

Offuscamento di Programmi Java

Stelvio Cimato

DIA

Università di Salerno

cimato@dia.unisa.it

Sicurezza del sistema

L'esecuzione di programmi può provocare danni al sistema locale
Rimedi:

- Sandboxing
- Tool di protezione dai Virus



Sicurezza del software

Un host malizioso può cercare di accedere o modificare le informazioni contenute in una applicazione

Rimedi

- Tecniche di protezione del software da sistemi o utenti maliziosi



Protezione del software

Software contiene

- Informazioni riservate, chiavi segrete
- Codice (sorgente, oggetto, intermedio, ...)

Proprietà intellettuale:

- Distribuzione del software
- Pirateria

Reverse Engineering

Con abbastanza risorse in tempo e sforzo, un programmatore è sempre capace di fare il reverse engineering di una applicazione

Per applicazioni Java il compito è più facile:

- Sono distribuite in formato semplice da decodificare
- Maggior parte della computazione avviene in librerie standard

Attacchi

Attacchi ad Accesso Diretto (van Oorschot, 2003)

Accesso ad una copia locale del codice su una macchina locale

- Analisi
 - Statica: sul codice del programma
e.g.: disassemblaggio, decompilazione, ...
 - Dinamica: a run-time
e.g.: tracing, ...
- Tampering
 - Statica: senza eseguire il codice (on disk)
e.g.: "crack": dynamic analysis → static attack
 - Dinamica: durante l'esecuzione (in memory / processor)
- Software Differential Analysis
 - Confronto fra diverse versioni o copie del software
- Collusion Attacks
- Replay Attacks

Tools

Reverse Engineering:

- Osservazione del comportamento dinamico
- Recupero dal codice binario di astrazioni
- Modifica dinamica del codice e osservazione

Disassemblatori

Decompilatori

Debugger

Emulatori

Monitoring tools



Protezione del Codice

Mezzi Legali

- No Electronic Theft
- Digital Millennium Copyright Act (DMCA)
- Leggi per la protezione dei diritti d'autore
- ...
- Decreto Urbani
- ...



Protezione del Software

Goal: Rendere il reverse engineering difficile

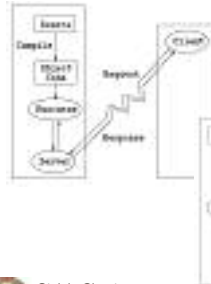
Mezzi tecnici:

- Client-server
- Cifratura
- Codice nativo firmato
- Offuscamento del codice
- White-box cryptography

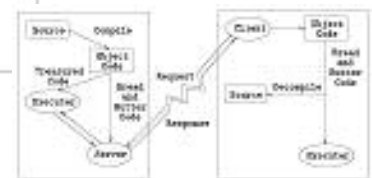


Client-Server

Client-server



Partial client-server



Client-Server

Client-server

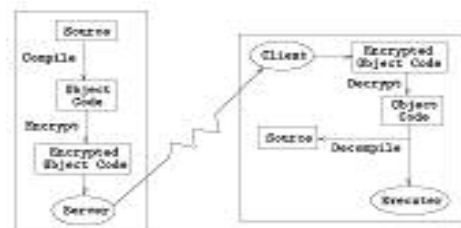
- Vantaggi
 - + Nessun accesso fisico al codice
- Svantaggi
 - Larghezza di banda e latenza
 - Network failures
 - Singolo punto di failure, bottleneck

Partial client-server

- Vantaggi
 - + Il codice è diviso in due parti
- Svantaggi
 - Comunicazioni frequenti



Cifratura del codice



Cifratura del codice

Software

- Possibilità di intercettare il codice durante l'esecuzione

Hardware

- + Nessuna possibilità di intercettare il codice
- Ogni processore ha bisogno di una sua interfaccia
- Costoso



Codice Nativo Cifrato

Compilatori Just-In-Time (JIT) :

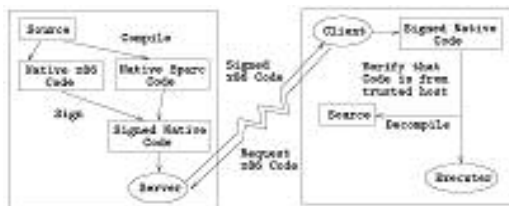
Java bytecode viene tradotto in codice nativo a run-time

