Sicurezza dei sistemi e delle reti[1]

Mattia Monga

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

a.a. 2014/15

Lezione XIII: IDS e attacchi imprevisti

Attacchi imprevisti

I NIDS signature-based si basano sull'assunzione di **saper caratterizzare un attacco**.

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.

Attacchi imprevisti

Zero day

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

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" casuali

honeypot un sistema che viene realizzato e messo in opera solo come bersaglio

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 a 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

dead-code insertion o trash insertion: aggiungere codice senza modificare il comportamento.

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

La ricerca di stringhe costanti fallisce.

```
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

Sposta le istruzioni in modo che l'ordine del codice binario sia differente dall'ordine di esecuzione

- riordinando casualmente blocchi di istruzioni e 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: dizionari di sequenze di istruzioni equivalenti, che possono essere sostituite tra loro.
- register reassignment sostituisce l'uso di un registro con un altro equivalente.

```
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]
```

Riassumendo

Gli IDS misuse-based necessitano di *firme* degli attacchi:

- A volte non sono ancora note
- È difficile prevedere le varianti introdotte con tecniche di polimorfismo

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".

Generatori di signature

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

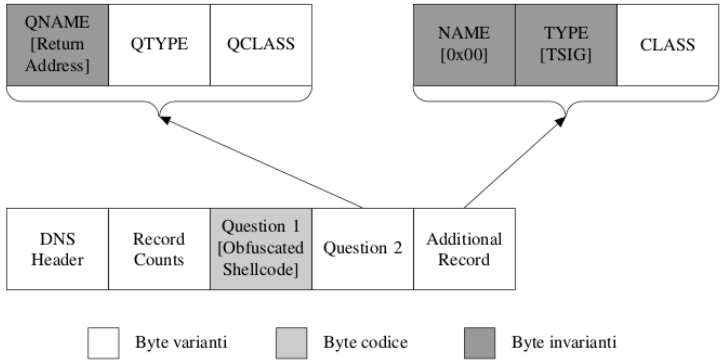
content-based si basano sulla ricerca di **invarianti**: in realtà è piuttosto raro che la parte invariante di un exploit sia sufficientemente ricca per limitare i falsi positivi.

Hamsa

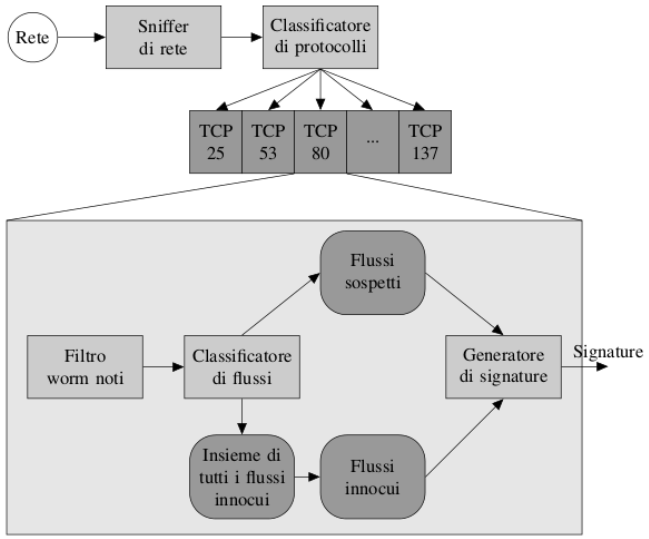
- Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao and Brian Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. IEEE Symposium on Security and Privacy, Oakland, CA, maggio 2006.
- Utilizzabile a livello di rete (gateway e router)
- *content-based*:

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

politica di selezione indica quali flussi prelevare e inviare al generatore di

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.

Firme Hamsa

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 invarianti all'interno dei flussi malevoli per portare alla generazione di signature che dipendono da tali finti invarianti al posto o in aggiunta agli invarianti veramente necessari.

Riassumendo

Generare automaticamente le varianti di un attacco:

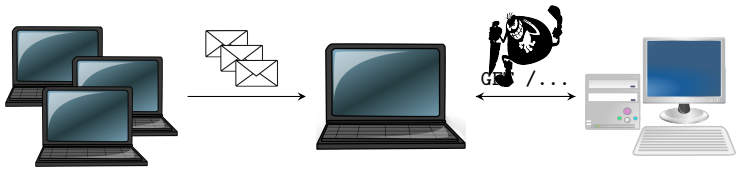
- È un'operazione con fortissime connotazioni empiriche (in generale è un obiettivo irrealizzabile)
- Come sempre, un meccanismo automatico può essere sfruttato anche dall'attaccante (*poisoning*)

Malware undeground economy

Il malware viene diffuso sfruttando vulnerabilità generiche allo scopo di compiere attacchi piú redditizi.

Phishing

1. campagna di spam
2. social engineering
3. furto credenziali & malware
4. infezione macchine



Underground economy
Vendita informazioni rubate

Goods & services	Percentage	Range of prices
Bank accounts	22%	$10-$1000
Credit cards	13%	$0.40-$20
Full identities	9%	$1-$15
Online auction site accounts	7%	$1-$8
Scams	7%	$2.50-$50/week (hosting)
Mailers	6%	$1-$10
Email addresses	5%	$0.83/MB-$10/MB
Email passwords	5%	$4-$30
Drop (request or offer)	5%	10%-20% of drop amount
Proxies	5%	$1.50-$30

Symantec

Underground economy
Furto credenziali — Portata del fenomeno

- Università di Mannheim — Limbo & ZeuS
- $\sim$ 70 dropzone
- **33 GB** di dati
- 11000 account bancari, 150000 account mail

Underground economy

Dropzone	# Machines	Data amount	Country
webpinkXXX.cn	26,150	1.5 GB	China
coXXX-google.cn	12,460	1.2 GB	Malaysia
77.XXX.159.202	10,394	503 MB	Russia
finXXXonline.com	6,932	438 MB	Estonia
Other	108,122	24.4 GB	
Total	164,058	28.0 GB	

Learning More About the Underground Economy — T. Holz, M. Engelberth, F. Freiling, 2008

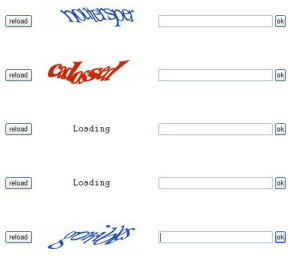
Underground economy
Malware as a service

- Bot in affitto ($\sim$ \$1000–\$2000/mese)
- MPACK: exploit toolkit a $\sim$ \$1000

Underground economy
The spam business

CAPTCHA?

- OCR, Fuzzy OCR, ...
- "Human computation"!



> 100K captcha al giorno, \$1.5–\$8 per 1000 captcha

Funzionalità del malware
Click fraud

- *Google*: 10% dei *click* sono fraudolenti ($\sim$ \$1B)
- Clickbot.A ($\sim$ 50k host infetti)
- molti "clickbot" commerciali
- ClickJacking