

# Offuscamento del bytecode Java

Corso di Sicurezza su reti  
Prof.: Alfredo De Santis

Anno Accademico 2003-2004

## A cura di :

Fiorillo Antonio  
Lo Prete Giovanni  
Miliana Antonella  
Santillo Pasquale

1

## Sommario

- Protezione del software ←
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

2

## Protezione del software

- Il software contiene
  - Informazioni riservate, chiavi segrete
  - Codice (sorgente, oggetto, intermedio, ...)
- Il software è una proprietà intellettuale di chi lo produce
- Il reverse engineering consiste nel ricavare il codice sorgente da una data applicazione
- Un malintenzionato può così accedere alle informazioni segrete di un'applicazione
- Occorre preservare questa proprietà dalla pirateria

3

## Protezione del software

- Una volta creata, un'applicazione deve essere distribuita in una qualche forma: codice nativo, codice indipendente dalla macchina (bytecode).
- Il codice nativo è difficile da decompilare ma se un programmatore ha abbastanza risorse in tempo e sforzo, è sempre capace di fare il reverse engineering di un'applicazione.
- Il bytecode ha un formato indipendente dall'architettura.
- Dal bytecode è possibile recuperare in modo estremamente facile la struttura del sorgente.

4

## Protezione del software

Consideriamo il seguente scenario

- Alice è una sviluppatrice software
- Bob è uno sviluppatore concorrente (malizioso) che vuole trarre vantaggi commerciali dall'accesso ai segreti (algoritmi, strutture dati) delle applicazioni create da Alice

5

## Protezione del software

Alice vuole proteggere i segreti delle proprie applicazioni

- Mezzi legali
  - le leggi sul copyright non coprono adeguatamente gli artefatti software, quindi poco utilizzabili
- Mezzi tecnici
  - Rendere il reverse engineering impossibile o almeno economicamente proibitivo

6

## Protezione del software : strategie

- Evitare che Bob possa avere accesso fisico all'applicazione o almeno alle parti più importanti di essa.
- Esistono varie strategie che Alice può adottare
  - 1) vendere i suoi servizi
  - 2) cifratura del codice
  - 3) compilatori just-in-time
  - 4) offuscamento del codice

7

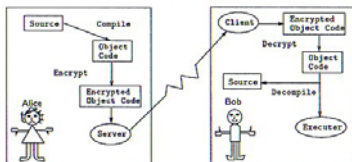
## Strategie : vendere i servizi

- Chi vuole usare l'applicazione creata da Alice deve collegarsi al suo sito e pagare una certa quota
- Vantaggi : Bob non avrà mai accesso fisico all'applicazione
- Svantaggi : applicazione più lenta
  - Inerente lentezza della comunicazione in rete
  - Soluzione parziale : dividere l'applicazione in due parti
    - Una pubblica eseguita sul sito dell'utente
    - Una privata eseguita in remoto sul sito di Alice.

8

## Strategie : Cifratura

- Cifrare il codice prima di inviarlo agli utenti
- Svantaggi : è efficace solo se l'intero processo di decifratura/esecuzione è effettuato in hardware
  - Bob potrebbe intercettare il codice che passa attraverso la VM



9

## Strategie : Compilatori just-in-time

- I compilatori just-in-time traducono al volo il bytecode Java al codice nativo.
- Alice crea una versione in codice nativo della sua applicazione per tutte le piattaforme più popolari.
- Vantaggi : Bob ha accesso solo al codice nativo
- Svantaggi : Il verificatore della VM non può controllare il codice ricevuto
  - può contenere codice maligno
- Soluzione : Alice firma il codice dimostrando che è affidabile

10

## Strategie : Offuscamento del codice

- Alice dà in input la sua applicazione ad un offuscatore
- L'offuscatore è un programma che trasforma l'applicazione in una che sia funzionalmente identica all'originale ma che è molto più difficile per Bob da capire
- Vantaggi
  - Mantiene l'indipendenza dalla piattaforma
  - Non serve la firma del codice
  - Non ci sono ritardi di rete
  - Non c'è bisogno di hardware specifico (es. cifratura)

11

## Strategie : Offuscamento del codice

- Svantaggi
  - accesso diretto all'applicazione
  - la probabilità di reverse engineering non è nulla.
  - Bob può usare deoffuscatori automatici
- Il livello di sicurezza aggiunto da un offuscatore ad un'applicazione dipende da
  - le sofisticazioni delle trasformazioni impiegate dall'offuscatore,
  - la potenza degli algoritmi di offuscamento disponibili
  - la quantità di risorse (tempo e spazio) disponibili al deoffuscatore.

12

## Sommario

- Protezione del software ←
- Design di un offuscatore Java ←
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

13

## Design di un Offuscatore Java

- L'input del tool è un'applicazione Java
  - insieme delle classi Java.
- Contiene una serie di trasformazioni offuscanti
- Si può selezionare il livello di offuscamento (la potenza) e il massimo overhead (tempo/spazio) che l'offuscatore aggiunge all'applicazione
- occorre effettuare una fase di preprocessing per raccogliere vari tipi di informazioni sull'applicazione per determinare le trasformazioni più adatte da applicare.

14

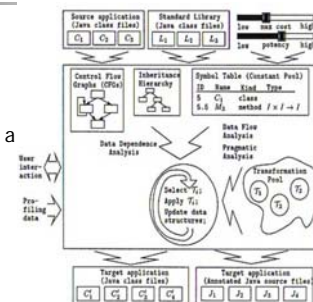
## Design di un Offuscatore Java

- Molti dati sono raccolti usando strumenti della teoria dei compilatori
  - l'analisi del flusso di controllo interprocedurale
  - l'analisi della dipendenza dei dati
  - ...
- **L'output del tool è una nuova applicazione funzionalmente equivalente a quella originale ...**
  - ...ma quasi irriconoscibile nella struttura.

15

## Design di un Offuscatore Java

- Permette di inserire dei profili per evitare che trasformazioni troppo costose siano applicate a parti del codice usate molto frequentemente.
- Contiene un gran numero di trasformazioni



16

## Sommario

- Protezione del software
- Design di un offuscatore Java ←
- Classificazione delle trasformazioni offuscanti ←
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

17

## Classificazione delle trasformazioni offuscanti

- Le trasformazioni si classificano e si valutano rispetto alla loro
  - **potenza** - il grado di confusione causato nell'utente malizioso
  - **resilienza** - quanto il codice offuscato è resistente ad un attacco deoffuscante
  - **costo** - quanto overhead è aggiunto all'applicazione originale.
- Il **comportamento osservabile** è definito debolmente come "comportamento sperimentato dall'utente"
  - possibilità di side effect (creare file, inviare messaggi sulla rete)

18

## Definizione di trasformazione offuscante

**Definizione 1 (Trasformazione offuscante)**\_ Sia  $P \xrightarrow{\tau} P'$  una trasformazione di un programma sorgente P in un altro programma P'.

$P \xrightarrow{\tau} P'$  è una trasformazione offuscante se P e P' hanno lo stesso comportamento osservabile e più precisamente devono essere mantenute le seguenti condizioni :  
se P fallisce nel terminare o termina con una condizione di errore, allora P' potrebbe terminare oppure no.  
Altrimenti, P' deve terminare e produrre lo stesso output di P.

19

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

20

## Valutazione delle trasformazioni offuscanti

- Misura della Potenza
- Misura della resilienza
- Misure del costo di esecuzione
- Misura di qualità

21

## Metriche della complessità di una trasformazione offuscante (1)

$E(X)$  è la complessità di un componente software X. F è una funzione o metodo, C è una classe, e P è un programma.

