Attacchi/1

Qualche esempio delle tecniche usate per violare i sistemi, e delle più comuni contromisure

Marco Prandini

Un po' di termini

- Minaccia (threat)
  - Un atto ostile intenzionale o meno che ha un qualsiasi effetto negativo sulle risorse o sugli utenti del sistema
- Vulnerabilità (vulnerability)
  - Un difetto nelle misure a protezione del sistema o dei dati
    - Può essere strutturale nell'hardware o software
    - Puo dipendere dalla configurazione
    - Può dipendere da un uso scorretto (es. navigare come root)
- Exploit
  - Uno strumento per trarre vantaggio da una vulnerabilità concretizzando una minaccia
    - Tecnico (cracking)
    - Umano (social engineering)
- Rischio (risk)
  - Il potenziale danno immateriale, perdita economica, o distruzione di risorse che risulterebbe dall'azione di una minaccia che riuscisse a sfruttare una vulnerabilità

Qualche statistica

- Provenienza degli attacchi
  - 81% degli intervistati riporta attacchi esterni
  - 37% degli intervistati riporta attacchi interni
- Tipologia di attacco
  - Più comune: Virus, worm e trojan horses (64% del totale)
  - Più dannosa economicamente: DDoS (50% dei costi totali per incidenti)
- Costo medio: 150k$
  - Fortemente dipendente dal tipo di attacco
- OWASP Top Ten
  - https://www.owasp.org/index.php/Top_10_2013-Top_10
- CWE/SANS Top 25
  - http://cwe.mitre.org/top25/

Vulnerabilità – origini (in sintesi)

- Codice di scarsa qualità
  - "Remotely exploitable buffer overflow vulnerabilities continue to be the number one issue that affects Windows services."
- Reazioni inadeguate dei produttori
  - "In many cases, the vulnerabilities were 0-days i.e. no patch was available at the time the vulnerabilities were publicly disclosed."

Vulnerabilità – origini (in sintesi)

- Attacchi data-driven/cross-platform
  - Cross-site scripting
  - Condivisione e backup di file
  - DNS
  - Database
  - Media players

La fase progettuale

- Non esistono metodi formali che consentano una verifica completa delle caratteristiche di sicurezza di un sistema - purtroppo la pigrizia dei progettisti tipicamente va ben oltre: esempi plateali includono
  - Code injection
  - WEP

La fase progettuale

- Un esempio di code injection: scavalcare un sistema di autenticazione SQL-based, o usarlo per tutt'altro


```
SELECT * FROM Users
WHERE uid='$username'
AND pwd='$password';
```

... e se digito come password:

```
' OR 'a'='a
'; DROP DATABASE WebApp;  --
```

SQL injection cheat sheet

- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/

La fase progettuale

- Un esempio di code injection: cross site scripting (XSS)
  - https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=John
```

```
<?php
        echo 'Welcome to our site ' . stripslashes($_GET['name']);
?>
```

