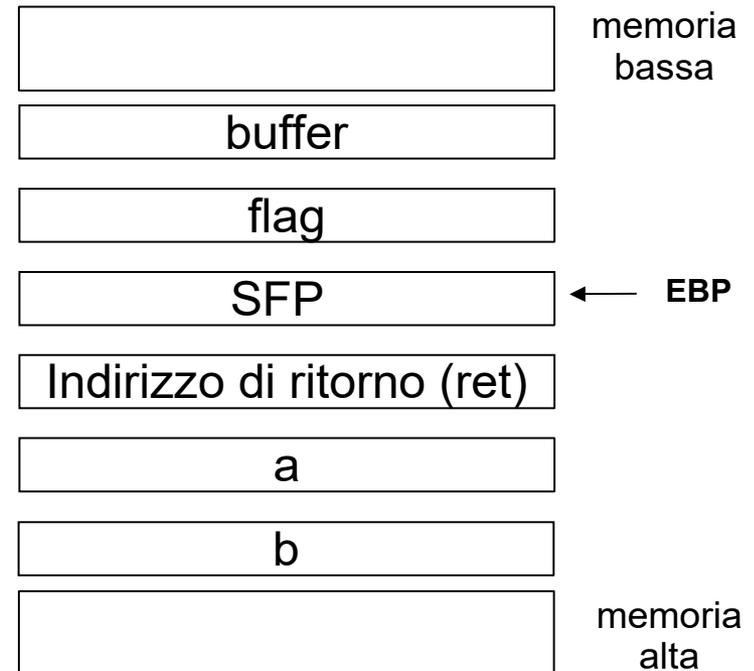
← Indirizzo di ritorno (ret) = EIP + 4 byte



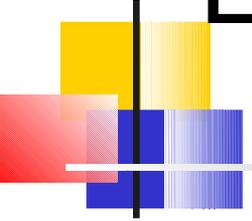
\***SFP**= Saved Frame Pointer, valore utilizzato per ripristinare lo stato originale di EBP (prima della chiamata di `test_function()`);

# Lo stack frame

- Alle variabili locali della funzione *test\_function* si fa riferimento mediante sottrazione del valore del frame pointer **EBP** e gli argomenti della funzione mediante addizione a tale valore.
- Quando una funzione viene richiamata, il puntatore **EIP** diventa l'indirizzo di inizio del codice della funzione.
- La memoria dello stack è utilizzata per le variabili locali e gli argomenti della funzione. Dopo il termine dell'esecuzione della funzione, l'intero *stack frame* viene estratto dallo stack in modo da riprendere l'esecuzione sull'**istruzione di ritorno (ret)**.



# Esempio di overflow



---

Analizziamo quest'esempio di codice in C che provoca un overflow:

```
void overflow_function (char *str) {  
    char buffer[20];  
  
    strcpy(buffer, str); // Funzione che copia str nel buffer  
}
```

```
int main() {  
    char big_string[128];  
    int i;  
  
    for(i=0; i < 128; i++)  
    {  
        big_string[i] = 'A';  
    }  
    overflow_function(big_string);  
    exit(0);  
}
```

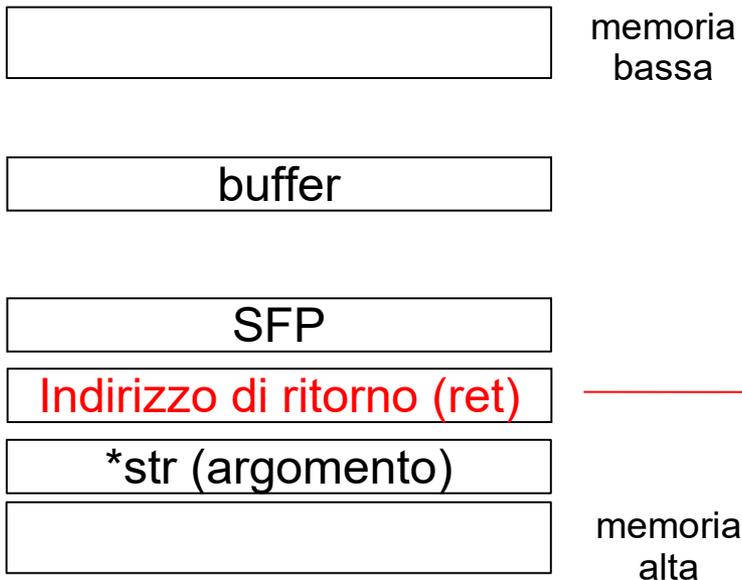
La procedura non  
controlla lunghezza.  
Questa istruzione  
provoca un overflow!



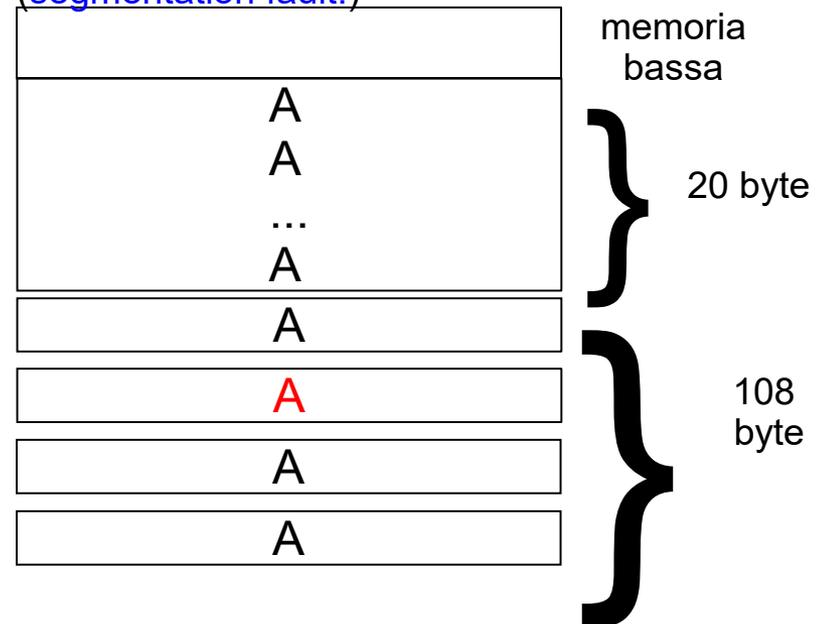
# Segmentation fault

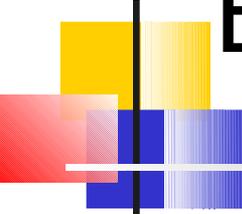
Perchè il codice precedente provoca un *Segmentation fault*?