$\mu_1$  **Lunghezza del programma** :  $E(P)$  aumenta col numero di operatori e di operandi in P.

$\mu_2$  **Cyclomatic Complexity** :  $E(F)$  aumenta col numero di predicati in F.

$\mu_3$  **Complessità annidata** :  $E(F)$  aumenta con il numero di livelli annidati di predicate condizionali in F .

$\mu_4$  **Complessità del flusso di dati** :  $E(F)$  aumenta col numero di riferimenti a basic block in F .

22

## Metriche della complessità di una trasformazione offuscante (2)

$\mu_5$  **Complessità fan-in/out** :  $E(F)$  aumenta col numero di parametri formali di F, e con il numero di strutture dati globali lette o aggiornate da F.

$\mu_6$  **Complessità delle strutture dati** :  $E(P)$  aumenta con la complessità delle strutture dati statiche dichiarate in P .

- La complessità di una variabile scalare è costante.
- La complessità di un array aumenta col numero di dimensioni e con la complessità del tipo degli elementi.
- La complessità di un record aumenta con il numero e la complessità dei suoi campi.

23

## Metriche della complessità di una trasformazione offuscante (2)

**Metrica 00** :  $E(C)$  aumenta con

- $\mu_{7a}$ , il numero di metodi in C,
- $\mu_{7b}$ , la profondità (la distanza dalla radice) di C nell'albero di ereditarietà
- $\mu_{7c}$ , il numero di sottoclassi dirette di C
- $\mu_{7d}$ , il numero di altre classi alle quali C è associata
- $\mu_{7e}$ , il numero di metodi che possono essere eseguiti in risposta all'invio di un messaggio ad un oggetto di C
- $\mu_{7f}$ , il grado di quali metodi di C non fanno riferimenti allo stesso insieme di variabili d'istanza.

24

## Potenza della trasformazione

**Definizione 2 (Potenza della trasformazione)** \_ Sia  $T$  una trasformazione che conserva il comportamento, tale che  $P \xrightarrow{T} P'$ , cioè che trasforma un programma sorgente  $P$  in un programma  $P'$ . Sia  $E(P)$  la complessità di  $P$

$T_{pot}(P)$ , la potenza di  $T$  rispetto al programma  $P$ , è la misura di quanto  $T$  cambia la complessità di  $P$ . Essa è definita nel seguente modo :

$$T_{pot}(P) = E(P')/E(P)-1.$$

$T$  è una trasformazione offuscante potente se  $T_{pot}(P) > 0$ .

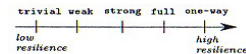
• **una trasformazione è potente** se essa fa un buon lavoro nel confondere Bob

• si misura su una scala a tre punti (low, medium, high).

25

## Misura della resilienza

- $T$  è una trasformazione
  - **locale** se ha effetto su un singolo basic block di un **grafo del flusso di controllo** (CFG)
  - **globale** se ha effetto sull'intero CFG
  - **interprocedurale** se ha effetto sul flusso di informazioni tra le procedure
  - **interprocesso** se ha effetto sulle interazioni tra thread di controllo in esecuzione indipendente.
- Una trasformazione è potente se riesce a confondere un lettore umano, ma è resiliente se riesce a confondere un deoffuscatore automatico.
- Si misura su una scala che va da "trivial" a "one-way"



26

## Misura della resilienza

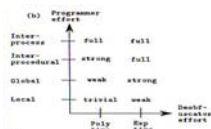
**Definizione 3 (Resilienza della trasformazione)** \_ Sia  $T$  una trasformazione che conserva il comportamento, tale che  $P \xrightarrow{T} P'$ , cioè che trasforma un programma sorgente  $P$  in un programma  $P'$ .

$T_{res}(P)$  è la resilienza di  $T$  rispetto al programma  $P$ .

$T_{res}(P)$  = one-way se l'informazione è rimossa da  $P$  in modo che  $P$  non possa più essere ricostruito da  $P'$ .

Altrimenti,  $T_{res} = \text{Resilienza}(T_{\text{deoffuscatore}}, T_{\text{programmatore}})$ ,

dove **Resilienza** è la funzione definita nella matrice



27

## Misura del costo di esecuzione

- è l'overhead di tempo/spazio che una trasformazione aggiunge ad un'applicazione.
- è classificato su una scala a 4 punti (free, cheap, costly, dear)

**Definizione 4 (Costo della trasformazione)** \_ Sia  $T$  una trasformazione che conserva il comportamento,

• tale che  $P \xrightarrow{T} P'$ , cioè che trasforma un programma sorgente  $P$  in un programma  $P'$ .  $T_{cost}(P)$  è l'overhead in termini di tempo/spazio di  $P'$  rispetto a  $P$  e può essere :

- dear**, se  $P'$  richiede una quantità esponenziale di risorse rispetto a  $P$ .
- costly**, se  $P'$  richiede più di  $O(n^p)$ ,  $p > 1$ , risorse rispetto a  $P$ .
- cheap**, se  $P'$  richiede più di  $O(n)$  risorse rispetto a  $P$ .
- free**, se  $P'$  richiede più di  $O(1)$  risorse rispetto a  $P$ .

28

## Misura di qualità

**Definizione 5 (Trasformazione della qualità)** \_  $T_{qual}(P)$ , la qualità di una trasformazione  $T$ , è definita come la combinazione della potenza, della resilienza e del costo di  $T$  :

$$T_{qual}(P) = (T_{pot}(P), T_{res}(P), T_{cost}(P)).$$

29

## Trasformazioni del layout

- Sono trasformazioni tipiche dei correnti offuscatori Java come CREMA
- Si rimuovono le informazioni di formattazione del codice nei .class
  - è one-way
    - la formattazione originale non può più essere recuperata
  - è poco potente
    - c'è pochissima semantica contenuta nella formattazione
  - è free
    - la complessità di tempo/spazio dell'applicazione non ne risente
- si mischiano i nomi degli identificatori
  - è one-way
  - è free
  - è più potente della precedente
    - Gli identificatori contengono molte informazioni

30

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

31

## Trasformazioni del controllo

- Predicati opachi
- Trasformazioni computazionali
- Trasformazioni per aggregazione
- Trasformazioni dell' Ordine

32

## Trasformazioni del controllo

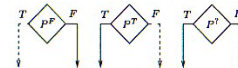
- Le trasformazioni del controllo tentano di oscurare il flusso di controllo dell'applicazione sorgente.
  - **Aggregazione** : spezzano le computazioni logicamente affini oppure fonde computazioni non logicamente affini
  - **Ordine di controllo** : randomizza l'ordine con cui le computazioni sono eseguite
  - **Computazioni del flusso di controllo** : inseriscono nuovo codice
    - Creano molto overhead

33

## Predicati opachi

**Definizione 6 (Costrutti opachi)** Una variabile  $V$  è opaca nel punto  $p$  di un programma, se  $V$  ha una proprietà  $q$  in  $p$  che è conosciuta al momento dell'offuscamento. Indicheremo ciò con  $V_p^q$  o con  $V^q$  se  $p$  è libero dal contesto.

Un **predicato**  $P$  è opaco nel punto  $p$  di un programma, se il suo esito è conosciuto al momento dell'offuscamento. Indicheremo con  $P_p^T$  ( $P_p^F$ ) se  $P$  è sempre valutato con False(True) nel punto  $p$ , e  $P_p^?$  se  $P$  a volte è valutato True e altre volte è valutato False. Ancora,  $p$  è omesso se è libero dal contesto.



- misura della resilienza di una variabile o di un predicato : trivial, ..., one-way.
- misura del costo aggiunto di un costrutto opaco : free, ..., dear.