```
http://www.yourdomain.com/welcomedir/welcomepage.php?name=
<script language=javascript>alert('Hey, you are going to be hijacked!');</script>
```

- Esecuzione nel dominio di sicurezza associato al sito vulnerabile
  - può manipolare cookie di autenticazione come già consentito al sito
  - se la pagina vulnerabile è locale può eseguire codice privilegiato

Attacchi locali – Exploit

- Gli attacchi locali sfruttano le vulnerabilità di:
  - Sistema Operativo
  - Hardware sottostante
  - Software in esecuzione locale
- Non sono coinvolti:
  - Protocolli di rete
  - Router e infrastruttura di rete
- Focus on:
  - Architettura Intel IA32
  - Sistema Operativo Linux
- Considerazioni analoghe per altri OS e architetture

Obiettivo Exploit

- Lo scopo di un exploit è far eseguire ad un processo operazioni per cui non era stato pensato
- Tre obiettivi (non mutuamente esclusivi):
  - Fermare il processo (Denial Of Service – DOS)
  - Dirottare il flusso di esecuzione (Esecuzione di codice maligno)
  - Ottenere i privilegi dell'utente root

Preparazione agli Exploit

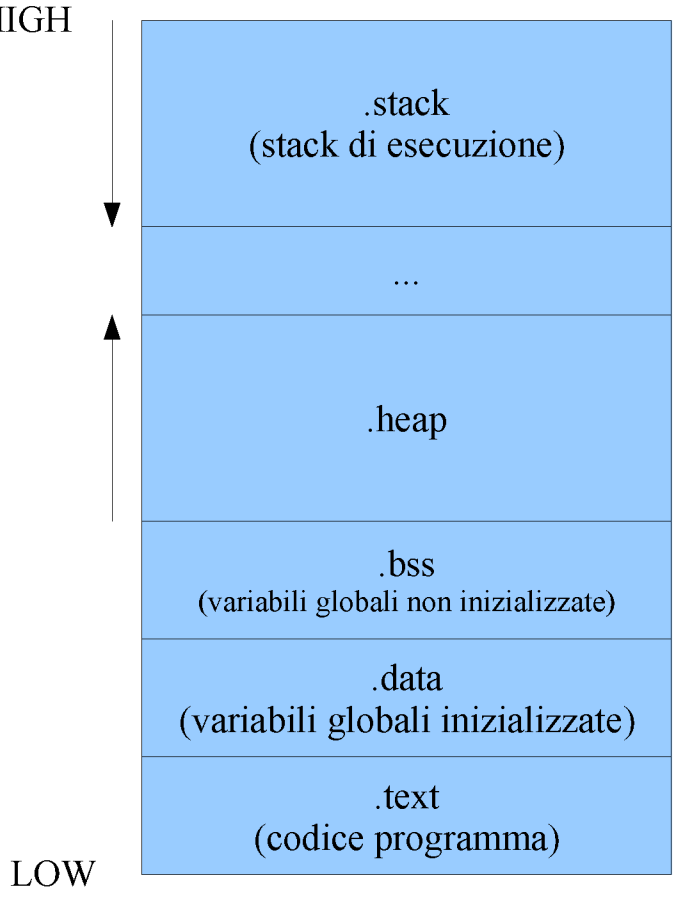
- La comprensione dei meccanismi alla base delle tecniche di Exploit necessita di una conoscenza basilare di:
  - Disposizione in memoria di un processo
  - Funzionamento dell'architettura del processore su cui lavora il sistema vittima (nel nostro caso IA32)
- Si noti che la maggior parte delle vulnerabilità sono relative a codice scritto in C/C++ e linguaggi derivati. Infatti tali linguaggi, più vicini all'hardware, non realizzano controlli (in automatico) sui dati
- Linguaggi di più alto livello (come ADA, ad es.) in fase di compilazione aggiungono controlli ad ogni istruzione/gruppi di istruzioni

Processo in memoria

- Nell'OS Linux lo spazio di indirizzamento di un processo in memoria è suddiviso in un insieme di segmenti:
  - Segmento .text, che contiene il codice eseguibile
  - Segmento .data, contenente i dati inizializzati (variabili statiche inizializzate)
  - Segmento .bss, contenente le variabili non inizializzate
  - Stack d'esecuzione, con i record d'attivazione del processo e le variabili locali
  - Heap, segmento di memoria contenete le variabili dinamiche (può crescere dinamicamente)
  - Altri segmenti necessari al funzionamento (.got, .ctor, .dtor)

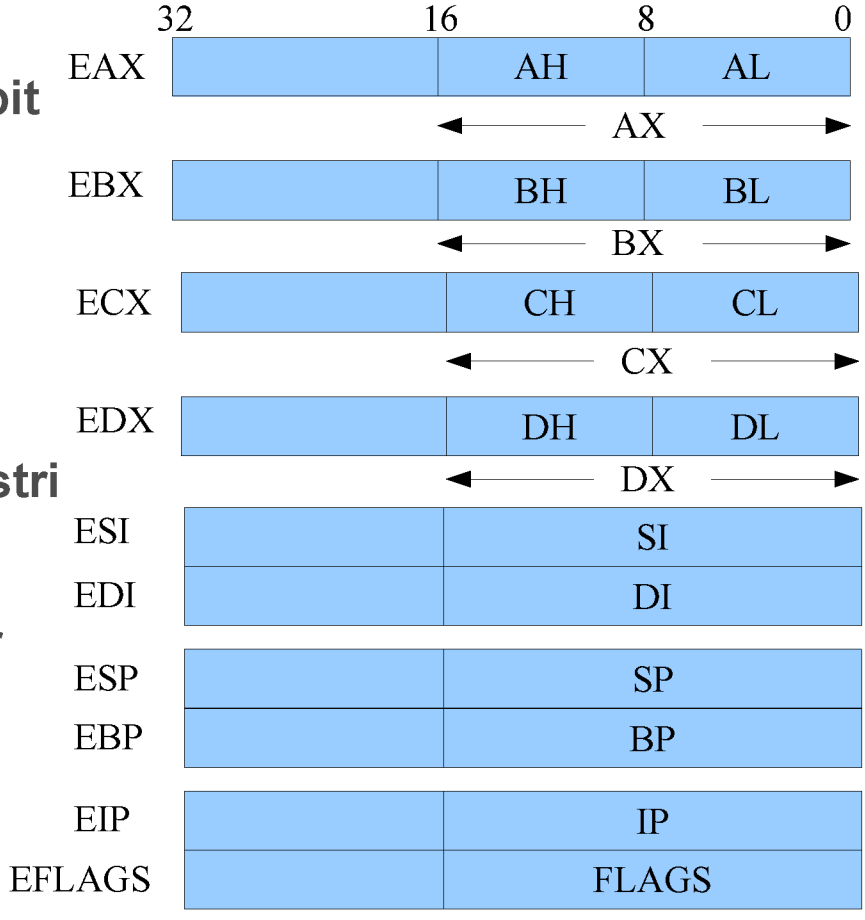
Processo in memoria

- Si noti che il programmatore "vede" gli indirizzi virtuali. Questi ultimi sono tradotti dall'OS (+ HW) in indirizzi fisici:
  - I segmenti non necessariamente sono contigui
- Andamento indirizzi:
  - Lo stack cresce verso il basso
  - L'heap cresce verso l'alto



Cenni architettura IA32

- L'architettura IA32 è dotata di 8 registri 32 bit "general purpose":
  - EAX, EBX, ECX, EDX
  - ESP: stack pointer
  - EBP: base pointer
  - ESI: source
  - EDI: destination
- Inoltre è dotata di registri speciali:
  - EIP: Instruction Ptr
  - EFLAGS: Status register



Convenzioni di chiamata C IA32

- La traduzione C → Assembly adotta convezioni standard (a differenza di altri linguaggi)
- Tre convenzioni di chiamata:
  - __cdecl (Convenzione di Default)
    - Inserimento sullo stack di tutti i parametri attuali di chiamata in ordine inverso rispetto alla signature del metodo.
    - Chiamata tramite “CALL” salvando l'indirizzo di ritorno sullo stack.
    - Il chiamato salva il contenuto del registro EBP ponendolo sullo stack e aggiorna il valore di EBP al contenuto attuale di ESP (in altri termini il nuovo EBP punterà alla locazione dello stack in cui è salvato il vecchio EBP).
    - Il chiamato ritorna il risultato delle proprie computazioni nel registro EAX.
    -

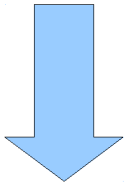
Convenzioni di chiamata C IA32

  - __cdecl (continuo)
    - E' compito del chiamato ripristinare il valore originario di EBP, con quello (da lui) salvato sullo stack in precedenza.
    - E' compito del chiamante rimuovere i parametri attuali dallo stack
  - __stdcall
    - L'unica differenza con la modalità precedente è che è responsabilità del chiamato eliminare i parametri dallo stack.
  - __fastcall
    - La differenza con la __cdecl è che I parametri attuali non sono passati via stack bensì tramite i registri generali (da EAX a EDX e ESI e EDI, quindi con un limite di 6 parametri di 32 bit)

Esempio di chiamata (1/3)

Codice **chiamante** (convenzione __cdecl)

```
…
int result;
…
result = sum(4,5);
…
```



Compilazione

```
0x80401000 result: DW 1
…
0x8040200A PUSH    0x5
0x8040200F PUSH    0x4
0x80402015 CALL    _sum
0x8040201A ADD     ESP, 0x8
0x8040201F MOV     result, EAX
…
```

Il chiamante immette i parametri della funzione in cima allo stack in ordine inverso

Il chiamante invoca la funzione tramite l'operazione CALL, che pone in cima allo stack l'indirizzo di ritorno

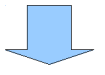
Di ritorno dalla funzione, il chiamante rimuove dallo stack i parametri passati

Il chiamante prende il risultato dal registro EAX

Esempio di chiamata (2/3)

Codice **chiamato** (convenzione __cdecl)

```
int __cdecl sum(int a, int b) {
    int c;
    c = a+b;
    return c;
}
```



Compilazione