1) La prima chiamata di *overflow\_function* inizializza correttamente lo stack frame:



2) Al momento del termine dell'esecuzione della funzione *overflow\_function*, l'istruzione di ritorno è stata sovrascritta con il carattere **A** (*segmentation fault!*)



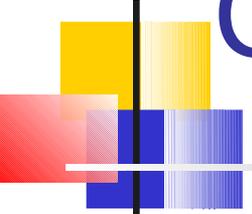


# Buffer (stack) overflow

---

Cosa succede se l'*istruzione di ritorno* (**ret**) contiene un indirizzo di memoria valido?

- In questo caso il processo continuerebbe indisturbato eseguendo l'istruzione successiva contenuta in **ret**.
- Il *buffer overflow* basato sullo stack consiste proprio nello sfruttare tale possibilità sostituendo l'istruzione di ritorno **ret** con un nuovo puntatore ad una porzione di codice inserita manualmente da un intruso.
- Come è possibile
  - modificare tale *istruzione di ritorno* ed
  - inserire arbitrariamente del codice in un processo?



# Cos'è un Buffer Overrun

---

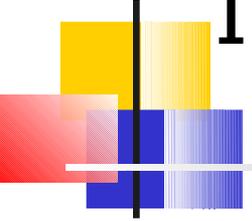
- Si verifica quando i dati superano la dimensione prevista e sovrascrivono altri valori
- È frequente soprattutto nel codice C/C++ non gestito
- Può essere di quattro tipi:
  - buffer overrun basato sullo stack
  - buffer overrun dell'heap
  - Sovrascrittura della v-table e del puntatore a funzione
  - Sovrascrittura del gestore eccezioni
- Può essere sfruttato dai worm per implementare un attacco automatico

# I buffer overrun dell'heap

Sovrascrivono i dati memorizzati nell'heap  
Sono più difficili da sfruttare di un buffer  
overrun

**strcpy**

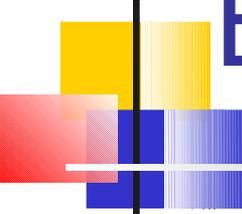




# I buffer overrun dell'heap: example 1

---

```
class CustomerRecord
{
private:
    char szName[20];
    char szAddress[10];
    char szPassword[20];
    char szCreditHistory[200];
    char szBankDetails[25];
```



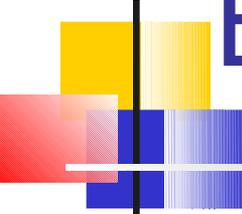
# Esempio -1

---

```
void ChangeAddress (char *a)
{
    strcpy (szAddress, a);
}
```

```
bool ChangePassword (char *newpwd, char *oldpwd)
{
    bool res=false;

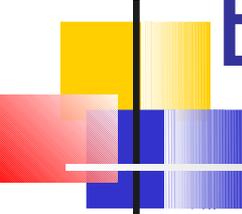
    if (strcmp (oldpwd, szPassword)==0)
    {
        strcpy (szPassword, newpwd);
        res=true;
    }
    return res;
}
```



# Esempio - 2

---

```
char * GetSensitiveData (char *pwd)
{
    if (strcmp (pwd, szPassword)==0)
    {
        // return all the personal info!
        return "Here's all the personal info...\n";
    }
    else
        return "";
}
```



# Esempio

---

Eseguire `procedura`

Come faccio ad ottenere

`GetSensitiveData password`

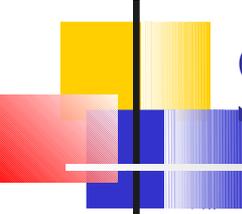
Se non conosco la password=secret??

Trucco: invoco `ChangeAddress ...` 😊

Buffer overflow su ?? Heap!! 😊

```
char szAddress[10];
```

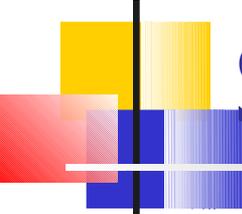
```
char szPassword[20];
```



# Stack Overflow

---

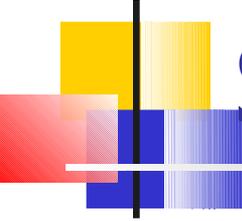




# Stack Overflow

---

- Quando viene copiato il valore di x si distrugge
  - il punto di ritorno
  - altri valori sullo stack
- Si genera un programma memorizzato sullo stack
- Quando il programma preleva il punto di ritorno salta ad un indirizzo "dentro" il nuovo valore di x
- Effetto finale: accesso ad una shell da amministratore
- Attacco è ovviamente utile solo se la procedura attaccata è eseguita da un amministratore.

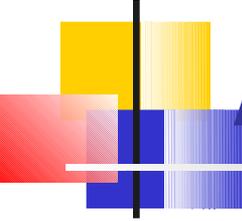


# Stack overflow

---

Vulnerabilità = visioni alternative

1. mancanza di controlli nel programma
2. tipi mal gestiti nel programma
3. operazioni di memoria malgestite
4. stack che cresce verso il basso
5. ...