Way-ahead-of-time (WAT) compilation:

si evita la compilazione quando si esegue ripetutamente Java bytecodes



Codice Nativo Cifrato



Codice Nativo Cifrato

Bytecode

- + Platform independent
- + La verifica del bytecode garantisce codice legale.
- Facile da decompilare

Codice nativo cifrato

- + Difficile da decompilare
- + Firmato per evitare manomissioni
- Non è platform independent
- Non è possibile la verifica del bytecode (sono possibili operazioni illegali)



Offuscamento del Codice

Idea: trasformare una applicazione:

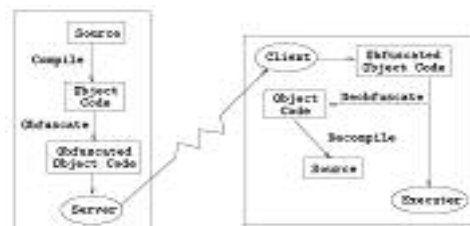
- Stesse funzionalità
- Comportamento interno difficile da capire

Sicurezza si basa su:

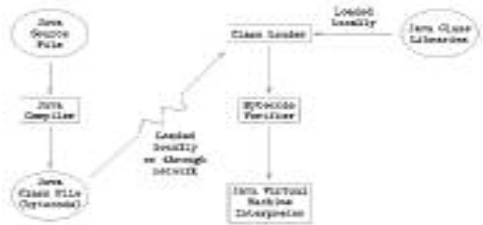
- Sofisticatezza delle trasformazioni
- Potenza dell'algoritmo di offuscamento
- Risorse disponibili al deoffuscatore



Offuscamento del Codice



Java Obfuscation



Decompilatori

Dal bytecode permettono di recuperare la struttura del sorgente:

- Da file .class a file .java

Esempi:

- Free
 - Javap, JAD, JODE
- Commerciali
 - SourceAgain, Klassmaster, ecc

Offuscamento del Codice

Vantaggi:

- + Mantiene l'indipendenza dalla piattaforma
- + Non serve la firma del codice
- + Non ci sono ritardi di rete (vs. client-server)
- + Non c'è bisogno di hardware specifico (vs. encryption)

Svantaggi:

- Tecnica euristica: molte trasformazioni non sono one-way.
- Difficile misurare la bontà di una trasformazione
- ± Difficile l'aggiornamento del software, (vs. object oriented languages)

White-Box Cryptography

White-box cryptography (Chow *et al.*, 2002): Nascondere le informazioni segrete quando l'attaccante ha pieno accesso al software crittografico

Tecnica matematica

$$E'_k = G \circ E_k \circ F^{-1}$$

con:

E_k : funzione di cifratura con chiave k

F : codifica input scelta a caso

G : codifica output scelta a caso

Trasformazione:

Algoritmo crittografico + chiave → serie di tabelle di lookup (con chiave nascosta)

Trasformazioni Offuscanti

Offuscamento:

applicazione di una trasformazione di offuscamento risulta in un programma funzionalmente equivalente ma più difficile da analizzare

“Funzionalmente equivalente” ≈ “stesso comportamento osservabile”

$T: P \rightarrow P'$ è una *obfuscation transformation* se P e P' hanno lo stesso comportamento osservabile.

1. Se P non termina o termina con errore, allora P' può terminare o non terminare
2. Altrimenti P' deve terminare e produrre lo stesso output di P

Classificazione

Classificazione delle trasformazioni

- *Tipo* dell'informazione da modificare
- *Come* l'informazione viene modificata
 - Aggregation: rompere o creare astrazioni definite dall'utente
 - Ordering: modificare l'ordine delle dichiarazioni o della computazione



Qualità

E' possibile individuare dei criteri per misurare la qualità di una trasformazione. Le seguenti voci misurano:

- **Potenza**: quanta oscurità la trasformazione aggiunge al programma
 - **Resilienza**: quanto è lo sforzo richiesto per rompere la trasformazione
 - **Costo**: quanto sforzo computazionale aggiuntivo la trasformazione aggiunge all'applicazione offuscata
 - **Stealth**: quanto il codice introdotto dalla trasformazione si adatta al codice originale
- ➔ *Quality and appropriateness*