```
PUSH    EBP
MOV     EBP, ESP
SUB     ESP, 0X4
MOV     [EBP-4], [EBP+8]
ADD     [EBP-4], [EBP+12]
MOV     EAX, [EBP-4]
MOV     ESP, EBP
POP     EBP
RET
```

Salva il contenuto di EBP, e memorizza in EBP il puntatore al frame pointer (contenuto di ESP)

Riserva sullo stack lo spazio per la variabile locale c

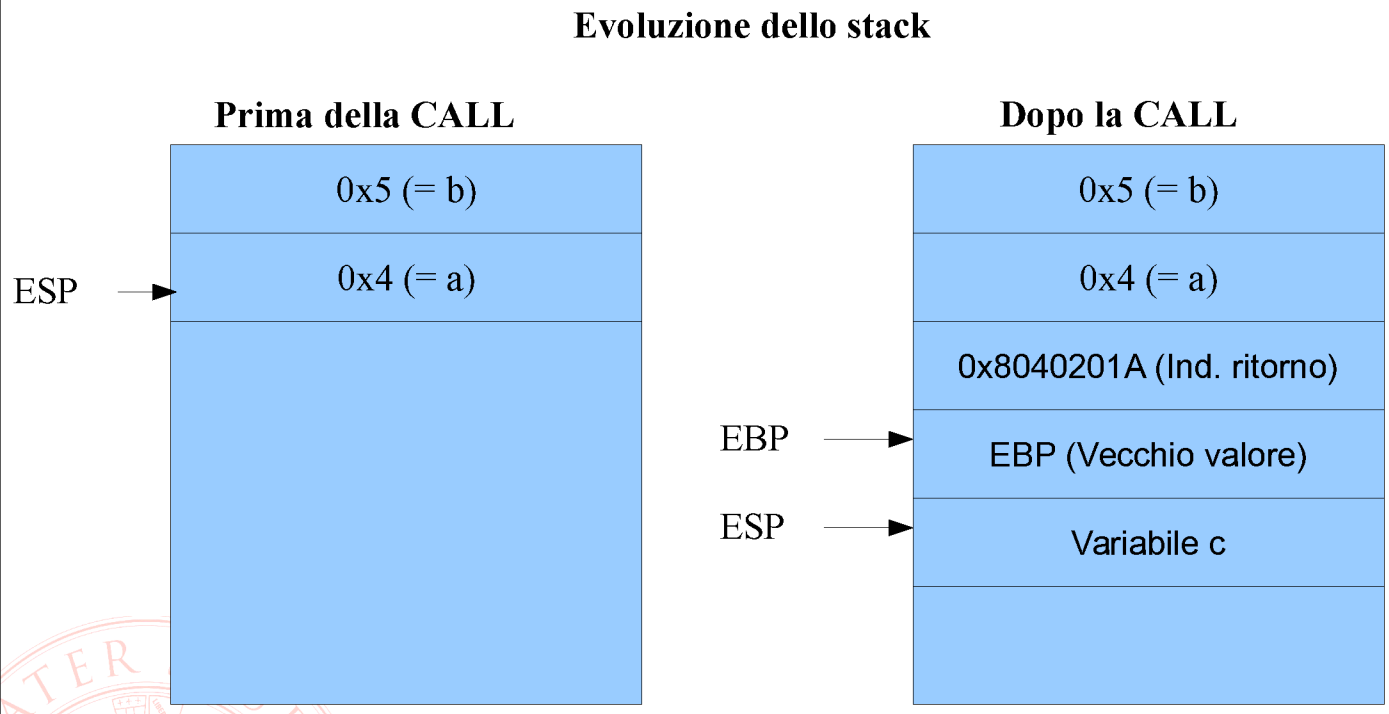
Effettua le operazioni usando come Riferimento (in base a cui muoversi sullo stack) il registro EBP

Salva il risultato in EAX

Ripristino di EBP e ESP

Ritorno al chiamante

Esempio di chiamata (3/3)



Stack Overflow (1/9)

- Prerequisiti per l'attuazione:
  - Presenza di un buffer locale ad una funzione (tale buffer si troverà quindi sullo stack).
  - Possibilità di alimentare il buffer con input esterni (ad es. da tastiera, da file, da rete).
- Modalità di attuazione:
  - L'attaccante riempie il buffer con dati fittizi (padding), sforandone i limiti, fino ad arrivare a porre sullo stack "nella posizione giusta" un nuovo indirizzo di ritorno dalla chiamata (sovrascrivendo quello preesistente).
- Risultato:
  - Il flusso di controllo non procede con il ritorno al chiamante "lecito" bensì al "nuovo" indirizzo di ritorno posto dall'attaccante.

Stack Overflow (2/9)

- Un buffer locale ad una funzione/routine è realizzato con un blocco di memoria riservato sullo stack (di lunghezza finita e predeterminata).
- Il linguaggio C non controlla che i dati con cui alimentare i buffer/array abbiano una lunghezza congrua (tale controllo è a carico del programmatore).
- Si ricordi che:
  - Lo stack cresce con indirizzi decrescenti (cioè i dati "più vecchi" hanno indirizzi più alti)
  - Il riempimento del buffer avviene invece per indirizzi crescenti (sebbene questo si trovi sullo stack)

Stack Overflow (3/9)

- Conseguenza della strategia di riempimento dello stack:
  - In assenza di controlli di overflow, i dati in eccesso (immessi nel buffer) vanno a sovrascrivere i dati dello stack immessi in precedenza.

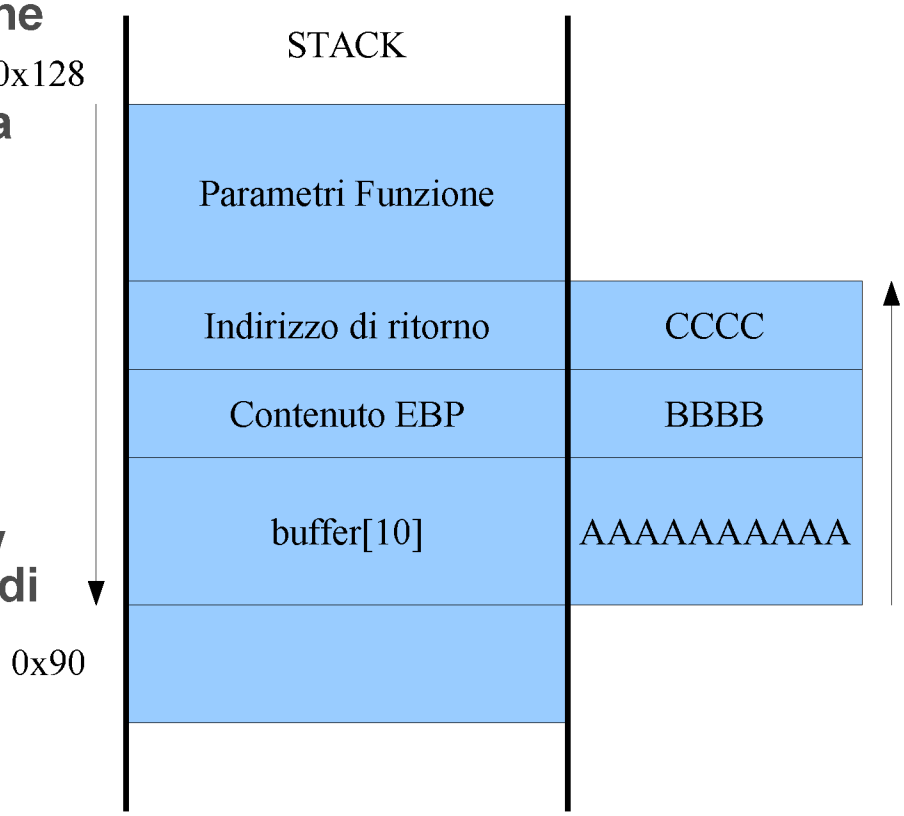
- Esempio