34

## Costrutti opachi trivial e weak

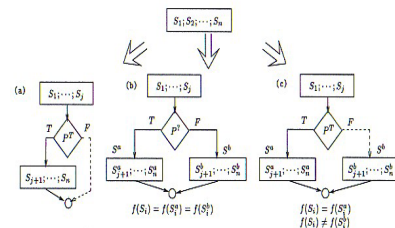
- (a) **trivial**
  - Rotto solo attraverso un'analisi locale statica.
  - Un'analisi locale è ristretta ad un singolo basic block del CFG
- (b) **weak**
  - rotto solo attraverso un'analisi globale statica.
  - Un'analisi globale è ristretta ad un singolo CFG

<pre> { int v, a=5; b=6;   v=11 = a + b;   if (b &gt; 5) ...   if (random(1,5) &lt; 0) ... }                 </pre> <p>(a)</p>	<pre> { int v, a=5; b=6;   if (...) ...   : (b is unchanged)   if (b &lt; 7) a++;   v=36 = (a &gt; 5)?v=b*b:v=b }                 </pre> <p>(b)</p>
--	---

35

## Trasformazioni computazionali : Inserire codice morto o irrilevante

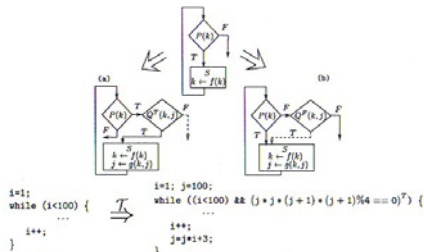
- abbiamo il basic block  $S_1; S_2; \dots; S_n$ ;
- aumentano le metriche  $\mu_2$  e  $\mu_3$



36

## Trasformazioni computazionali : Estendere le condizioni dei cicli

- Il predicato usato  $x^2(x+1)^2 \equiv 0 \pmod{4}$  è sempre valutato True



37

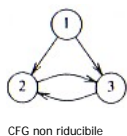
## Trasformazioni computazionali : Convertire un grafo di flusso riducibile ad uno non riducibile (1)

- Trasformazioni *language-breaking*
  - il bytecode Java ha un'istruzione goto mentre il linguaggio Java no
  - il CFG prodotto dai programmi Java sarà sempre riducibile
  - il bytecode Java può esprimere CFG non riducibili
  - i CFG non riducibili rendono inutilizzabili molte tecniche di ottimizzazione utilizzate dai compilatori
    - sono rari
- Un CFG  $G$  è *riducibile* se e solo se possiamo partizionare gli archi in due gruppi disgiunti, spesso chiamati archi in avanti e archi all'indietro, aventi le seguenti proprietà :
  - Gli *archi in avanti* formano un grafo aciclico in cui ogni nodo può essere raggiunto dal nodo iniziale di  $G$ .
  - Gli *archi all'indietro* consistono solo di archi le cui teste dominano le loro code

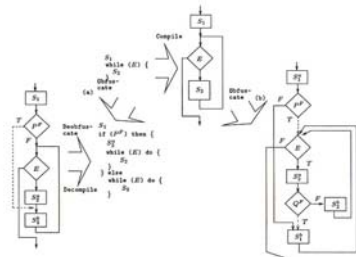
38

## Trasformazioni computazionali : Convertire un grafo di flusso riducibile ad uno non riducibile (2)

- Un ciclo con più di un header non è riducibile
- Idea : convertiamo un ciclo in un ciclo con header multipli



CFG non riducibile



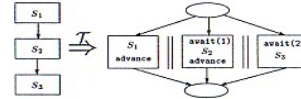
## Trasformazioni computazionali : Parallelizzazione del codice

- 1. Possiamo creare dei processi fantoccio che non eseguono alcun compito utile
- 2. possiamo spezzare una sezione sequenziale dell'applicazione in sezioni multiple eseguite in parallelo.
- Vantaggi**
  - alta resilienza e potenza
  - analisi statica molto difficile
    - numero di possibili percorsi di esecuzione di un programma cresce esponenzialmente

43

## Trasformazioni computazionali : Parallelizzazione del codice

- Svantaggi**
  - su macchine uniprocessore l'applicazione gira più lentamente
  - la parallelizzazione risulta difficile in presenza di dipendenza dei dati
    - Occorrono costrutti di sincronizzazione



44

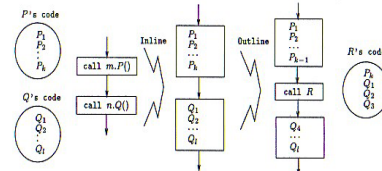
## Trasformazioni per aggregazione

- I programmatori vincono l'inerente difficoltà del programmare usando varie astrazioni tra cui l'astrazione procedurale
- Queste astrazioni sono rimosse usando
  - inlining
  - outlining
  - interleaving
  - clonazione
- idea comune
  - (1) il codice che il programmatore ha aggregato insieme in un metodo deve essere rotto e sparpagliato nel programma
  - (2) il codice che sembra non essere logicamente connesso deve essere aggregato in un metodo.

45

## Trasformazioni per aggregazione : Metodi Inline e Outline (1)

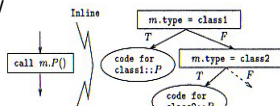
- Inlining** si rimpiazza una chiamata a procedura con il corpo della procedura chiamata e si rimuove la procedura
  - altamente resiliente (generalmente one-way)
  - Non c'è più traccia dell'astrazione
- outlining** si trasforma una sequenza di espressioni in una subroutine



46

## Trasformazioni per aggregazione : Metodi Inline e Outline (2)

- Nei linguaggi object oriented come Java, l'inlining può non sempre essere una trasformazione totalmente one-way



- L'attuale procedura chiamata dipenderà dal tipo di m determinato a run-time.
- occorre applicare l'inlining a tutti i possibili metodi e ramificare su m

47

## Trasformazioni per aggregazione : Metodi Interleave

- La scoperta del codice interfogliato è un difficile compito di reverse engineering
- idea
  - fondere i corpi e la lista dei parametri dei due metodi
  - aggiungere un parametro extra (o parametro globale) per discriminare tra le chiamate ai metodi individuali.

```

class C {
  method M1 (T1 a) {
    S1^1; ... S1^m;
  }
  method M2 (T1 b; T2 c) {
    S2^1; ... S2^m;
  }
}

class C' {
  method M (T1 a; T2 c; int V) {
    if (V == p) {
      S1^1; ... S1^m;
    }
    else {
      S2^1; ... S2^m;
    }
  }
}

{ C x=new C;
  x.M1(a); x.M2(b, c); }

{ C' x=new C';
  x.M(a, c, V^p);
  x.M(b, c, V^q); }

```

48

## Trasformazioni per aggregazione : Metodi di clonazione

- per capire il comportamento di una subroutine sono importanti anche i differenti ambienti in cui è stata chiamata
- Idea : offuscare il punto di una chiamata ad un metodo per far sembrare che differenti routine sono state chiamate

```

class C {
  method m (int x)
  { S1...S4 }
}
{ C x = new C;
  x.m(5); ... x.m(7);
}

class C1 {
  method m (int x)
  { S1'...S4' }
}
class C2 inherits C1 {
  method m (int x)
  { S1''...S4'' }
}
{ C1 x ;
  if (P) x=new C1 else x=new C2;
  x.m(5); ...; x.m(7);
}
    
```