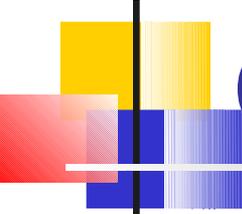


# Altri overflow

---

- Buffer overflow
- Heap overflow
- Format string overflow

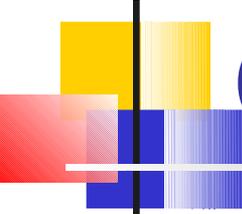
Richiedono comunque di violare il meccanismo dei tipi del linguaggio



# Overflow: contromisure possibili

---

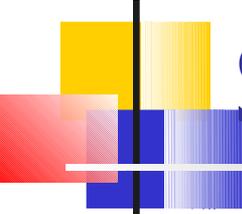
- Tipi forti
- Inserzione di controlli su lunghezza stringhe
- Inserzione di "canary"
- Rendere la memoria "non eseguibile"
- Address Space Layout Randomization
- Check ad hoc nel compilatore



# Canary

---

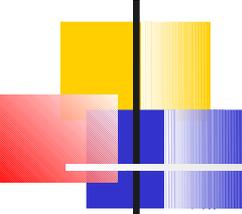
- Un valore che cambia ad ogni invocazione di procedura
- Inserito nello stack prima di ogni altro parametro
- Prima di ritornare si controlla che non sia stato cambiato
- Generato ogni volta per impedire copia da parte dell'attaccante



# Stack non eseguibile

---

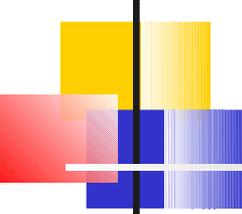
- Si utilizzano controlli in fase di traduzione degli indirizzi (con MMU ho anche un supporto hw)
- A tutto lo spazio che memorizza delle strutture dati non si assegna il diritto di esecuzione
- Bit NX in architetture AMD
- Non funziona in Linux che usa lo stack per i driver



# Address Space Layout Rand.

---

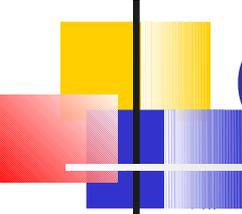
- Allocazione casuale dei vari segmenti
- Attaccante non può conoscere in anticipo l'indirizzo iniziale dei vari segmenti
- Attacchi devono avere un preambolo che calcola l'indirizzo iniziale dei vari segmenti
- Creati attacchi che sfruttano politiche di rimpiazzamento nel cache condiviso per scoprire indirizzi



# ASLR – entropia

---

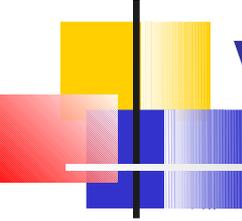
Type	Description	Protection	Granularity of Rebasing
Free	Free space	Inaccessible	Not rebased
Code	Executable or DLL code	Read-only	15 bits
Static data	Within executable or DLL	Read-Write	15 bits
Stack	Process and thread stacks	Read-Write	29 bits
Heap	Main and other heaps	Read-Write	20 bits
TEB	Thread Environment Block	Read-Write	19 bits
PEB	Process Environment Block	Read-Write	19 bits
Parameters	Command-line and Environment variables	Read-Write	19 bits
VAD	Returned by virtual memory allocation routines	Read-Write	15 bits
VAD	Shared Info for kernel and user mode	Unwritable	Not rebased



# Costi delle contromisure

---

- Quello che cambia è il costo
  - Tipi forte = costo max = 10-30%
  - Controlli su lunghezza stringhe = costo elevato ma meno del precedente
  - Canary = controllo specifico
  - Rendere lo stack non eseguibile = costo minimo, sfrutto meccanismo di protezione delle aree di memoria



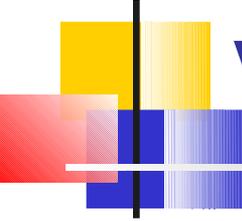
# Vulnerabilità Strutturali TCP/IP

---

Quando è stato sviluppato lo stack TCP/IP uno degli obiettivi era costruire una rete che sopravvivesse ad attacchi fisici distruttivi dei componenti (minaccia=bombardamento) = obiettivo era la disponibilità

⇒ Esistono dei meccanismi per scoprire se alcuni nodi sono vivi e raggiungibili

⇒ Non esistono dei meccanismi per garantire l'origine delle informazioni



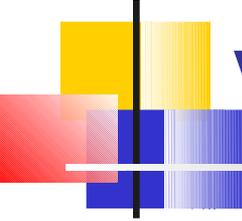
# Vulnerabilità Strutturali TCP/IP

---

Quando è stato sviluppato lo stack TCP/IP uno degli obiettivi era costruire una rete che sopravvivesse ad attacchi fisici distruttivi dei componenti (minaccia=bombardamento) = obiettivo era la disponibilità

⇒ Esistono dei meccanismi per scoprire se alcuni nodi sono vivi e raggiungibili

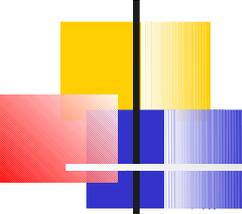
⇒ Non esistono dei meccanismi per garantire l'origine delle informazioni



# Vulnerabilità strutturali -Esempio

---

1. Per controllare se un nodo è vivo si invia un messaggio di ECHO, a cui il nodo risponde con lo stesso messaggio
2. Esiste un modo di dare un indirizzo IP parziale per realizzare broadcast ad un insieme di nodi
3. Non esiste un controllo sui campi di un pacchetto IP

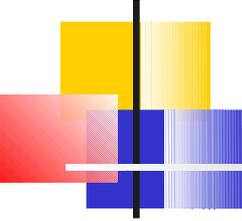


# Mettiamo il tutto insieme ..

---

- R è una rete di 1000 nodi, con indirizzo parziale X, un indirizzo con meno di 32 bit comune a tutti i nodi di R
- A manda un messaggio di ECHO a X fingendo di essere B
- Tutti i nodi di R rispondono a B
- B per un certo tempo non può comunicare perché le sue linee sono intasate

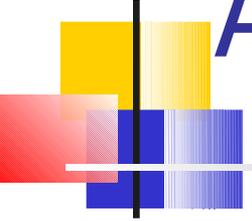
***Distributed Denial of Service***



# Effetti e prevenzione

---

- L'effetto si può amplificare a piacere aumentando il numero di nodi che hanno il ruolo di A (zombies)
- Questo è un tipico attacco da cui è estremamente difficile difendersi perché B non ha un modo di scoprire che l'attacco è in preparazione



# Approcci al progetto

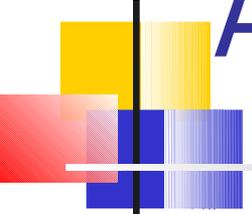
---

Possiamo assumere che nei componenti che componiamo per costruire un sistema

a) non ci siano vulnerabilità

(penetrate and patch)

b) ci siano anche se non le conosciamo e costruiamo il sistema in modo da compensare le vulnerabilità che non conosciamo ancora (proattivo)