```
void function() {
   char buffer[10];
   gets(buffer);
   ...
}
```

- La funzione gets legge una stringa dallo stdin ma non effettua controlli sulla lunghezza.

Stack Overflow (4/9)

- Esempio di disposizione in memoria.
- L'attaccante scrive una stringa lunga più del dovuto
- Nell'esempio di prima:
  - 10 Byte di padding
  - 4 byte per coprire EBP
  - 4 byte per sovrascrivere l'indirizzo di ritorno
- Si noti come l'overflow sovrascriva l'indirizzo di ritorno.



Stringa: "AAAAAAAAAABBBBCCCC"

Stack Overflow (5/9)

- Conseguenze dell'attacco:
  - Denial Of Service
    - Se l'indirizzo di ritorno "illecito" è relativo ad una posizione in memoria non accessibile (dal programma in esecuzione) avviene un crash per "segmentation fault".
  - Controllo del flusso. L'attaccante prende il controllo dell'esecuzione. Si hanno due alternative:
    - Shell Coding: si "inietta" (come parte della stringa) un pezzo di codice maligno preparato in precedenza. In questo modo, è possibile eseguire codice con i privilegi del processo vittima.
    - Return to LibC: la stringa iniettata per overflow scrive sullo stack i dati necessari per effettuare una invocazione di una funzione di libreria C.

Stack Overflow (6/9)

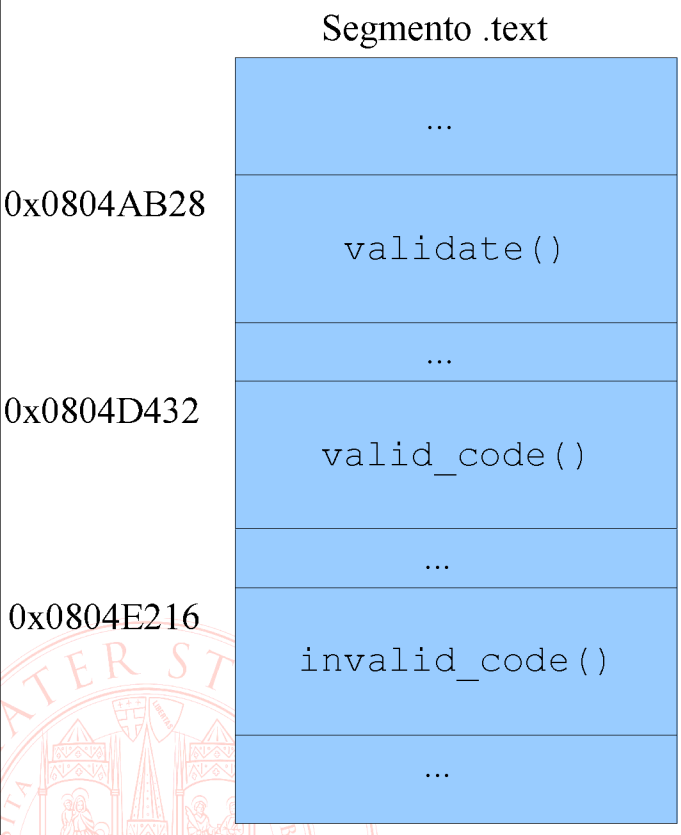
- Un semplice esempio di controllo del flusso
- In questo programma l'input copiato in un buffer determina se un utente ha i diritti di eseguire una porzione di codice o meno (ad es. potrebbe essere richiesta la password da linea di comando)

```
int validate() {
  char buffer[10];
  gets(buffer);
  if(//Valid input)
    return 1;
  else
    return 0;
}

void valid_code() {
  // Codice per utenti autorizzati
  ...
}

void invalid_code() {
  // Codice per utenti non aut.
  ...
}
```

Stack Overflow (7/9)



- L'obiettivo di un attaccante che non ha le credenziali affinché `validate` ritorni 1, è di forzare l'esecuzione del codice di `valid_code`
- `validate` utilizza la funzione `gets` per ottenere la password
- La `gets` è non sicura
- In tali condizioni un attacco di stack overflow è molto semplice

Stack Overflow (8/9)

- Nel momento in cui `validate` richiede l'input all'utente l'attaccante prepara una stringa così composta
  - 10 byte di padding
  - 4 byte per "scavalcare" l'EBP
  - 4 byte per scrivere l'indirizzo di `valid_code`: **0x0804D432**

E' necessario tradurre l'indirizzo di `valid_code` nella sua rappresentazione ASCII. Il risultato comprende anche caratteri non stampabili. Esistono varie tecniche per passare al processo tali caratteri (ad es. printf di bash).

Stringa d'attacco: "AAAAAAAAAABBBB\x32\xD4\x04\x08"

L'indirizzo è fornito con i byte in ordine inverso: l'architettura IA32 è infatti **little endian**

Stack Overflow (9/9)

- L'attacco di Stack Overflow può essere utilizzato **anche su architetture che presentano stack con indirizzi crescenti**

```
void vulnerable() {
    …
    char src[10];
    char dest[10];
    …
    gets(src);
    …
    strcpy(dest, src);
    …
}
```

- **In questo caso, inserendo una stringa più lunga del dovuto in `src`, nel momento in cui verrà richiamata la `strcpy` per copiarla in `dest`, essa andrà a sovrascrivere l'indirizzo di ritorno della `strcpy` stessa e non di `vulnerable`**

Stack Overflow – Canarini (1/3)

- Canarini
  - Prima forma di contromisura
  - Deve il suo nome alla similitudine storica con i canarini dei minatori in grado di rilevare fughe di gas
- Il concetto alla base è di porre sullo stack, prima di un buffer, un dato di riferimento.
  - Il processo è in grado di rilevare un tentativo di attacco verificando l'integrità di tale dato.
  - In caso di attacco con overflow il dato viene sovrascritto e la verifica da esito negativo
  - Tale protezione è realizzata tramite la collaborazione congiunta di compilatore e libreria standard
  - Non è un meccanismo fornito dall'OS o dall'HW
  - Generazione del dato e verifica causano overhead

Stack Overflow – Canarini (2/3)

- Funzionamento in GCC
  - Tale protezione era inizialmente fornita come patch ProPolice (a partire dalla versione 3.x)
  - Inclusa nel main branch a partire dalla versione 4.1
- Abilitazione:
  - Non è abilitato di default su tutti gli OS / Distro Linux
  - `-fstack-protector` abilita solo per buffer di stringhe
  - `-fstack-protector-all` abilita per tutti i tipi di buffer
  - `--param ssp-buffer-size=` imposta una soglia di dimensione del buffer oltre la quale la protezione viene attivata. Questo evita che la protezione venga attivata per tutte le chiamate a funzione, riducendo l'overhead.

Stack Overflow – Canarini (3/3)

- Ad ogni chiamata di funzione:
  - Si genera e scrive un valore casuale di 4 byte
- L'attaccante dovrebbe sovrascrivere anche il canarino per modificare l'indirizzo di ritorno (non può "scavalcarlo")
- La modifica del canarino viene rilevata nel momento in cui si ritorna dalla chiamata
- L'azione di default in tal caso è la terminazione del processo
- Tale evento può essere inoltre catturato tramite segnali dell'OS

Shellcoding (1/5)

- Non è una forma di attacco alternativa, bensì è un tecnica per sfruttare lo stack overflow.
- Consiste nell'iniettare (tramite stack overflow):
  - Codice maligno
  - Indirizzo di ritorno che punti al codice di cui sopra
- L'obiettivo tipico di tale approccio è l'apertura di una shell (da cui il nome), con i privilegi del processo attaccato (tipicamente root o con il bit setuid attivato).
- Il principio alla base di tale tecnica è utilizzabile anche con altri tipi di attacco alternativi allo stack overflow.

Shellcoding (2/5)

- Un esempio di codice maligno che realizza l'invocazione della system call exit con il valore 0, terminando il processo è il seguente:

