Sicurezza delle reti[1]

Mattia Monga

Dip. di Informatica e Comunicazione
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2010/11

Lezione XV: IDS e attacchi imprevisti

Attacchi imprevisti

I NIDS signature-based si basano sull'assunzione di saper caratterizzare un attacco. In generale ciò significa saper fare una delle seguenti due cose:

1. Identificare una vulnerabilità: la firma cercherà di rappresentare tutti gli attacchi capaci di sollecitarla;
2. Riconoscere un exploit: la firma cercherà di rappresentare tutte le varianti.

Un attacco può essere del tutto inatteso: in questo caso si parla di zero-day, ossia il giorno *prima* di quando i NIDS sono in grado di riconoscerlo.

Finestra di vulnerabilità

Il tempo che intercorre fra il momento in cui un attaccante si rende conto di una vulnerabilità e capisce come sfruttarla e il momento in cui l'attacco è identificato dal difensore può essere molto lungo (*vulnerability window*).
Nel 2008 Microsoft ha reso nota una vulnerabilità di IE presente dal 2001, quindi con una finestra potenzialmente di 7 anni!

Ricerca delle vulnerabilità

La ricerca delle vulnerabilità non note è una delle attività dei "laboratori di sicurezza".

- Si cercano vulnerabilità generiche (non di una rete specifica): si analizzano applicazioni e protocolli
- Gli *zero-day* hanno un mercato (non solo underground!)

Tecniche per la ricerca

Le tecniche per identificare le vulnerabilità sono riconducibili a tre grandi categorie

studio analitico si studiano le specifiche piú o meno formali di applicazioni e protocolli

fuzzing si provano le applicazioni (o i protocolli) con input "strani", scelti in modo casuale.

honeypot un sistema che viene realizzato e messo in opera con il solo fine di essere bersaglio di attacchi. Lo scopo principale e quello di raccogliere piú informazioni possibili circa le tecniche e le modalità utilizzate dagli attaccanti.

Polimorfismo degli attacchi

La medesima vulnerabilità può essere sfruttata da exploit con forme diverse: gli attacchi hanno quindi natura polimorfica.
In generale è impossibile prevedere tutte le possibili varianti e costruire le firme che permettano di rilevarli.
Una forma completamente nuova è analoga ad uno zero-day, anche se la vulnerabilità è già nota.

Cifratura

Una delle tecniche piú diffuse è la cifratura.

- Viene generata per ogni attacco una chiave casuale
- il *payload* dell'attacco viene cifrato, apparendo cosí sempre diverso
- l'unica parte di codice costante è una piccola routine di decifratura (possono bastare 3-4 istruzioni: p.es. cifratura XOR)
- anche la routine di decifratura può essere variata con ulteriori tecniche di polimorfismo

Dead-code insertion

La dead-code insertion o trash insertion consiste nell'aggiungere codice ad un programma senza modificarne il comportamento.

- La tecnica piú semplice è inserire `nop`
- Metodi piú sofisticati fanno uso di sequenze di codice che si annullano vicendevolmente

Diventa impossibile rilevare l'attacco con la ricerca di stringhe costanti.

```
call 0h
pop ebx
lea ecx, [ebx + 45h]
nop
nop
push ecx
push eax
inc eax
push eax
dec [esp - 0h]
dec eax
sidt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```


Code transposition

Code transposition sposta le istruzioni in modo che l'ordine del codice binario sia differente dall'ordine di esecuzione o dall'ordine delle istruzioni considerato nella stesura della signature.

- riordinando casualmente blocchi di istruzioni, per poi procedere a ristabilire l'ordine di esecuzione corretto inserendo salti incondizionati (facile da fare automaticamente)
- mischiando istruzioni indipendenti (richiede analisi sofisticate del codice)

```
call 0h
pop ebx
jmp Step2
Step3: push eax
push eax
sidt [esp - 02h]
jmp Step4
add ebx, 1Ch
jmp Step6
Step2: lea ecx, [ebx + 45h]
push ecx
jmp Step3
Step4: pop ebx
cli
jmp Step5
Step5: mov ebp, [ebx]
```


Instruction substitution e register reassignment

- instruction substitution prevede l'utilizzo di un dizionario di sequenze di istruzioni equivalenti, sequenze che possono essere sostituite tra loro quando contenute nel corpo dei programmi.
- register reassignment sostituisce l'uso di un registro con un altro equivalente allinterno di una sequenza limitata di istruzioni.