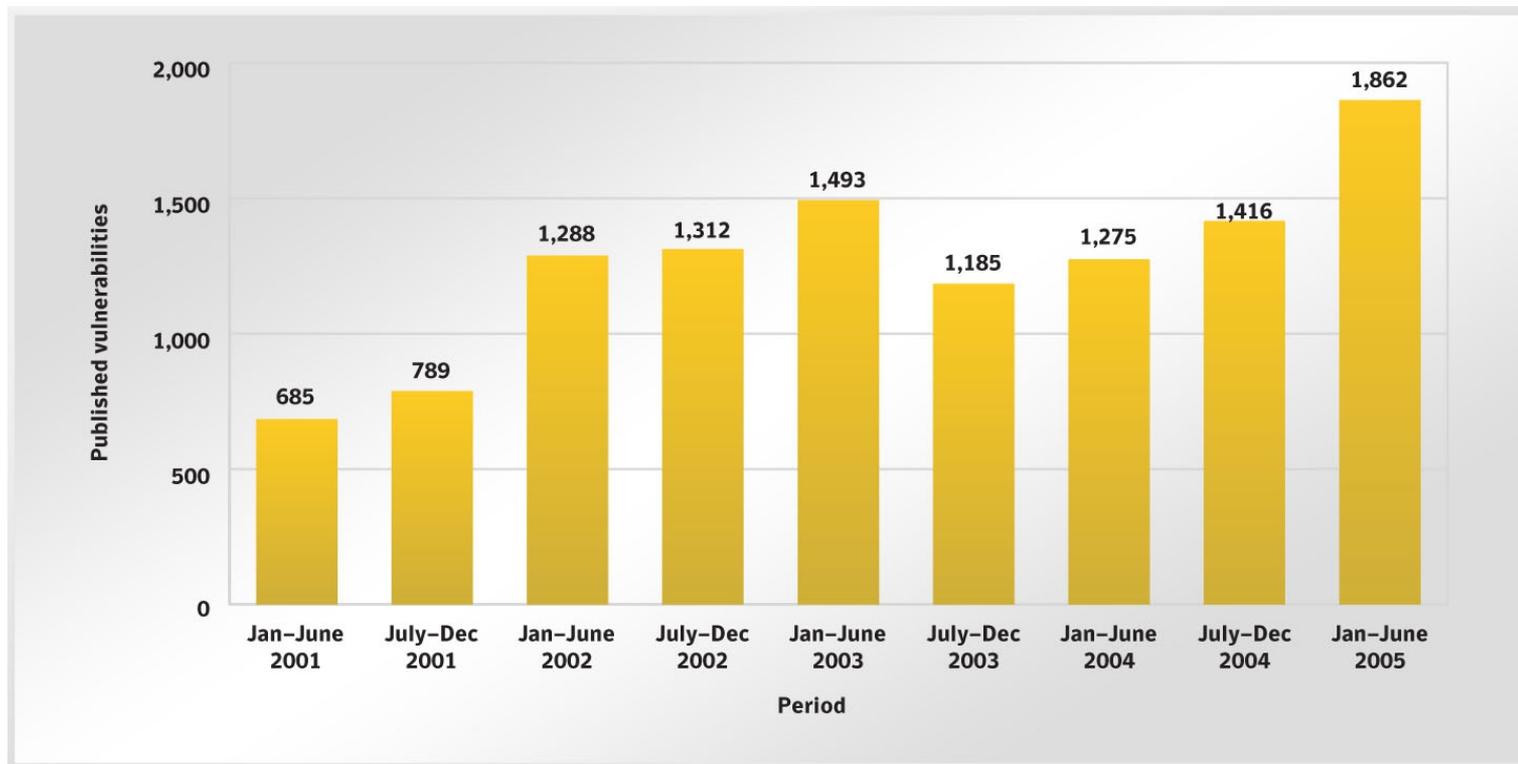


# Approccio penetrate and patch

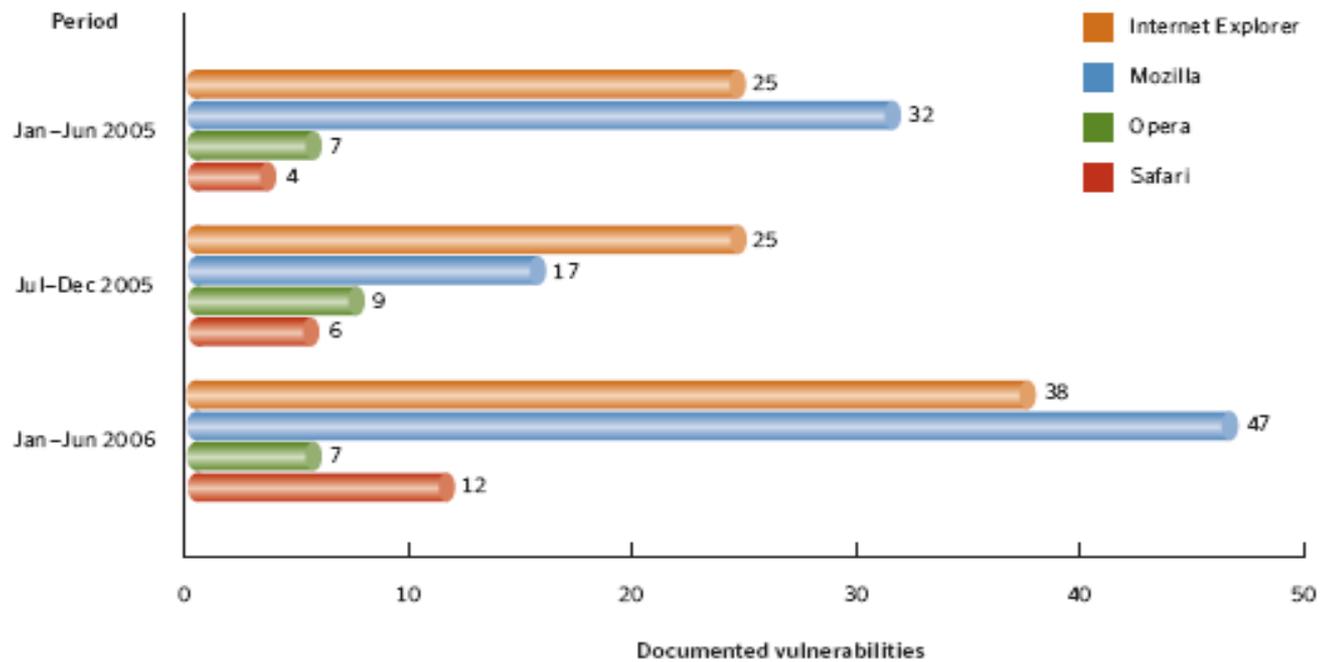
---

- Poiché il progetto di un sistema ha assunto che tutti i componenti fossero perfetti occorre eliminare immediatamente le vulnerabilità che vengono scoperte
- Abbiamo una corsa tra chi scopre le vulnerabilità e chi le elimina
- La presenza di vulnerabilità non è stata "anticipata"