```
mov EBX, 0
mov EAX, 1
int 0x80
```

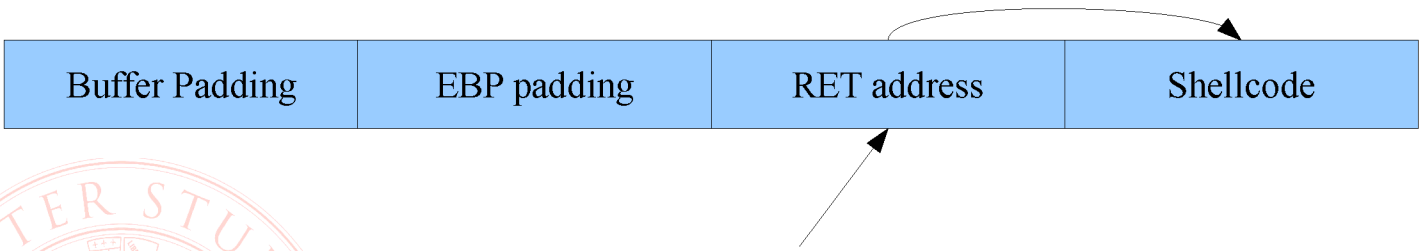
I parametri di una system call devono essere caricati nei registri in ordine EBX, ECX, EDX, ESI, EDI. Nel nostro caso l'unico parametro è il valore 0.

Il registro EAX va predisposto con l'identificativo numerico della system call. nel nostro caso exit=1

Si genera un interrupt sulla linea 0x80
(che corrisponde al gestore delle system call in Linux)

Shellcoding (3/5)

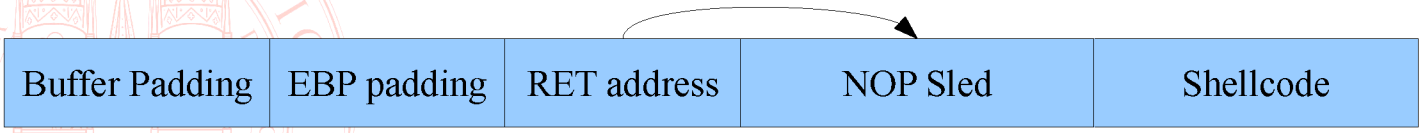
- Dopo aver preparato il codice assembly da iniettare si procede come segue:
  - Si compila il codice di cui sopra.
  - Si genera la rappresentazione esadecimale del risultato compilato.
  - Si genera la stringa da iniettare come visto per lo stack overflow:



- Il problema che si presenta è come stabilire l'indirizzo di ritorno (ovvero l'indirizzo in cui andrà a posizionarsi il codice maligno iniettato).

Shellcoding (4/5)

- Per calcolare/intuire l'indirizzo di ritorno esistono varie tecniche:
  - Tipicamente il segmento .stack di un processo comincia sempre a partire dallo stesso indirizzo
  - L'attaccante può procurarsi il codice sorgente del programma da attaccare e la sua immagine binaria (facile se la vittima usa una qualsiasi distro Linux)
- Per avere maggiori garanzie sulla riuscita si può ricorrere ad un "NOP Sled" (slitta di NOP):
  - Il NOP Sled è una sequenza di NOP (operazione assembly che non effettua nulla) che precede lo Shellcode.
  - E' sufficiente che il RET Address cada in un punto qualsiasi del NOP Sled:



Shellcoding (5/5)

- Non sempre un codice maligno è iniettabile
  - Tornando all'esempio di prima:

```
mov EBX, 0          BB 00 00 00 00
mov EAX, 1    ->    B8 01 00 00 00
int 0x80            CD 80
```

  - Nel risultato esadecimale c'è una serie di byte uguali a 0
  - Il primo di questi verrà interpretato dalle routine di lettura stringhe come **terminatore di stringa**
  - Uno shellcode, per poter essere iniettabile, deve essere sottoposto ad un'operazione di **Zeros Cut-off**
  - Nell'esempio precedente si può agire in questo modo:

NB: AL è l'insieme di 8 bit meno significativi Del registro EAX