49

## Trasformazioni per aggregazione : Trasformazioni dei cicli

- progettate con l'intento di aumentare le prestazioni) delle applicazioni numeriche.
- Alcune di queste trasformazioni sono molto utili in quanto accrescono la complessità delle metriche
  - loop blocking
  - loop unrolling
  - loop fission
- aumentano la dimensione del codice totale e il numero di condizioni dell'applicazione sorgente
- Singolarmente la resilienza è bassa
- L'uso combinato innalza drammaticamente la resilienza

50

## Trasformazioni per aggregazione : Trasformazioni dei cicli – loop blocking

- Il loop blocking è usato per migliorare il comportamento della cache di un ciclo

```

for(i=1,i<=n,i++)
for(j=1,j<=n,j++)
  a[i,j]=b[j,i]

for(I=1,I<=n,I+=64)
for(J=1,J<=n,J+=64)
for(i=I,i<=min(I+63,n),i++)
for(j=J,j<=min(J+63,n),j++)
  a[i,j]=b[j,i]
    
```

51

## Trasformazioni per aggregazione : Trasformazioni dei cicli – loop unrolling

- Il loop unrolling replica il corpo di un ciclo una o più volte.

```

for(i=2,i<=(n-1),i++)
  a[i] += a[i-1]*a[i+1]

for(i=2,i<=(n-2),i+=2) {
  a[i] += a[i-1]*a[i+1]
  a[i+1] += a[i]*a[i+2]
};
if (((n-2) % 2) == 1)
  a[n-1] += a[n-2]*a[n]
    
```

52

## Trasformazioni per aggregazione : Trasformazioni dei cicli – loop fission

- //loop fission converte un ciclo avente un corpo composto in diversi cicli aventi lo stesso spazio di iterazione

```

for(i=1,i<=n,i++) {
  a[i] += c;
  x[i+i]=d+x[i+1]*a[i]
}

for(i=1,i<=n,i++)
  a[i] += c;
for(i=1,i<=n,i++)
  x[i+i]=d+x[i+1]*a[i]
    
```

53

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

54

## Trasformazioni dei dati

- oscurano le strutture dati usate nell'applicazione sorgente.
- sono classificate in
  - Storage
  - Encoding
  - Aggregation
  - Ordering .

55

## Trasformazioni Storage e Encoding

- In molti casi c'è
  - un naturale modo di immagazzinare un particolare oggetto in un programma
    - Es. `int i; a[i]=3;`
  - una naturale interpretazione del pattern di bit che una particolare variabile può possedere
    - Se i contiene `000000000001100`, questo sarà interpretato come 12.
- Le trasformazioni storage tentano di scegliere una classe di immagazzinamento non naturale sia per i dati dinamici che per quelli statici
- le trasformazioni encoding tentano di scegliere codifiche non naturali per i comuni tipi di dati.

56

## Trasformazioni Storage e Encoding : Cambiare la codifica

Come semplice esempio rimpiazzeremo una variabile intera i da con  $i=c1*i+c2$ , dove c1 e c2 sono costanti.

```

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

 $\Rightarrow$ 

```

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

Può essere facilmente deoffuscato usando le comuni tecniche di analisi dei compilatori

57

## Trasformazioni Storage e Encoding : Promozione di variabili

- ❖ Esistono trasformazioni del tipo di memorizzazione dei dati che promuovono le variabili da una classe di memorizzazione ad una più generale
- ❖ in Java, una variabile integer può essere promossa ad un oggetto integer

```

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

 $\Rightarrow$ 

```

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

- ❖ E' anche possibile cambiare la durata del ciclo di vita di una variabile

```

void P() {
  int i; ... i ...
}
void Q() {
  int k; ... k ...
}
    
```

 $\Rightarrow$ 

```

int C;
void P() {
  ... C ...
}
void Q() {
  ... C ...
}
    
```

58

## Trasformazioni Storage e Encoding : Splitting delle Variabili (1)

- ❖ Le variabili aventi raggio d'azione ridotto possono essere spezzate in due o più variabili.
  - ❖ una variabile V che è spezzata in k variabili  $p_1, p_2, \dots, p_k$  si indica con  $V=[p_1, p_2, \dots, p_k]$
- ❖ La potenza, la resilienza, e il costo di questa trasformazione crescono con k
  - ❖ Purtroppo i valori di k utilizzabili sono 2 e 3.
- ❖ Per permettere ad una variabile V di tipo T di essere spezzata in due variabili p e q di tipo U, è necessario fornire 3 informazioni:
  - ❖ una funzione  $f(p, q) = V$
  - ❖ una funzione  $g(V)$
  - ❖ nuove operazioni che possono essere effettuate su p e q.

59

## Trasformazioni Storage e Encoding : Splitting delle Variabili (2)

- ❖ ci sono diversi possibili modi di computare la stessa espressione booleana
- ❖ La resilienza può essere ulteriormente migliorata selezionando la codifica a run-time.

$g(V)$	$f(p, q)$	$2p+q$	$VAL(p, q)$	$P$	$AND(A, B)$	$A$	$OR(A, B)$	$A$
0 0	False	0	0 1	B	0 0 1 0 1 0	0	0 1 2 3	0 1 2 3
0 1	True	1	0 1 1 0	B	1 3 1 1 2 3	B	1 1 1 1 2 2	1 1 2 2 3 3
1 0	True	2	1 1 1 0	B	2 0 2 1 1 3	2	2 2 2 1 1 1	2 2 2 1 1 1
1 1	False	3	1 1 1 0	B	3 3 0 0 1 3	3	0 1 1 2 1 0	3 0 1 1 2 1 0
	(a)		(b)		(c)		(d)	

```

(1) bool A, B, C;
(2) A = True;
(3) B = False;
(4) C = False;
(5) C = A & B;
(6) C = A | B;
(7) C = A | B;
(8) if (A) ...;
(9) if (B) ...;
(10) if (C) ...;
    
```

 $\Rightarrow$ 

```

(1') short a1, a2, b1, b2, c1, c2;
(2') a1=0; a2=1;
(3') b1=0; b2=0;
(4') c1=1; c2=1;
(5') x=AND[2*a1+a2, 2*b1+b2]; c1=x/2; c2=x%2;
(6') c1=(a1 & a2) & (b1 & b2); c2=0;
(7') x=OR[2*a1+a2, 2*b1+b2]; c1=x/2; c2=x%2;
(8') x=2*a1+a2; if ((x&1) | (x==2)) ...;
(9') if (b1 & b2) ...;
(10') if (VAL[c1, c2]) ...;
    
```

60

## Trasformazioni Storage e Encoding :

### Convertire dati statici in dati procedurali

- I dati statici, particolarmente stringhe di caratteri, contengono molte informazioni utili per il reverse engineer
- Idea : convertire una stringa statica in un programma che produce le stringhe (DFA)

```
String G (int n) {
    int i=0,k;
    String S;
    while (1) {
        L1: if (n==1) {S[i++]="A";k=0;goto L6;}
        L2: if (n==2) {S[i++]="B";k=2;goto L6;}
        L3: if (n==3) {S[i++]="C";goto L9;}
        L4: if (n==4) {S[i++]="X";goto L9;}
        L5: if (n==5) {S[i++]="O";goto L11;}
            if (n>12) goto L1;
        L6: if (k++<2) {S[i++]="A";goto L6} else goto L8;
        L8: return S;
        L9: S[i++]="C"; goto L10;
        L10: S[i++]="B"; goto L8;
        L11: S[i++]="O"; goto L12;
        L12: goto L10;
    }
}
```

- G offusca le stringhe "AAA", "BAAAA", e "CCB".
- G(1) = "AAA"
- G(2) = "BAAAA"
- G(3) = G(5) = "CCB"
- G(4) = "XCB"