# Numero di vulnerabilità scoperte



# Numero di vulnerabilità browser



**Figure 5. Web browser vulnerabilities**

*Source: Symantec Corporation*

# Vulnerabilità: gli ultimi 16 anni

Navigation: Home, SCAP, Checklists, 800-53/800-53A, Product Dictionary, Impact Metrics, Data Feeds, Statistics, FAQs

## Statistics Results Page (Refine Search)

[New Query](#)

Below are a table and graphs with data matching the characteristics you specified on the Statistics Query Page.

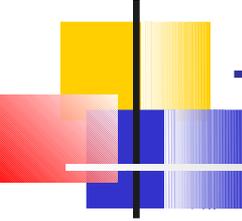
### Search Parameters:

- Contains Software Flaws (CVE)
- Publication Start Date: January 2000
- Publication End Date: December 2016
- CVSS Version: 3

### Total Matches By Year

Year	Number of Vulnerabilities
2000	1000
2001	2000
2002	3000
2003	2500
2004	3500
2005	5000
2006	7000
2007	7000
2008	5500
2009	6000
2010	4500
2011	4000
2012	5500
2013	5000
2014	8500
2015	7000
2016	7000

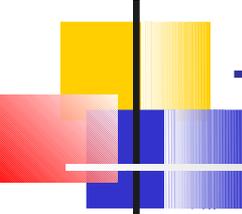
### Percent Matches By Year



# Top 10 Vulnerabilities - Windows Systems

---

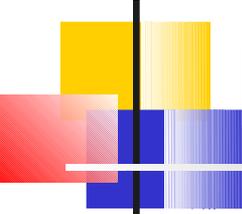
1. Internet Information Services
2. Microsoft SQL Server
3. Windows Authentication
4. Internet Explorer
5. Windows Remote Access Services
6. Data Access Components(MDAC
7. Windows Scripting Host
8. Outlook and Outlook Express
9. Peer to Peer File Sharing
10. Simple Network Management



# Top 10 Vulnerabilities - Unix Systems

---

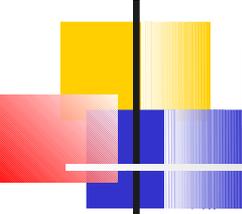
1. BIND Domain Name System
2. Remote Procedure Calls (RPC)
3. Apache Web Server
4. Accounts with No Passwords or Weak Passwords
5. Clear Text Services
6. Sendmail
7. Simple Network Management Protocol
8. Secure Shell (SSH)
9. Misconfiguration of NIS/NFS
10. Open Secure Sockets Layer (SSL)



# Lista - I

---

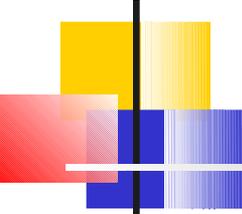
- Top Vulnerabilities in Windows Systems
  - W1. Windows Services
  - W2. Internet Explorer
  - W3. Windows Libraries
  - W4. Microsoft Office and Outlook Express
  - W5. Windows Configuration Weaknesses
- Top Vulnerabilities in Cross-Platform Applications
  - C1. Backup Software
  - C2. Anti-virus Software
  - C3. PHP-based Applications
  - C4. Database Software
  - C5. File Sharing Applications
  - C6. DNS Software
  - C7. Media Players
  - C8. Instant Messaging Applications
  - C9. Mozilla and Firefox Browsers
  - C10. Other Cross-platform Applications



# Lista - II

---

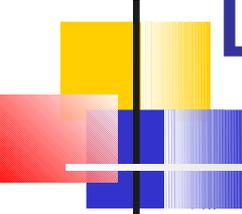
- Top Vulnerabilities in UNIX Systems
  - U1. UNIX Configuration Weaknesses
  - U2. Mac OS X
- Top Vulnerabilities in Networking Products
  - N1. Cisco IOS and non-IOS Products
  - N2. Juniper, CheckPoint and Symantec Products
  - N3. Cisco Devices Configuration Weaknesses



# Lista Hipaa

---

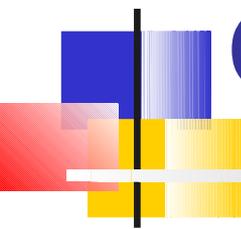
- Firewall and System Probing
- Network File Systems (NFS) Application
- Electronic Mail Attacks
- Vendor Default Password Attacks
- Spoofing, Sniffing, Fragmentation and Splicing
- Social Engineering Attacks
- Easy-To-Guess Password
- Destructive Computer Viruses
- Prefix Scanning (Illegal Modem)
- Trojan Horses



# Lista Hippa

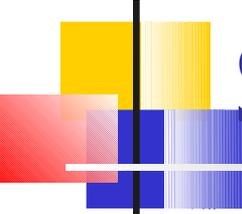
---

- Dimostra come un difetto possa diventare una vulnerabilità nel momento in cui venga introdotta una legge che vieta il comportamento permesso dalla vulnerabilità
- Ovviamente anche questa vulnerabilità deve essere eliminata