```
xor EBX, EBX        31 DB
mov AL, 1     ->    B0 01
int 0x80            CD 80
```

Shellcoding / ASLR

- Una prima contromisura contro l'iniezione di codice maligno è l'**ASLR: Address Space Layout Randomization**
  - E' una protezione offerta dall'OS
  - Si tratta di rendere casuale l'indirizzo di partenza dei segmenti che compongono un processo
  - Questa randomizzazione interessa solo gli indirizzi virtuali di un processo non la sua disposizione in RAM
  - In questo modo l'attaccante non ha modo di trovare un plausibile indirizzo assoluto di ritorno a cui puntare
- Per Linux tale feature è offerta dalla patch grsecurity/PAX che **solo in alcune distribuzioni è inserita nel kernel**

Shellcoding / NX Stacks

- Una seconda contromisura alla possibilità di eseguire codice maligno dallo stack è fornita dagli **Stack Non Eseguibili**
- E' una feature **fornita dall'HW ma che deve essere supportata dall'OS**
- Alcune architetture permettono di impostare un flag associato a pagine di memoria
  - Tale flag, che deve essere impostato dall'OS, indica se la pagina contiene codice che può essere eseguito o meno
  - Intel commercializza questa feature con il nome di XD bit (eXecution Disable bit) ed è stata introdotta solo a partire dai processori a 64 bit
  - Esistono alcuni OS (Solaris) che implementano tale feature in software, ma ciò causa un leggero overhead

Shellcoding / NX Stacks

- Tale feature **non è presente sui processori INTEL/AMD a 32 bit**
- **Linux supporta gli stack non eseguibili a partire dal kernel 2.6.8 per architettura x86_64**
  - **Introdotta con la patch PAX/grsecurity**
  - **La versione x86 a 32 bit può sfruttare tale feature solo se in esecuzione su un processore x86_64**
- Un'evoluzione di tale tecnologia è la **W^X**
  - **Piuttosto che marcare una pagina come non eseguibile, ogni pagina viene marcata come Eseguibile (eXecutable) o Scrivibile (Writable) ma non entrambe**
  - **Offre una sicurezza maggiore rispetto al bit XD, che comunque permette di eseguire codice da pagine scrivibili**
  - **In Linux è implementato parzialmente da ExecShield**

Altre possibilità di attacco

- **RET2LIBC / RET2SYSCALL**
  - **Tramite Stack Overflow si dirotta il flusso di esecuzione, piuttosto che verso codice sullo stack protetto da NX, verso una (o più) funzioni della onnipresente libreria C, oppure verso una system-call del s. o.**
- **Format Strings**
  - **Tramite la stringa di formato passata a printf si inietta codice e si forza il salto che lo esegue**
- **Heap Overflow**
  - **Si sfruttano vulnerabilità specifiche dei metadati inseriti dalle librerie C per l'allocazione dinamica di memoria**
- **Return Oriented Programming**
  - **Si assembla il codice da eseguire come sequenza di "gadget" (brevi sequenze di linguaggio macchina prese dai binari già in memoria)**

Rootkit (1/2)

- Gli attacchi mostrati in precedenza consentono ad un intruso di controllare un sistema
  - Tale controllo può avvenire ad es. con una shell con i diritti del proprietario del processo attaccato
- Il problema dell'attaccante è nascondere la propria presenza
- Per Rootkit si intende proprio un sistema software in grado di nascondere la compromissione di un sistema

Rootkit (2/2)

- Tipologie di rootkit
  - Application level
    - Trojan, sostituzione di utility di sistema o demoni con versioni contenenti backdoor
  - Kenel level
    - Moduli kernel o driver maligni che consentono di controllare (anche a livello hardware tramite i device driver) la macchina attaccata o di sostituire le system call del sistema operativo (ad esempio per nascondere tra i file presenti in una directory quelli maligni o tra i processi in esecuzione i trojan installati)
  - Library Level
    - Sostituzione di librerie con una propria versione

Rootkit – rilevazione

- I rootkit hanno l'obiettivo di mascherare l'attacco
  - La rilevazione della presenza del rootkit è un'operazione difficile poiché deve far uso degli stessi strumenti di diagnostica che il rootkit ha alterato (questo perchè tipicamente la ricerca dei rootkit non dovrebbe comportare lo spegnimento e l'eventuale reinstallazione dell'OS).
  - Esistono varie tecniche di rilevazione
    - Ricerca euristica: tipica degli antivirus, consiste nella ricerca di pattern binari ricorrenti nei processi maligni
    - Verifica della memoria di massa del sistema effettuando il boot da un OS pulito (ad es. tramite una distribuzione live). Tale soluzione è indicata per rootkit di livello kernel

Difese

- Prevenzione
- Rilevazione

Un po' di sigle

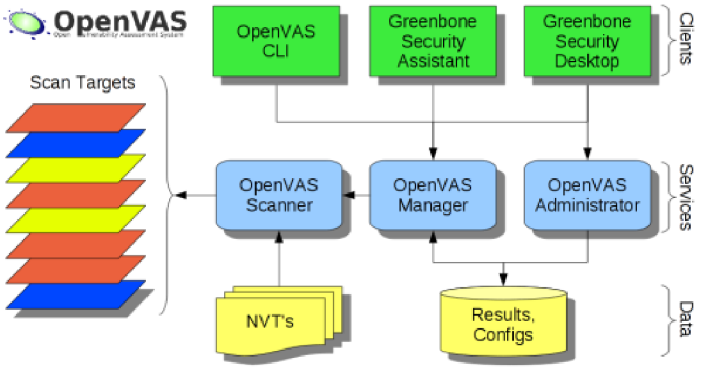
- IDS = Intrusion Detection System
  - è genericamente un sistema in grado di rilevare tentativi di attacco
    - *signature based* (riconosce attacchi noti)
    - *anomaly detection* (riconosce deviazioni dall'uso standard)
  - a seconda di cosa controlla e dove è collocato:
    - *NIDS = Network IDS*
    - *HIDS = Host IDS*
- SIEM = Security Information and Event Management
  - un nome un po' commerciale per racchiudere strumenti, politiche e procedure per la gestione integrata delle fonti di informazione e degli incidenti

Conoscere i propri punti deboli

- Le vulnerabilità vengono pubblicate secondo il principio della *responsible disclosure*
  - su varie mailing list (es. bugtraq, full disclosure, si veda http://seclists.org/)
  - su archivi e database organizzati
    - Common Vulnerabilities and Exposures http://cve.mitre.org/
    - National Vulnerability Database http://nvd.nist.gov/
    - Open Sourced Vulnerability Database http://osvdb.org/
    - SecurityFocus http://www.securityfocus.com/vulnerabilities
    - US-CERT http://www.kb.cert.org/vuls/
- I Computer Emergency Readiness Team
  - coordinano lo scambio di informazioni critiche per la sicurezza
  - stabiliscono programmi e procedure di reazione agli attacchi
  - gestiti a livello nazionale, coordinati da FIRST http://www.first.org/

Tenere sotto controllo i sistemi

- Una pratica consigliata è “attaccare” i propri sistemi con gli stessi test di vulnerabilità documentati nei db citati
- Ad esempio: Open Vulnerability Assessment System (OpenVAS) - http://www.openvas.org/



OpenVAS – Caratteristiche principali

- architettura distribuita che separa
  - il motore di scansione (applica i test ai target)
  - il manager (coordina i task di scansione)
  - l'interfaccia (pianifica i task sul manager e mostra i risultati raccolti)
    - diverse interfacce (cli, grafica, web) usano lo stesso protocollo verso il manager
  - l'amministrazione (gestisce gli utenti e i database)
- database di vulnerabilità
  - Network Vulnerability Tests
    - descrizione della vulnerabilità
    - piattaforme colpite
    - procedura di verifica
  - aggiornato quotidianamente!

Host IDS

- La rilevazione di intrusioni sull'host è tipicamente svolta per mezzo di un *integrity checker*
- Principio:
  - Si memorizza in un database lo stato del filesystem quando è certamente "pulito"
  - Si confronta periodicamente il filesystem col database
- Tra i più diffusi:
  - Tripwire (commerciale)
  - AIDE (fork FOSS di Tripwire)
  - AFICK

https://www.sans.org/reading-room/whitepapers/detection/ids-file-integrity-checking-35327

Integrity checkers

- Caratteristiche da valutare:
  - Algoritmi usati per calcolare le impronte dei file
  - Performance e dimensioni del DB
  - Capacità di proteggere i propri stessi binari
  - Capacità di proteggere il database
  - Portabilità
  - Complessità degli aggiornamenti
    - Del sw
    - Del database

AIDE

- Un esempio di configurazione