```
call 0h
pop ebx
lea ecx, [ebx + 42h]
sub esp, 03h
sidt [esp - 02h]
add [esp], 1Ch
mov ebx, [esp]
inc esp
cli
mov ebp, [ebx]
```


Generatori di signature

L'idea di base è che grazie alla conoscenza di vulnerabilità e di un certo numero di exploit, si vogliono generare automaticamente signature utili a bloccare exploit non ancora rilevati "in the wild".

semantic-based modellano il comportamento di un attacco: se la rilevazione richiede l'interpretazione del modello, può essere molto dispendiosa.

content-based si basano sulla ricerca di invarianti: in realtà è piuttosto raro che la parte invariante di un exploit sia sufficientemente ricca per limitare i falsi positivi.

Hamsa

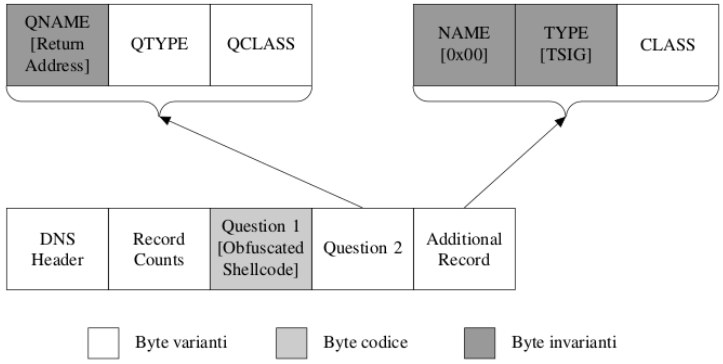
- Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao and Brian Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. IEEE Symposium on Security and Privacy, Oakland, CA, maggio 2006.
- Sistema per la generazione automatica di signature per worm polimorfici utilizzabile a livello di rete (gateway e router)
- La generazione è di tipo *content-based*, con le seguenti assunzioni rispetto alla forma di un exploit:

  invarianti byte il cui valore e fissato a priori e la cui variazione implica il fallimento dell'attacco

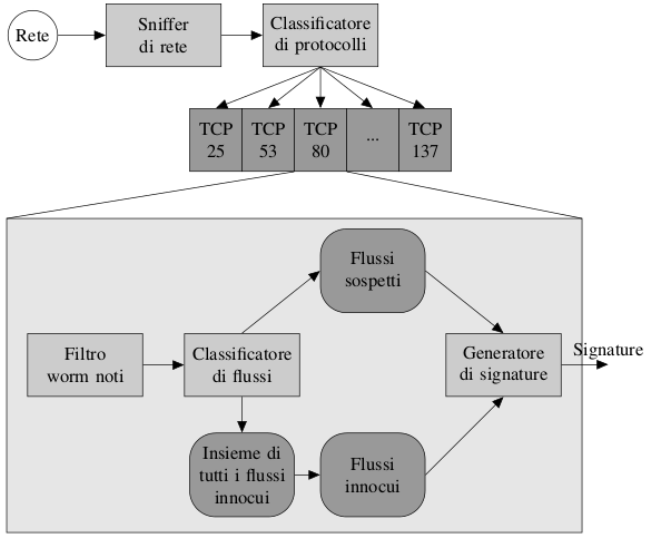
  code byte parte potenzialmente polimorfica, ma con una semantica fissa

  wildcard byte possono assumere qualsiasi valore

Esempio: Lion worm



Architettura di Hamsa



classificatore di protocolli considera i flusso TCP (o i pacchetti UDP) e li classifica secondo la porta destinazione

Firme Hamsa

- Sono dette conjuction signature: consiste in un insieme di stringhe e un flusso viene considerato malevolo se contiene tutte le stringhe, indipendentemente dall'ordine.
- Si tratta in realtà di *multi-insiemi* di token, cioè insiemi in cui un elemento può apparire piú volte.

Code-Red II	{'.ida?':1, '%u780':1, ' HTTP/1.0\r\n':1,'GET /':1, '%u':2}
ATPhttpd	{'\x9e\xf8':1, ' HTTP/1.1\r\n':1, 'GET /': 1}

I token indicati devono comparire in un unico flusso e con un numero di occorrenze maggiore o uguale a quello indicato.

Algoritmo di generazione delle firme

Input: Insieme degli invarianti I, insieme dei flussi malevoli M e dei flussi innocui N, vettore u dei falsi positivi massimi
Output: Signature S per un worm presente in M

```
S = creaSignatureVuota()
SignatureCandidata = S
VettoreSignature = []
i = 1
while i < k do
    foreach t ∈ I do
        S = S.aggiungi(t)
        FP = calcolaFalsiPositivi(S, N)
        if FP < u[i] then
            TP = calcolaVeriPositivi(S, M)
            if SignatureCandidata.TP < TP then
                SignatureCandidata = S
            end
        end
        S = S.rimuovi(t)
    end
    if SignatureCandidata == creaSignatureVuota() then
        break
    end
    VettoreSignature.appendi(SignatureCandidata)
    S = SignatureCandidata
    SignatureCandidata = creaSignatureVuota()
    i = i + 1
end
foreach S ∈ VettoreSignature do
    calcolaPunteggio(S)
end
return S con punteggio massimo
```

- k è il numero di token in $\mathcal{I}$
- $\text{calcolaPunteggio}(\mathcal{S}) = -log_{10}(\delta + FP_{\mathcal{S}}) + a \cdot TP_{\mathcal{S}} + b \cdot \text{lunghezza}(\mathcal{S})$
- tutti i parametri sono scelti in modo empirico (anche per la classificazione innocuo/sospetto)
- $u(i) = u(1) \cdot {u_r}^{(i-1)}$ con $u(1) = 0{,}15$ e $u_r = 0{,}5$

Attacchi a Hamsa

Target feature manipulation Si cerca di variare le parti considerate invarianti

Innocuous pool poisoning prima di iniziare la diffusione vera e propria e quindi prima di lanciare un attacco verso una nuova macchina, ci si preoccupa di inviare una serie di pacchetti leciti contenenti ciascuno un invariante inserito nelle parti di traffico che possono essere modificate a piacere.

Suspicious pool poisoning l'attaccante incorpora finti invarianti1 all'interno dei flussi malevoli al fine di portare il sistema alla generazione di signature che dipendono da tali finti invarianti al posto o in aggiunta agli invarianti veramente necessari.