# Ciclo di vita di una vulnerabilità

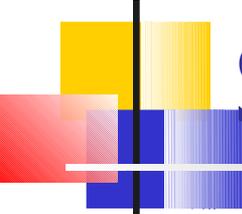
---



# Stato di una vulnerabilità - 1

---

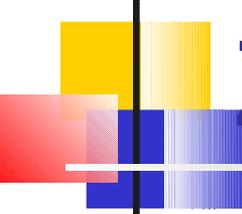
1. Si conosce esistenza di vulnerabilità
2. Si conosce esistenza di vulnerabilità e di un attacco (exploit) che la sfrutta
3. Si conosce esistenza di vulnerabilità e di una patch che la elimina
4. Si conosce esistenza di vulnerabilità, di una patch che la elimina e di un attacco



## Stato di una vulnerabilità - 2

---

- Spesso i sistemi vengono attaccati con successo anche se lo stato della vulnerabilità è l'ultimo
- Lentezza degli utenti nell'applicare la patch anche se è nota
- Disparità tra cliente e fornitore (non esiste il richiamo di un modello, l'utente deve aggiornare da solo il sistema)



# Zero day exploit

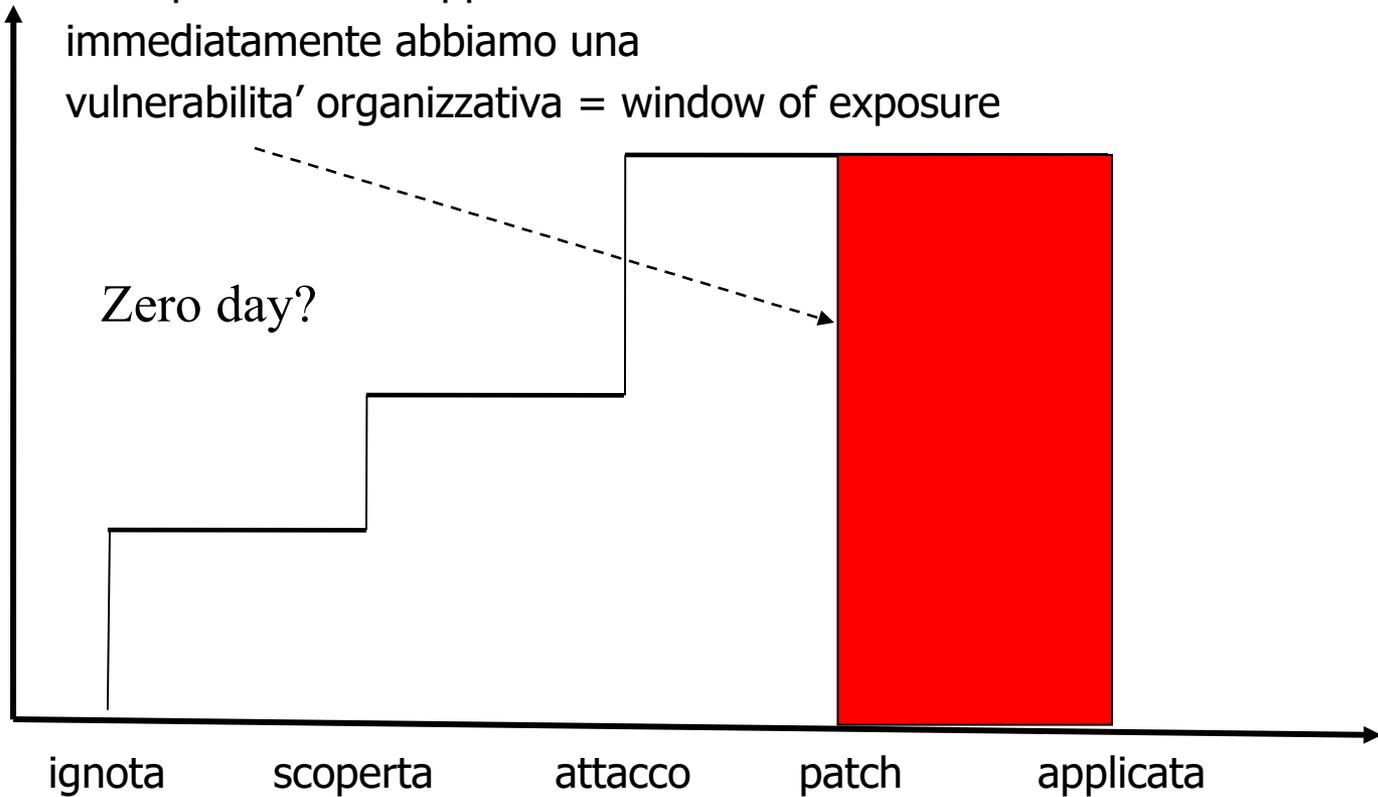
---

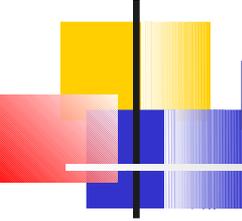
- Attacchi per vulnerabilità che non sono ancora pubbliche
- Non sempre chi scopre un difetto lo rende pubblico, il silenzio può essere più redditizio
- È possibile costruire sistemi in modo che non perdano sicurezza anche in presenza di vulnerabilità non ancora pubbliche?
- Sempre più diffuse ...

# Pericolosità di una vulnerabilità

Se la patch non e' applicata  
immediatamente abbiamo una  
vulnerabilita' organizzativa = window of exposure