```
MyRule = p+i+n+u+g+s+b+m+c+md5+sha1
/etc p+i+u+g     # check only permissions, inode, user and group for etc
/bin MyRule      # apply the custom rule to the files in bin
/sbin MyRule     # apply the same custom rule to the files in sbin
!/var/log/.*     # ignore the log dir it changes too often
```

- Caratteristiche
  - Compilato, molto veloce
  - Integrabile con permessi estesi (acl, selinux)

http://aide.sourceforge.net/

http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.aide.html

AFICK

- Configurazione molto simile ad AIDE ma con un numero minore di check supportati
- Caratteristiche
  - Può essere eseguito da media ottici read only per garantire la sua stessa integrità
  - Report CSV semplici e facili da importare in altri sw
  - Scritto in Perl, molto portabile

http://afick.sourceforge.net/

Log di sistema

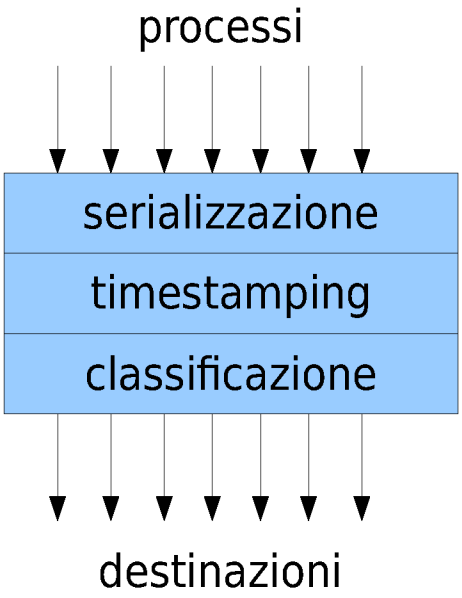
- I log (diari) tenuti dal sistema sono indispensabili per la diagnostica in generale, e in particolare per rilevare attività malevole o sospette
- La loro stessa sicurezza va garantita!
  - Usare appropriatamente un integrity checker
  - Replicarli su macchine remote
- Logging su server remoto
  - Vantaggio aggiuntivo: centralizzazione
  - Implementazioni avanzate: shadow loggers
  - Problema: diventa un bersaglio appetibile
    - DoS

Linux logging

- Soluzioni comuni
    - Tipicamente producono file di testo
    - Nessuna garanzia di uniformità di formato a parte la marcatura temporale
  - BSD syslog (obsoleto)
    - klogd
  - Rsyslog
  - Syslog-ng
- In prospettiva integrato in systemd
  - *Journal*
  - Attivo dal boot, non dipende dall'avvio di altri servizi
  - Formato binario, visualizzabile con `journalctl`

syslog

- Principi di base mantenuti anche dalle evoluzioni



syslog: selettori e destinazioni

- Ogni messaggio è etichettato con una coppia **facility.priority**
  - Facility = argomento
    - `auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0..local7`
  - Priority = importanza in ordine decrescente:
    - `emerg, alert, crit, err, warning, notice, info, debug`
- Le destinazioni possibili sono
  - File: identificato da path assoluto
  - STDIN di un processo: identificato da una pipe verso il programma da lanciare
  - Utenti collegati: username, o * per tutti
  - Server syslog remoto: @indirizzo o @nome
    - La comunicazione avviene su UDP, porta 514

syslog: selettori

- `/etc/syslog.conf` contiene le regole di smistamento dei messaggi
- Ogni riga = una regola
  - [etichetta di interesse] [destinazione]
  - Parsate tutte, quindi un messaggio può finire su più destinazioni
- Trattamento delle priority
  - Soglia: una regola che specifica una priority fa match con tutti i messaggi di tale priority e superiori a meno che non sia preceduta da "="
  - Priority speciale *none*: serve per ignorare i messaggi con la facility specificata prima del punto
- Es:

```
kern.*                  /dev/console
*.info;mail.none;       /var/log/messages
*.emerg                 *
kern.crit               "|/usr/bin/alerter"
*.=warning              @loghost
```

rsyslog

- Struttura modulare per caricare solo le funzioni necessarie
  - es. attivazione della ricezione di messaggi via rete:

```
$ModLoad imudp
$UDPServerAddress 1.2.3.4
$UDPServerRun 514
```

- File di configurazione modulare
  - Direttive globali in `/etc/rsyslog.conf`
  - Direttive specifiche in file separati sotto `/etc/rsyslog.d`
- Scarto di messaggi (per evitare che vengano catturati da troppi selettori)
  - Basta mettere `~` come destinazione

rsyslog – modalità di output evolute

- Template per definire canali di output
  - Possono sostituire le destinazioni in modo più flessibile
  - Es:

```
$template apacheAccess,"/var/log/external/%fromhost%/apache/
%msg:R,ERE,1,ZERO:imp:([a-zA-Z0-9\-]+)\.--end%-access.log"
local6.notice ?apacheAccess
```

Segnaposto che verrà sostituito per mezzo di una elaborazione del messaggio fatta via regex

Segnaposto che verrà sostituito dal nome dell'host che origina il messaggio

- TCP logging
  - Per evitare perdita di messaggi (finché non ci sono crash!)

http://blog.gerhards.net/2008/04/on-unreliability-of-plain-tcp-syslog.html

```
• *.*  @@indirizzo
```

- Shell execute
  - Passa il messaggio come parametro a un programma

```
• *.*  ^programma;template
```

rsyslog – selettori evoluti

- Rsyslog seleziona i messaggi in tre modi
  - i tradizionali facility.priority
  - Filtri basati su proprietà
  - Filtri basati su espressioni
- Filtri basati su proprietà
  - `:property, [!]compare-operation, "value"`
  - Es: `:msg, !contains, "error"  /var/log/good.log`
- Filtri basati su espressioni
  - Ancora in evoluzione!
  - `if expr then destinazione`
  - Diventeranno un sistema completo di scripting, che consentirà di eseguire programmi arbitrari per determinare la destinazione

rsyslog – moduli

- Troppi per citarli esaustivamente
  http://www.rsyslog.com/doc/v8-stable/
- Tra i più interessanti:
  - RELP logging - per garanzia totale di consegna

```
$ModLoad omrelp
*.*  :omrelp:indirizzo
```

  - Output su tabelle di database

```
$ModLoad ommysql
```

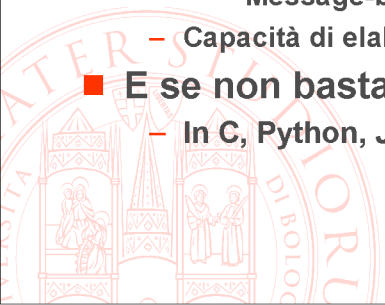
  - Acquisizione diretta dei messaggi del kernel (sostituisce klogd)

```
$ModLoad imklog
```

syslog-ng

https://syslog-ng.org/

- Flessibile
  - Input compatibile con
    - Formati standard syslog (RFC3164, RFC5424)
    - JSON
    - Journald (systemd)
  - Output verso molteplici destinazioni
    - Tutti i DB SQL più diffusi
    - DB NOSQL (es. MongoDB)
    - Cloud databases (es. Redis)
  - Varietà di protocolli
    - Client-server
    - Message-based (AMQP, STOMP)
  - Capacità di elaborazione del contenuto dei messaggi
- E se non basta, estendibile con plugin
  - In C, Python, Java, Lua, o Perl



syslog-ng

- Configurazione:
  - Definizione di una *source*, che può unificare più ingressi fisici