## Trasformazioni per Aggregazione

- in un programma object oriented, il controllo è organizzato attorno alle strutture dati
  - array , oggetti.
- un offuscatore deve provare a nascondere queste strutture dati
  - Fusioni di variabili scalari
  - Ristrutturazione di Array
  - Modificare le relazioni di ereditarietà

## Trasformazioni per Aggregazione :

### Fusioni di variabili scalari (1)

- Due o più variabili scalari  $V_1, \dots, V_k$  possono essere fuse in una variabile  $V_n$ 
  - L'aritmetica sulle variabili individuali deve essere trasformata in una aritmetica su  $V_n$
- Es. fusione di due variabili integer a 32 bit X e Y in una variabile Z a 64 bit usando la funzione  $Z(X,Y) = 2^{32} * Y + X$

$$\begin{aligned}
 Z(X+r, Y) &= 2^{32} \cdot Y + (r + X) = Z(X, Y) + r \\
 Z(X, Y+r) &= 2^{32} \cdot (Y+r) + X = Z(X, Y) + r \cdot 2^{32} \\
 Z(X \cdot r, Y) &= 2^{32} \cdot Y + X \cdot r = Z(X, Y) + (r-1) \cdot X \\
 Z(X, Y \cdot r) &= 2^{32} \cdot Y \cdot r + X = Z(X, Y) + (r-1) \cdot 2^{32} \cdot Y
 \end{aligned}$$

## Trasformazioni per Aggregazione :

### Fusioni di variabili scalari (2)

- La resilienza è veramente bassa
  - Basta solo esaminare l'insieme di operazioni aritmetiche applicate ad una particolare variabile
  - Può aumentare introducendo finte operazioni
    - Es. if ( $P^F$ ) Z = rotate(Z,5)

```
(1) int X=45, Y=95;
(2) X += 5;
(3) Y += 11;
(4) X *= c;
(5) Y += d;

(1') long Z=167759066119551045;
(2') Z += 5;
(3') Z += 47244640256;
(4') Z += (c-1)*(Z & 4294967295);
(5') Z += (d-1)*(Z & 18446744069414584320);
```