Pericolosità  
della  
vulnerabilità



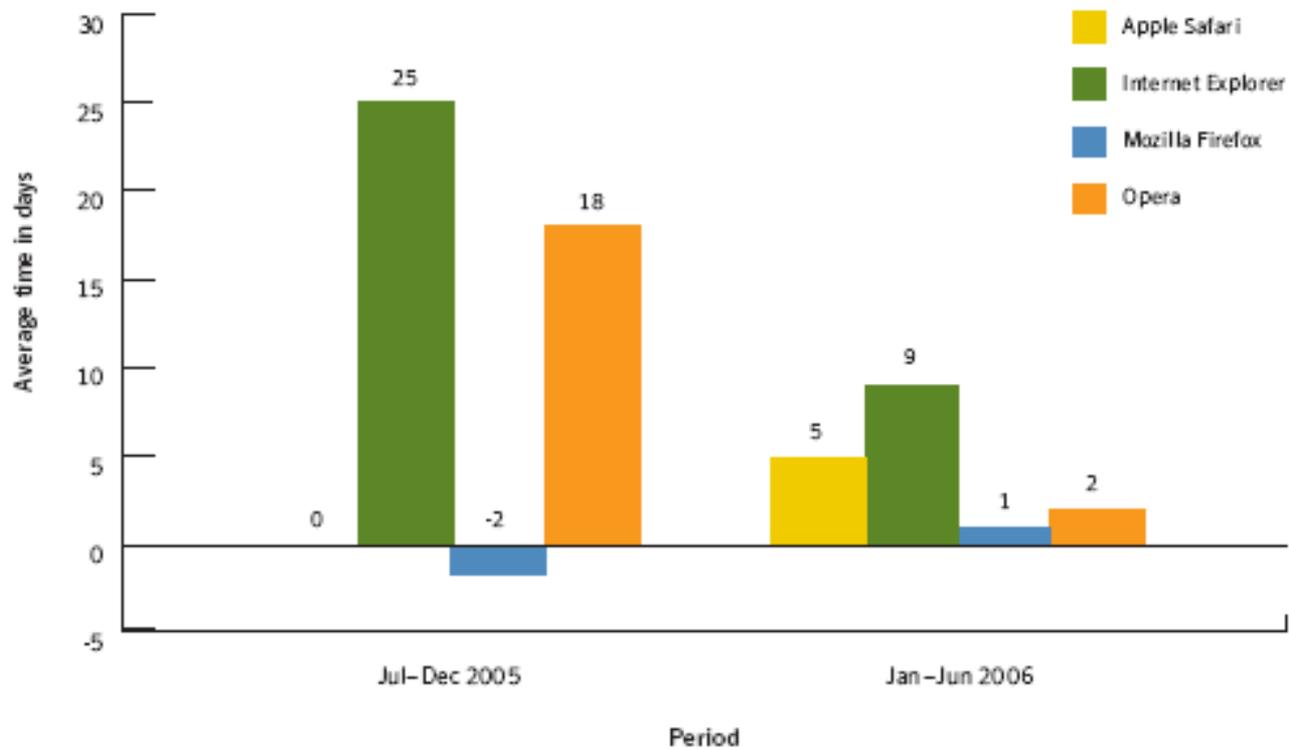


## Pericolosità di una vulnerabilità

---

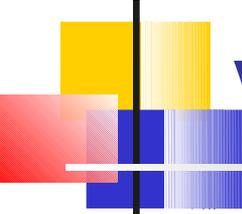
- Il caso migliore si ha quando la patch viene rilasciata prima che sia noto un attacco
- Se, per una vulnerabilità organizzativa, non si applica la patch allora si perde questo vantaggio

# Window of exposure



**Figure 4. Web browsers window of exposure**

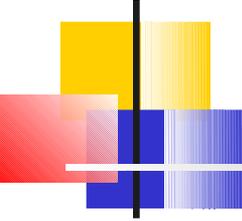
*Source: Symantec Corporation*



# Vulnerabilità e qualità del software

---

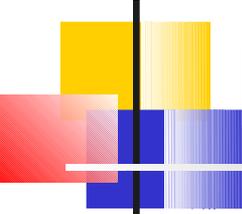
- Il numero di vulnerabilità **individuate** in uno strumento software, sempre minore o uguale a quelle **esistenti**, dipende
  - dalla disponibilità dei sorgenti
  - dalla diffusione dello strumento
  - dalle applicazioni che lo usano
  - dal numero di persone che le cercano
- Di uno strumento poco usato si conoscono poche vulnerabilità, questo **non** vuol dire che le vulnerabilità non esistano



# Diversità genetica

---

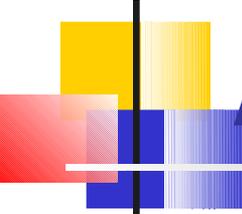
- L'esistenza di una varietà di componenti di produttori diversi aumenta la robustezza del sistema complessivo
- Un monopolio di componenti con vulnerabilità genera una minaccia globale perchè quasi tutti i sistemi informativi ne possono essere affetti
- Configurazione modifica genetica (??!!)



# Defence in depth

---

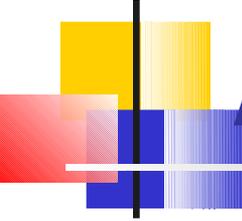
- Ogni componente di un sistema può essere affetto da vulnerabilità
- L'esperto di sicurezza
  - non conosce necessariamente tutte le vulnerabilità
  - sa progettare un sistema in modo che sia sicuro anche quando saranno note vulnerabilità che ora non si conoscono
- Gestione diversa dalla semplice applicazione di patch



# Approccio generale -I

---

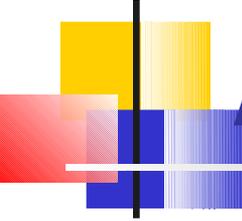
- La soluzione che ottiene la sicurezza eliminando tutte le vulnerabilità (penetrate and patch) ha un costo non ragionevole e non permette di valutare la sicurezza ottenuta
- Occorre poter ottenere sicurezza
  - accettando
  - anticipando(prevedendo)la presenza di alcune vulnerabilità
- Quali vulnerabilità accettare/anticipare è il vero problema da considerare



# Approccio generale - II

---

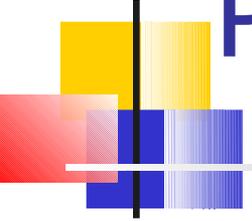
- La valutazione della sicurezza di un sistema richiede l'analisi del rischio
- Rischio
  - perdita media dovuta all'attacco con successo del sistema
  - $R = P_{\text{attsucc}} * \text{Imp}$ 
    - $P_{\text{attsucc}}$  = probabilità di un attacco con successo
    - $\text{Imp}$  = perdita media causata da attacco con successo



# Approccio generale - III

---

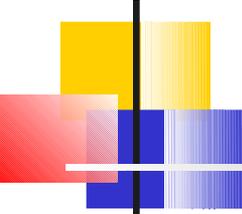
- $P_{\text{attsucc}}$  dipende da
  - Minaccia che può eseguire l'attacco e dalle risorse di cui dispone
  - Complessità assoluta dell'attacco = automatizzabile oppure no
  - Esistono o meno altri attacchi per raggiungere lo stesso obiettivo
  - Complessità relativa