```
source s_two {
 network(ip(10.1.2.3) port(1999));
 network(ip(10.1.2.3) port(1999) transport("udp"));
};
```

  - Definizione di una *destination*, con relative opzioni

```
destination d_file {
 file("/var/log/${YEAR}.${MONTH}.${DAY}/messages"
 template("${HOUR}:${MIN}:${SEC} ${TZ} ${HOST} [${LEVEL}] ${MSG}\n")
 template-escape(no));
};
```

  - Attivazione di un canale di log

```
log { source(s_two); destination(d_file); };
```

Sistemi di analisi dei log

	Logwatch	Swatch	Splunk
Regular expression support	yes	yes	no
Real-time monitoring	no	yes	yes
Support for multiple log files	yes	no	yes
Good preconfiguration	yes	no	yes
Modular configuration	yes	no	-
Reactive	no	yes	no
Interactive	no	no	yes

Logwatch (System log analyzer and reporter):
http://www.logwatch.org/

Swatch (Simple WATCHer of Logfiles):
http://swatch.sourceforge.net/

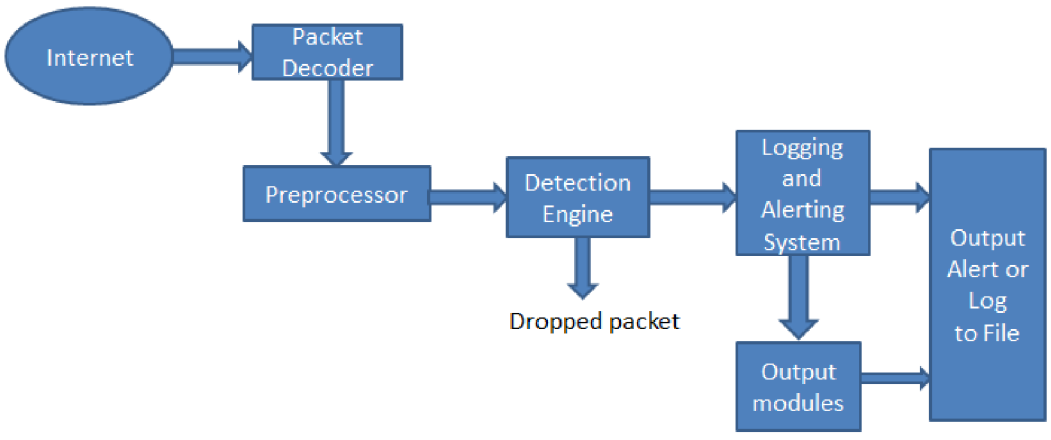
Splunk (The search engine for IT data):
http://www.splunk.com/

Network IDS

- La rilevazione di attacchi che giungono via rete viene svolta analizzando il traffico in entrata/uscita
- Problema essenziale:
  - esaminare tutto il traffico senza rallentarlo
  - generando pochissimi falsi allarmi
  - senza lasciar sfuggire attacchi reali
- Due approcci:
  - Signature based: rileva flussi con caratteristiche notoriamente malevole
  - Anomaly based: rileva flussi che si discostano dalla "normalità"
- Tra i più diffusi
  - Snort
  - Suricata
  - Bro

SNORT

- http://www.snort.org/



SNORT

- Architettura di base:
  - Libpcap-based sniffing interface
    - cattura i pacchetti e li salva in un formato standard per l'analisi successiva
  - Rules-based detection engine
    - possibilità di utilizzare un vasto set di regole già pronte e di personalizzarle
  - Plug-in system
    - funzionamento estendibile aggiungendo moduli
- CAVEAT: Snort è uno strumento potente, ma per massimizzare la sua efficacia serve una competenza specifica ed un lungo affinamento della configurazione
  - stima dell'autore originale: 12 mesi di formazione per acquisire i fondamenti di intrusion detection, 24-36 mesi per diventare esperti

SNORT – Detection Engine

- Le regole definiscono le “signature” di un attacco, cioè l'insieme di caratteristiche per riconoscerlo
- Possono essere formate combinando più elementi semplici
- Possono riconoscere una molteplicità di scenari
  - Stealth scans, OS fingerprinting, buffer overflows, back doors, CGI exploits, ecc.
- Il sistema è molto flessibile e la creazione di nuove regole è relativamente semplice

```
alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR
subseven 22"; flags: A+; content: "|0d0a5b52504c5d3030320d0a|";
reference:arachnids,485; reference:url,www.hackfix.org/subseven/;
sid:103; classtype:misc-activity; rev:4;)
```

SNORT – Plug-Ins

La struttura di base permette di attivare diversi moduli per le tre fasi principali

- Preprocessor
  - esamina e manipola i pacchetti prima di passarli al detection engine (evitando ad esempio la scansione di cose ovviamente innocue)
- Detection
  - ogni modulo implementa un singolo test semplice su di un singolo aspetto o parte di un pacchetto
  - può anche essere saltata se si vuole usare Snort solo per salvare traffico interessante da processare successivamente, fuori linea
- Output
  - Riporta i risultati degli altri plug-in (quindi consente la personalizzazione delle destinazioni e dei formati dei messaggi diagnostici)

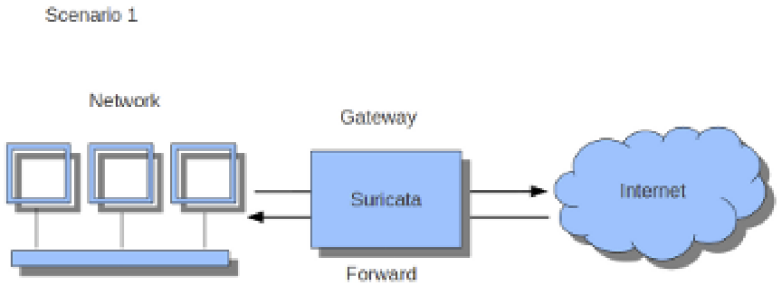
Suricata

https://suricata-ids.org/

- Configurazione:
  - Implementa un linguaggio per la rilevazione di signature
    - Compatibile con SNORT
  - Può essere configurato per rilevare anomalie
  - Può essere esteso con LUA per processing oltre la capacità del linguaggio a regole
- Caratteristiche di funzionamento particolari:
  - Riconosce automaticamente il tipo di traffico e adatta il dettaglio dei log
    - es. salva i certificati X.509 usati nelle connessioni TLS, salva l'header a livello applicazione per i protocolli più comuni
- Output facilmente integrabile con molti strumenti di analisi e visualizzazione

Suricata

- Suricata può agire da IPS (Intrusion Prevention System)
  - Se interposto tra due reti, può non inoltrare il traffico malevolo



Bro

https://www.bro.org/

- Un framework per definire piattaforme di analisi del traffico
- Scalabile
  - Sistema distribuito
  - Nodi replicati con load balancing
- Particolarmente usato in ambito accademico