Layout Obfuscation

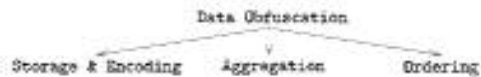
Agisce sulla struttura lessicale del programma:

- Rimozione della formattazione del codice (disponibile anche nei file .class)
- Rimozione delle informazioni di debug ("*stripping*")
- Renaming degli identificatori ("*scrambling*")



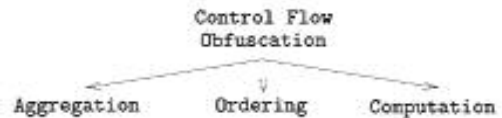
Data obfuscation

Agisce sui tipi, sulle strutture dati, sulle dichiarazioni usati nell'applicazione.



Control Flow Obfuscation

- Raggruppamento di istruzioni (inline procedure, ...)
- Alterazione dell'ordine di esecuzione (loops, ...)
- Nascondere il flusso di controllo (inserimento di dummy code, ...)



Trasformazioni Preventive

- Hanno lo scopo di impedire il funzionamento dei decompilatori



- **Targeted**: Contrattaccano specifici strumenti di analisi (es. Mocha)
- **Inherent**: Rendono difficile l'impiego delle tecniche di deobfuscation note



Trasformazioni preventive

```
for(i=1;i<=10;i++)
  A[i]=1;

int B[10];
for(i=10;i>=1;i--)
  A[i]=1;
  B[i]=B[i-1]/2;
```

L'indipendenza dei dati blocca l'inversione della trasformazione T



Data Obfuscation

Le trasformazioni oscurano i tipi e le strutture dati usati nell'applicazione
Possono influenzare:

- Storage: il modo in cui i dati vengono memorizzati
- Encoding: il modo in cui un dato viene interpretato
- Aggregation and ordering



Encoding

$I' = c_1 * i + c_2$ dove $c_1 = 8$ e $c_2 = 3$

```
int i=1;
while (i < 1000) {
  ... A[i] ...;
  i++;
}

int i=11;
while (i<8003) {
  ... A[(i-3)/8] ...;
  i+=8;
}
```



Storage: Split delle Variabili

Una variabile V può essere rappresentata da altre k variabili:

- $f(v_1, \dots, v_k)$ mappa i valori nel corrispondente valore di V
- $G(V)$ che mappa V nelle k variabili
- Nuove operazioni corrispondenti



Storage: Promote Variabili

Le variabili possono essere "promosse" da una classe specifica ad una classe generale.

Es. in Java un intero può essere promosso in un oggetto

```
int i=1;
while (i<9)
  ...A[i]...
  i++;

int i= new Int(1);
while (i.value<9)
  ...A[i.value]...
  i.value++;
```



Encoding: Change Lifetime

E' possibile agire sul tempo di vita di una variabile.

Ad es. se due procedure P e Q non vengono mai eseguite contemporaneamente e fanno uso di due variabili locali, è possibile utilizzare una singola variabile globale condivisa



Conversione di dati statici

Un dato statico, es. una stringa, può essere convertito in una procedura che emette in output il dato.

Es. invece delle stringhe "AAA"; "BAAA", "CCB", una funzione G produce i valori dati con input 1,2,3, $G(1) = "AAA"$, ecc.



Aggregazione

- Merge di variabili
 - Due variabili possono essere fuse insieme. Es. da due variabili a 32 bit una a 64 bit.
- Merge o Split di Array
 - Merge, da due array se ne usa uno solo (usando indici pari o dispari)
 - Fold (flatten), si aumenta o diminuisce il numero delle sue dimensioni
- Modifica di relazioni di ereditarietà



Ordering

Si modifica l'ordine:

- Delle dichiarazioni delle variabili
- Delle dichiarazioni dei metodi
- Dei parametri dei metodi
- Degli elementi contenuti in un array



Control Flow Obfuscation

- Costrutti Opachi
- Computation obfuscations
- Aggregation obfuscations
- Ordering obfuscations



Opaque Constructs

Una variabile V è opaca in un punto p in un programma, se la proprietà q di V nel punto p è nota al tempo in cui si opera la trasformazione offuscante

Un predicato P il cui valore è opaco si indica con:

$PV \quad PT \quad P?$



Costrutti Opachi

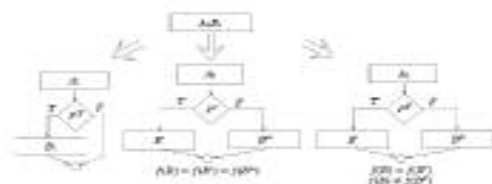
La resilienza (quanto sia resistente ad un deoffuscatore) di un costrutto è:

- *Trivial*: si aggirano con *static local analysis*
es: con l'uso di pattern matching di funzioni ben note
- *Weak*: si aggirano con *static global analysis*
es.: con l'uso di un automatic theorem prover



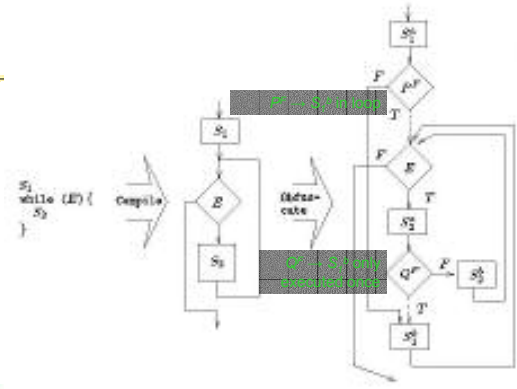
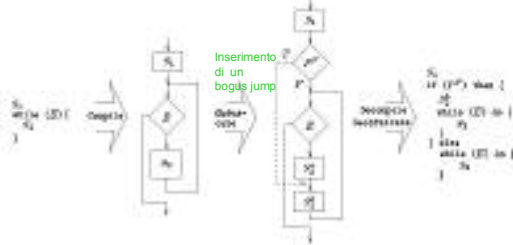
Inserimento di codice

1. Inserimento di codice irrilevante
2. Inserimento di due versioni offuscate
3. Scelta sempre della versione corretta



High-level language breaking features

- Inserimento di codice oggetto senza corrispondenze del sorgente



Control flow de-abstraction

- High-level sequence → equivalent low-level sequence

Esempio

```

int i;
i = 0;
loop:
  if (i >= 100) goto endloop;
  printf("%d\n", i);
  i++;
  goto loop;
endloop;

```

↔

```

int i;
for (i = 0; i < 100; i++)
  printf("%d\n", i);

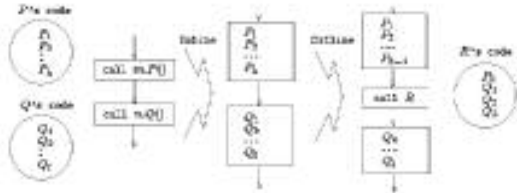
```



Aggregation Obfuscations

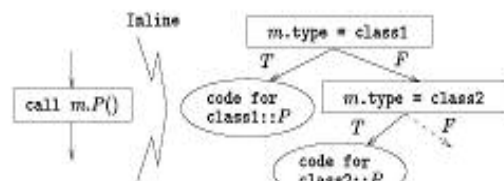
Idea: spezzare computazioni che avvengono insieme logicamente o unire computazioni logicamente separate

- Inline: rimuove astrazioni procedurali dal programma
- Outline: crea una subroutine da una sequenza



Aggregation Obfuscations

Modifica con il tipo a runtime dell'oggetto



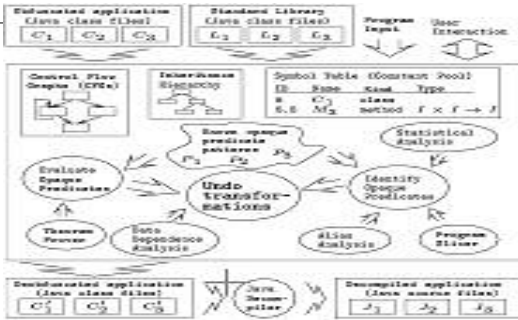
Ordering Obfuscations

Idea:

- Alternare le chiamate di procedura in un programma
- Cambiare l'ordine delle istruzioni in un programma
- Usata in congiunzione con trasformazioni di aggregazione



Deoffuscator



References

- Software protection and Application Security: Understanding the Battleground (A. Main and P.C. van Oorschot), LNCS 2003
- A Taxonomy of Obfuscating Transformation (C. Collberg, C. Thomborson and D. Low), TR 1998