## Trasformazioni per Aggregazione :

### Ristrutturare Array

Varie trasformazioni possono essere create per oscurare operazioni eseguite su array : splitting(1-2), merging(3-5), folding(6-7), flatting(8-9)

```
(1) int A[10];
(2) A[1] = ...;
...
(3) int B[10], C[10];
(4) B[1] = ...;
(5) C[1] = ...;
...
(6) int D[1, 4];
(7) for(i=0; i<=4; i++)
    D[i] = B[i+1];
...
(8) int E[2, 2];
(9) for(i=0; i<=1; i++)
    for(j=0; j<=1; j++)
        E[i][j] = B[i][j+1];
...
(10) int F[1, 13];
(11) F[0] = ...;
...
(12) int G[1, 13];
(13) G[0] = ...;
...
(14) int H[1, 13];
(15) H[0] = ...;
...
(16) int I[1, 13];
(17) I[0] = ...;
...
(18) int J[1, 13];
(19) J[0] = ...;
...
(20) int K[1, 13];
(21) K[0] = ...;
...
(22) int L[1, 13];
(23) L[0] = ...;
...
(24) int M[1, 13];
(25) M[0] = ...;
...
(26) int N[1, 13];
(27) N[0] = ...;
...
(28) int O[1, 13];
(29) O[0] = ...;
...
(30) int P[1, 13];
(31) P[0] = ...;
...
(32) int Q[1, 13];
(33) Q[0] = ...;
...
(34) int R[1, 13];
(35) R[0] = ...;
...
(36) int S[1, 13];
(37) S[0] = ...;
...
(38) int T[1, 13];
(39) T[0] = ...;
...
(40) int U[1, 13];
(41) U[0] = ...;
...
(42) int V[1, 13];
(43) V[0] = ...;
...
(44) int W[1, 13];
(45) W[0] = ...;
...
(46) int X[1, 13];
(47) X[0] = ...;
...
(48) int Y[1, 13];
(49) Y[0] = ...;
...
(50) int Z[1, 13];
(51) Z[0] = ...;
...
(52) int AA[1, 13];
(53) AA[0] = ...;
...
(54) int AB[1, 13];
(55) AB[0] = ...;
...
(56) int AC[1, 13];
(57) AC[0] = ...;
...
(58) int AD[1, 13];
(59) AD[0] = ...;
...
(60) int AE[1, 13];
(61) AE[0] = ...;
...
(62) int AF[1, 13];
(63) AF[0] = ...;
...
(64) int AG[1, 13];
(65) AG[0] = ...;
...
(66) int AH[1, 13];
(67) AH[0] = ...;
...
(68) int AI[1, 13];
(69) AI[0] = ...;
...
(70) int AJ[1, 13];
(71) AJ[0] = ...;
...
(72) int AK[1, 13];
(73) AK[0] = ...;
...
(74) int AL[1, 13];
(75) AL[0] = ...;
...
(76) int AM[1, 13];
(77) AM[0] = ...;
...
(78) int AN[1, 13];
(79) AN[0] = ...;
...
(80) int AO[1, 13];
(81) AO[0] = ...;
...
(82) int AP[1, 13];
(83) AP[0] = ...;
...
(84) int AQ[1, 13];
(85) AQ[0] = ...;
...
(86) int AR[1, 13];
(87) AR[0] = ...;
...
(88) int AS[1, 13];
(89) AS[0] = ...;
...
(90) int AT[1, 13];
(91) AT[0] = ...;
...
(92) int AU[1, 13];
(93) AU[0] = ...;
...
(94) int AV[1, 13];
(95) AV[0] = ...;
...
(96) int AW[1, 13];
(97) AW[0] = ...;
...
(98) int AX[1, 13];
(99) AX[0] = ...;
...
(100) int AY[1, 13];
(101) AY[0] = ...;
...
(102) int AZ[1, 13];
(103) AZ[0] = ...;
...
(104) int BA[1, 13];
(105) BA[0] = ...;
...
(106) int BB[1, 13];
(107) BB[0] = ...;
...
(108) int BC[1, 13];
(109) BC[0] = ...;
...
(110) int BD[1, 13];
(111) BD[0] = ...;
...
(112) int BE[1, 13];
(113) BE[0] = ...;
...
(114) int BF[1, 13];
(115) BF[0] = ...;
...
(116) int BG[1, 13];
(117) BG[0] = ...;
...
(118) int BH[1, 13];
(119) BH[0] = ...;
...
(120) int BI[1, 13];
(121) BI[0] = ...;
...
(122) int BJ[1, 13];
(123) BJ[0] = ...;
...
(124) int BK[1, 13];
(125) BK[0] = ...;
...
(126) int BL[1, 13];
(127) BL[0] = ...;
...
(128) int BM[1, 13];
(129) BM[0] = ...;
...
(130) int BN[1, 13];
(131) BN[0] = ...;
...
(132) int BO[1, 13];
(133) BO[0] = ...;
...
(134) int BP[1, 13];
(135) BP[0] = ...;
...
(136) int BQ[1, 13];
(137) BQ[0] = ...;
...
(138) int BR[1, 13];
(139) BR[0] = ...;
...
(140) int BS[1, 13];
(141) BS[0] = ...;
...
(142) int BT[1, 13];
(143) BT[0] = ...;
...
(144) int BU[1, 13];
(145) BU[0] = ...;
...
(146) int BV[1, 13];
(147) BV[0] = ...;
...
(148) int BW[1, 13];
(149) BW[0] = ...;
...
(150) int BX[1, 13];
(151) BX[0] = ...;
...
(152) int BY[1, 13];
(153) BY[0] = ...;
...
(154) int BZ[1, 13];
(155) BZ[0] = ...;
...
(156) int CA[1, 13];
(157) CA[0] = ...;
...
(158) int CB[1, 13];
(159) CB[0] = ...;
...
(160) int CC[1, 13];
(161) CC[0] = ...;
...
(162) int CD[1, 13];
(163) CD[0] = ...;
...
(164) int CE[1, 13];
(165) CE[0] = ...;
...
(166) int CF[1, 13];
(167) CF[0] = ...;
...
(168) int CG[1, 13];
(169) CG[0] = ...;
...
(170) int CH[1, 13];
(171) CH[0] = ...;
...
(172) int CI[1, 13];
(173) CI[0] = ...;
...
(174) int CJ[1, 13];
(175) CJ[0] = ...;
...
(176) int CK[1, 13];
(177) CK[0] = ...;
...
(178) int CL[1, 13];
(179) CL[0] = ...;
...
(180) int CM[1, 13];
(181) CM[0] = ...;
...
(182) int CN[1, 13];
(183) CN[0] = ...;
...
(184) int CO[1, 13];
(185) CO[0] = ...;
...
(186) int CP[1, 13];
(187) CP[0] = ...;
...
(188) int CQ[1, 13];
(189) CQ[0] = ...;
...
(190) int CR[1, 13];
(191) CR[0] = ...;
...
(192) int CS[1, 13];
(193) CS[0] = ...;
...
(194) int CT[1, 13];
(195) CT[0] = ...;
...
(196) int CU[1, 13];
(197) CU[0] = ...;
...
(198) int CV[1, 13];
(199) CV[0] = ...;
...
(200) int CW[1, 13];
(201) CW[0] = ...;
...
(202) int CX[1, 13];
(203) CX[0] = ...;
...
(204) int CY[1, 13];
(205) CY[0] = ...;
...
(206) int CZ[1, 13];
(207) CZ[0] = ...;
...
(208) int DA[1, 13];
(209) DA[0] = ...;
...
(210) int DB[1, 13];
(211) DB[0] = ...;
...
(212) int DC[1, 13];
(213) DC[0] = ...;
...
(214) int DD[1, 13];
(215) DD[0] = ...;
...
(216) int DE[1, 13];
(217) DE[0] = ...;
...
(218) int DF[1, 13];
(219) DF[0] = ...;
...
(220) int DG[1, 13];
(221) DG[0] = ...;
...
(222) int DH[1, 13];
(223) DH[0] = ...;
...
(224) int DI[1, 13];
(225) DI[0] = ...;
...
(226) int DJ[1, 13];
(227) DJ[0] = ...;
...
(228) int DK[1, 13];
(229) DK[0] = ...;
...
(230) int DL[1, 13];
(231) DL[0] = ...;
...
(232) int DM[1, 13];
(233) DM[0] = ...;
...
(234) int DN[1, 13];
(235) DN[0] = ...;
...
(236) int DO[1, 13];
(237) DO[0] = ...;
...
(238) int DP[1, 13];
(239) DP[0] = ...;
...
(240) int DQ[1, 13];
(241) DQ[0] = ...;
...
(242) int DR[1, 13];
(243) DR[0] = ...;
...
(244) int DS[1, 13];
(245) DS[0] = ...;
...
(246) int DT[1, 13];
(247) DT[0] = ...;
...
(248) int DU[1, 13];
(249) DU[0] = ...;
...
(250) int DV[1, 13];
(251) DV[0] = ...;
...
(252) int DW[1, 13];
(253) DW[0] = ...;
...
(254) int DX[1, 13];
(255) DX[0] = ...;
...
(256) int DY[1, 13];
(257) DY[0] = ...;
...
(258) int DZ[1, 13];
(259) DZ[0] = ...;
...
(260) int EA[1, 13];
(261) EA[0] = ...;
...
(262) int EB[1, 13];
(263) EB[0] = ...;
...
(264) int EC[1, 13];
(265) EC[0] = ...;
...
(266) int ED[1, 13];
(267) ED[0] = ...;
...
(268) int EE[1, 13];
(269) EE[0] = ...;
...
(270) int EF[1, 13];
(271) EF[0] = ...;
...
(272) int EG[1, 13];
(273) EG[0] = ...;
...
(274) int EH[1, 13];
(275) EH[0] = ...;
...
(276) int EI[1, 13];
(277) EI[0] = ...;
...
(278) int EJ[1, 13];
(279) EJ[0] = ...;
...
(280) int EK[1, 13];
(281) EK[0] = ...;
...
(282) int EL[1, 13];
(283) EL[0] = ...;
...
(284) int EM[1, 13];
(285) EM[0] = ...;
...
(286) int EN[1, 13];
(287) EN[0] = ...;
...
(288) int EO[1, 13];
(289) EO[0] = ...;
...
(290) int EP[1, 13];
(291) EP[0] = ...;
...
(292) int EQ[1, 13];
(293) EQ[0] = ...;
...
(294) int ER[1, 13];
(295) ER[0] = ...;
...
(296) int ES[1, 13];
(297) ES[0] = ...;
...
(298) int ET[1, 13];
(299) ET[0] = ...;
...
(300) int EU[1, 13];
(301) EU[0] = ...;
...
(302) int EV[1, 13];
(303) EV[0] = ...;
...
(304) int EW[1, 13];
(305) EW[0] = ...;
...
(306) int EX[1, 13];
(307) EX[0] = ...;
...
(308) int EY[1, 13];
(309) EY[0] = ...;
...
(310) int EZ[1, 13];
(311) EZ[0] = ...;
...
(312) int FA[1, 13];
(313) FA[0] = ...;
...
(314) int FB[1, 13];
(315) FB[0] = ...;
...
(316) int FC[1, 13];
(317) FC[0] = ...;
...
(318) int FD[1, 13];
(319) FD[0] = ...;
...
(320) int FE[1, 13];
(321) FE[0] = ...;
...
(322) int FF[1, 13];
(323) FF[0] = ...;
...
(324) int FG[1, 13];
(325) FG[0] = ...;
...
(326) int FH[1, 13];
(327) FH[0] = ...;
...
(328) int FI[1, 13];
(329) FI[0] = ...;
...
(330) int FJ[1, 13];
(331) FJ[0] = ...;
...
(332) int FK[1, 13];
(333) FK[0] = ...;
...
(334) int FL[1, 13];
(335) FL[0] = ...;
...
(336) int FM[1, 13];
(337) FM[0] = ...;
...
(338) int FN[1, 13];
(339) FN[0] = ...;
...
(340) int FO[1, 13];
(341) FO[0] = ...;
...
(342) int FP[1, 13];
(343) FP[0] = ...;
...