# Probabilità e serie storiche

---

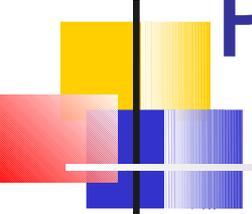
- Per stimare la probabilità si può ricorrere ad informazioni storiche sul sistema ed al numero di casi complessivo e a quello dei casi di interesse
- Ad esempio si calcola il numero di attacchi complessivi ed il numero di attacchi verso un certo componente
- Il rapporto fornisce una stima della probabilità che un attacco interessi un certo componente



# Probabilità e perdita - I

---

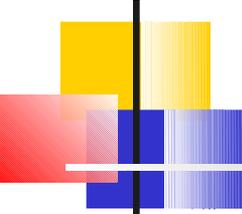
- E' estremamente difficile quantificare la probabilità di successo di un attacco
  - Assenza di informazioni storiche
  - Rapida evoluzione dei sistemi
  - Situazione estremamente diffusa (nel 1700 Laplace si lamentava di mancanza di dati)
- Anche la stima delle perdite e' spesso imprecisa (danni di immagine, perdita di clienti, ...)



# Probabilità senza serie storiche

---

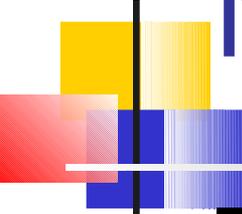
- Model Based Approach
- Si costruire un modello del sistema per simulare attacchi contro il sistema in uno scenario di attacco
- Raccolta di dati e costruzione di un istogramma per approssimare distribuzione di probabilità
- Evaluating risk with no data



## Probabilità - II

---

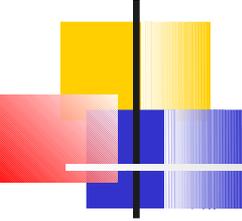
- Spesso sia la probabilità di un attacco con successo che il danno sono stimati in maniera qualitativa  
{bassa, media, alta} oppure  
{bassa, medio-bassa, media ...}
- E' inoltre necessario definire delle regole per il prodotto tra due valori non numerici definendo quelle che si chiamano matrici di rischio



# Matrice di rischio

---

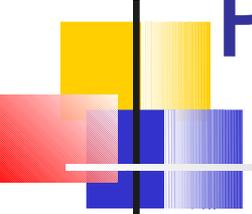
Prob Impatto	Mb	B	M	A	Ma
Ma	A	Ma	Ma	Ma	Ma
A	M	A	A	A	A
M	B	B	M	M	M
B	B	B	B	M	M
Prob Impatto	Mb	B	M	A	Ma



## Probabilità - III

---

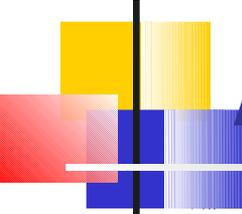
- In assenza di stime storiche come quelle usate ad esempio dalle assicurazioni per le tariffe RCA si ricorre ad un insieme di esperti
- Confronto delle opinioni degli esperti su probabilità' (metodo delphi)



# Problema fondamentale

---

- La stima della probabilità richiede informazioni storiche, quindi sul passato
- Da tali informazioni si estrapolano le possibili evoluzioni future assumendo una legge di corrispondenza rispetto alla distribuzione storica
- Una modifica tecnologica o organizzativa significativa può invalidare il metodo di estrapolazione generando quello che si chiama black swan

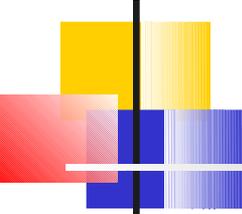


# Analisi del rischio

---

Il modo più moderno di considerare sicurezza ICT

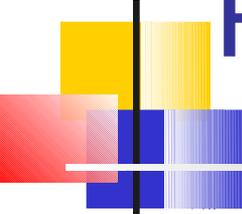
1. Analisi delle risorse da proteggere (asset)
2. Analisi delle vulnerabilità
3. Analisi degli attacchi
4. Analisi degli minacce
5. Analisi delle impatti
6. Individuazione rischio accettabile ed introduzione delle contromisure = gestione del rischio



# Return on investment ROI

---

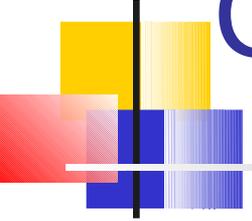
- L'analisi del rischio vuole giustificare i costi della sicurezza = giustificazione economica delle contromisure introdotte
  - É importante prevedere delle contromisure per quegli attacchi che
    - Possono avvenire con probabilità elevata +
    - Hanno un impatto elevato
- = Hanno un rischio elevato rispetto al costo della contromisura da adottare



# Return of investment

---

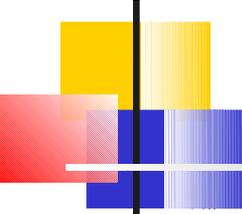
- Formalmente è la differenza tra
  - Il rischio prima della sua gestione
  - Il rischio dopo la sua gestione
- La differenza è dovuta alle contromisure introdotte e che hanno
  - Eliminato vulnerabilità e quindi attacchi
  - Ridotto alcune probabilità aumentando tempo e risorse necessarie per attacco
  - Ridotto il possibile danno



# Guadagno

---

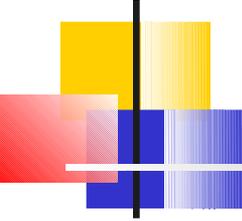
- E' dato dal ROI diminuito del costo delle contromisure
- Ovviamente si deve garantire che questa differenza sia non negativa
- Possiamo anche considerare rapporto = beneficio per ogni unità di spesa ed in questo caso deve essere maggiore di 1



# Passi successivi

---

- Esempi di attacchi
- Analisi delle risorse da proteggere
- Politica di sicurezza
- Analisi delle vulnerabilità
- Analisi delle contromisure
- Analisi degli attacchi
- Metodologie
  - Risk Analysis
  - Risk Assessment



## Passi successivi - II

---

- In linea teorica la politica di sicurezza fa parte delle contromisure
- In realtà comunque spesso è già stata definita al momento dell'analisi ed è utile tenerne conto quando si analizzano le vulnerabilità e gli attacchi