(344) int FQ[1, 13];
(345) FQ[0] = ...;
...
(346) int FR[1, 13];
(347) FR[0] = ...;
...
(348) int FS[1, 13];
(349) FS[0] = ...;
...
(350) int FT[1, 13];
(351) FT[0] = ...;
...
(352) int FU[1, 13];
(353) FU[0] = ...;
...
(354) int FV[1, 13];
(355) FV[0] = ...;
...
(356) int FW[1, 13];
(357) FW[0] = ...;
...
(358) int FX[1, 13];
(359) FX[0] = ...;
...
(360) int FY[1, 13];
(361) FY[0] = ...;
...
(362) int FZ[1, 13];
(363) FZ[0] = ...;
...
(364) int GA[1, 13];
(365) GA[0] = ...;
...
(366) int GB[1, 13];
(367) GB[0] = ...;
...
(368) int GC[1, 13];
(369) GC[0] = ...;
...
(370) int GD[1, 13];
(371) GD[0] = ...;
...
(372) int GE[1, 13];
(373) GE[0] = ...;
...
(374) int GF[1, 13];
(375) GF[0] = ...;
...
(376) int GG[1, 13];
(377) GG[0] = ...;
...
(378) int GH[1, 13];
(379) GH[0] = ...;
...
(380) int GI[1, 13];
(381) GI[0] = ...;
...
(382) int GJ[1, 13];
(383) GJ[0] = ...;
...
(384) int GK[1, 13];
(385) GK[0] = ...;
...
(386) int GL[1, 13];
(387) GL[0] = ...;
...
(388) int GM[1, 13];
(389) GM[0] = ...;
...
(390) int GN[1, 13];
(391) GN[0] = ...;
...
(392) int GO[1, 13];
(393) GO[0] = ...;
...
(394) int GP[1, 13];
(395) GP[0] = ...;
...
(396) int GQ[1, 13];
(397) GQ[0] = ...;
...
(398) int GR[1, 13];
(399) GR[0] = ...;
...
(400) int GS[1, 13];
(401) GS[0] = ...;
...
(402) int GT[1, 13];
(403) GT[0] = ...;
...
(404) int GU[1, 13];
(405) GU[0] = ...;
...
(406) int GV[1, 13];
(407) GV[0] = ...;
...
(408) int GW[1, 13];
(409) GW[0] = ...;
...
(410) int GX[1, 13];
(411) GX[0] = ...;
...
(412) int GY[1, 13];
(413) GY[0] = ...;
...
(414) int GZ[1, 13];
(415) GZ[0] = ...;
...
(416) int HA[1, 13];
(417) HA[0] = ...;
...
(418) int HB[1, 13];
(419) HB[0] = ...;
...
(420) int HC[1, 13];
(421) HC[0] = ...;
...
(422) int HD[1, 13];
(423) HD[0] = ...;
...
(424) int HE[1, 13];
(425) HE[0] = ...;
...
(426) int HF[1, 13];
(427) HF[0] = ...;
...
(428) int HG[1, 13];
(429) HG[0] = ...;
...
(430) int HH[1, 13];
(431) HH[0] = ...;
...
(432) int HI[1, 13];
(433) HI[0] = ...;
...
(434) int HJ[1, 13];
(435) HJ[0] = ...;
...
(436) int HK[1, 13];
(437) HK[0] = ...;
...
(438) int HL[1, 13];
(439) HL[0] = ...;
...
(440) int HM[1, 13];
(441) HM[0] = ...;
...
(442) int HN[1, 13];
(443) HN[0] = ...;
...
(444) int HO[1, 13];
(445) HO[0] = ...;
...
(446) int HP[1, 13];
(447) HP[0] = ...;
...
(448) int HQ[1, 13];
(449) HQ[0] = ...;
...
(450) int HR[1, 13];
(451) HR[0] = ...;
...
(452) int HS[1, 13];
(453) HS[0] = ...;
...
(454) int HT[1, 13];
(455) HT[0] = ...;
...
(456) int HU[1, 13];
(457) HU[0] = ...;
...
(458) int HV[1, 13];
(459) HV[0] = ...;
...
(460) int HW[1, 13];
(461) HW[0] = ...;
...
(462) int HX[1, 13];
(463) HX[0] = ...;
...
(464) int HY[1, 13];
(465) HY[0] = ...;
...
(466) int HZ[1, 13];
(467) HZ[0] = ...;
...
(468) int IA[1, 13];
(469) IA[0] = ...;
...
(470) int IB[1, 13];
(471) IB[0] = ...;
...
(472) int IC[1, 13];
(473) IC[0] = ...;
...
(474) int ID[1, 13];
(475) ID[0] = ...;
...
(476) int IE[1, 13];
(477) IE[0] = ...;
...
(478) int IF[1, 13];
(479) IF[0] = ...;
...
(480) int IG[1, 13];
(481) IG[0] = ...;
...
(482) int IH[1, 13];
(483) IH[0] = ...;
...
(484) int II[1, 13];
(485) II[0] = ...;
...
(486) int IJ[1, 13];
(487) IJ[0] = ...;
...
(488) int IK[1, 13];
(489) IK[0] = ...;
...
(490) int IL[1, 13];
(491) IL[0] = ...;
...
(492) int IM[1, 13];
(493) IM[0] = ...;
...
(494) int IN[1, 13];
(495) IN[0] = ...;
...
(496) int IO[1, 13];
(497) IO[0] = ...;
...
(498) int IP[1, 13];
(499) IP[0] = ...;
...
(500) int IQ[1, 13];
(501) IQ[0] = ...;
...
(502) int IR[1, 13];
(503) IR[0] = ...;
...
(504) int IS[1, 13];
(505) IS[0] = ...;
...
(506) int IT[1, 13];
(507) IT[0] = ...;
...
(508) int IU[1, 13];
(509) IU[0] = ...;
...
(510) int IV[1, 13];
(511) IV[0] = ...;
...
(512) int IW[1, 13];
(513) IW[0] = ...;
...
(514) int IX[1, 13];
(515) IX[0] = ...;
...
(516) int IY[1, 13];
(517) IY[0] = ...;
...
(518) int IZ[1, 13];
(519) IZ[0] = ...;
...
(520) int JA[1, 13];
(521) JA[0] = ...;
...
(522) int JB[1, 13];
(523) JB[0] = ...;
...
(524) int JC[1, 13];
(525) JC[0] = ...;
...
(526) int JD[1, 13];
(527) JD[0] = ...;
...
(528) int JE[1, 13];
(529) JE[0] = ...;
...
(530) int JF[1, 13];
(531) JF[0] = ...;
...
(532) int JG[1, 13];
(533) JG[0] = ...;
...
(534) int JH[1, 13];
(535) JH[0] = ...;
...
(536) int JI[1, 13];
(537) JI[0] = ...;
...
(538) int JJ[1, 13];
(539) JJ[0] = ...;
...
(540) int JK[1, 13];
(541) JK[0] = ...;
...
(542) int JL[1, 13];
(543) JL[0] = ...;
...
(544) int JM[1, 13];
(545) JM[0] = ...;
...
(546) int JN[1, 13];
(547) JN[0] = ...;
...
(548) int JO[1, 13];
(549) JO[0] = ...;
...
(550) int JP[1, 13];
(551) JP[0] = ...;
...
(552) int JQ[1, 13];
(553) JQ[0] = ...;
...
(554) int JR[1, 13];
(555) JR[0] = ...;
...
(556) int JS[1, 13];
(557) JS[0] = ...;
...
(558) int JT[1, 13];
(559) JT[0] = ...;
...
(560) int JU[1, 13];
(561) JU[0] = ...;
...
(562) int JV[1, 13];
(563) JV[0] = ...;
...
(564) int JW[1, 13];
(565) JW[0] = ...;
...
(566) int JX[1, 13];
(567) JX[0] = ...;
...
(568) int JY[1, 13];
(569) JY[0] = ...;
...
(570) int JZ[1, 13];
(571) JZ[0] = ...;
...
(572) int KA[1, 13];
(573) KA[0] = ...;
...
(574) int KB[1, 13];
(575) KB[0] = ...;
...
(576) int KC[1, 13];
(577) KC[0] = ...;
...
(578) int KD[1, 13];
(579) KD[0] = ...;
...
(580) int KE[1, 13];
(581) KE[0] = ...;
...
(582) int KF[1, 13];
(583) KF[0] = ...;
...
(584) int KG[1, 13];
(585) KG[0] = ...;
...
(586) int KH[1, 13];
(587) KH[0] = ...;
...
(588) int KI[1, 13];
(589) KI[0] = ...;
...
(590) int KJ[1, 13];
(591) KJ[0] = ...;
...
(592) int KK[1, 13];
(593) KK[0] = ...;
...
(594) int KL[1, 13];
(595) KL[0] = ...;
...
(596) int KM[1, 13];
(597) KM[0] = ...;
...
(598) int KN[1, 13];
(599) KN[0] = ...;
...
(600) int KO[1, 13];
(601) KO[0] = ...;
...
(602) int KP[1, 13];
(603) KP[0] = ...;
...
(604) int KQ[1, 13];
(605) KQ[0] = ...;
...
(606) int KR[1, 13];
(607) KR[0] = ...;
...
(608) int KS[1, 13];
(609) KS[0] = ...;
...
(610) int KT[1, 13];
(611) KT[0] = ...;
...
(612) int KU[1, 13];
(613) KU[0] = ...;
...
(614) int KV[1, 13];
(615) KV[0] = ...;
...
(616) int KW[1, 13];
(617) KW[0] = ...;
...
(618) int KX[1, 13];
(619) KX[0] = ...;
...
(620) int KY[1, 13];
(621) KY[0] = ...;
...
(622) int KZ[1, 13];
(623) KZ[0] = ...;
...
(624) int LA[1, 13];
(625) LA[0] = ...;
...
(626) int LB[1, 13];
(627) LB[0] = ...;
...
(628) int LC[1, 13];
(629) LC[0] = ...;
...
(630) int LD[1, 13];
(631) LD[0] = ...;
...
(632) int LE[1, 13];
(633) LE[0] = ...;
...
(634) int LF[1, 13];
(635) LF[0] = ...;
...
(636) int LG[1, 13];
(637) LG[0] = ...;
...
(638) int LH[1, 13];
(639) LH[0] = ...;
...
(640) int LI[1, 13];
(641) LI[0] = ...;
...
(642) int LJ[1, 13];
(643) LJ[0] = ...;
...
(644) int LK[1, 13];
(645) LK[0] = ...;
...
(646) int LL[1, 13];
(647) LL[0] = ...;
...
(648) int LM[1, 13];
(649) LM[0] = ...;
...
(650) int LN[1, 13];
(651) LN[0] = ...;
...
(652) int LO[1, 13];
(653) LO[0] = ...;
...
(654) int LP[1, 13];
(655) LP[0] = ...;
...
(656) int LQ[1, 13];
(657) LQ[0] = ...;
...
(658) int LR[1, 13];
(659) LR[0] = ...;
...
(660) int LS[1, 13];
(661) LS[0] = ...;
...
(662) int LT[1, 13];
(663) LT[0] = ...;
...
(664) int LU[1, 13];
(665) LU[0] = ...;
...
(666) int LV[1, 13];
(667) LV[0] = ...;
...
(668) int LW[1, 13];
(669) LW[0] = ...;
...
(670) int LX[1, 13];
(671) LX[0] = ...;
...
(672) int LY[1, 13];
(673) LY[0] = ...;
...
(674) int LZ[1, 13];
(675) LZ[0] = ...;
...
(676) int MA[1, 13];
(677) MA[0] = ...;
...
(678) int MB[1, 13];
(679) MB[0] = ...;
...
(680) int MC[1, 13];
(681) MC[0] = ...;
...
(682) int MD[1, 13];
(683) MD[0] = ...;
...
(684) int ME[1, 13];
(685) ME[0] = ...;
...
(686) int MF[1, 13];
(687) MF[0] = ...;
...
(688) int MG[1, 13];
(689) MG[0] = ...;
...
(690) int MH[1, 13];
(691) MH[0] = ...;
...
(692) int MI[1, 13];
(693)
```

## Trasformazioni dell'ordine

- randomizzare l'ordine con cui le computazioni sono svolte e l'ordine delle dichiarazioni delle variabili è un utile offuscamento
- La potenza è bassa ma la resilienza è one-way.
- possibile riordinare gli elementi all'interno di un array una funzione  $f(i)$

```

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

```

 $\xrightarrow{T}$ 

```

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

```

67

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

68

## Valori e predicati opachi

- Costrutti opachi che usano oggetti e alias
- Costrutti opachi che usano i thread

69

## Valori e predicati opachi

- vorremmo essere capaci di costruire predicati opachi che richiedano nel caso peggiore un tempo esponenziale (nella dimensione del programma) per romperli e tempo polinomiale per costruirli
- Ci sono due di tali tecniche.
  - Una basata sugli alias
  - L'altra sui thread.

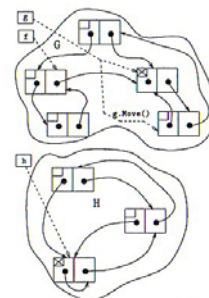
70

## Valori e predicati opachi : Uso di alias (1)

- L'aliasing complica significativamente l'analisi statica interprocedurale
  - L'analisi diventa un problema NP-hard o addirittura indecidibile
  - idea : costruire una struttura dinamica complessa e mantenere un insieme di puntatori in questa struttura

71

## Valori e predicati opachi : Uso di alias (2)



```

Node g, h;
method P(..., Node f) {
  /* 1 */ g = g.Move();
  h = h.Move();
  /* 2 */ h = h.Insert(new Node);
  ...
  /* 3 */ x.R(..., f.Move());
  ...
  /* 4 */ if (f == g) ...
  /* 5 */ if (g == h) ...
  ...
  /* 6 */ f.Token=False;
  g.Token=True;
  /* 7 */ if (f.Token) ...
  ...
  /* 8 */ f.Token=True;
  h.Token=False;
  /* 9 */ if (f.Token) ...
}

```

72

## Valori e predicati opachi : Uso dei thread (1)

- I programmi paralleli sono molto più difficili da analizzare rispetto alla loro controparte sequenziale.
  - semantica interferente : n espressioni in una regione parallela (thread)
  - Molte analisi devono considerare n! interferimenti
- Idea base analoga a quella che usa gli alias
  - struttura dati globale V aggiornato da thread
- Vantaggi
  - i predicati opachi richiedono nel caso peggiore tempo esponenziale per essere rotti
  - grado molto alto di resilienza
    - Si combinano il data race con gli effetti di interferimento e aliasing,

73

## Valori e predicati opachi : Uso dei thread (2)

```

thread S {
  int R;
  while (1) {
    R = random(1,C);
    X = R*R;
    sleep(3);
  }
}

thread T {
  int R;
  while (1) {
    R = random(1,C);
    Y = 7*R*R;
    sleep(2);
    X *= X;
    sleep(5);
  }
}

int X, Y;
const C = sqrt(maxint)/10;
main () {
  S.run(); T.run();
  ...
  if ((Y - 1) == X) P
  ...
}
    
```

74

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

75

## Deoffuscamento e trasformazioni preventive

- Trasformazioni preventive
- Identificare e valutare i costrutti opachi
- Identificazione tramite Pattern Matching
- Identificazione tramite decomposizione (slicing) del programma
- Analisi Statistica
- Valutazione attraverso il flusso dei dati
- Valutazione attraverso verifica di teoremi
- Deoffuscamento e valutazione parziale

76

## Deoffuscamento e trasformazioni preventive

- Un deoffuscatore deve esaminare l'applicazione offuscata e automaticamente identificare e rimuovere i falsi programmi in esso
  - il deoffuscatore deve prima identificare e valutare i costrutti opachi
- trasformazioni preventive sono contromisure che un offuscatore può impiegare per rendere il deoffuscamento più complicato.

```

{
  if (P1) { S1; }
  else { S2; }
}

{
  if (Q1) { S1; }
  else { S2; }
}

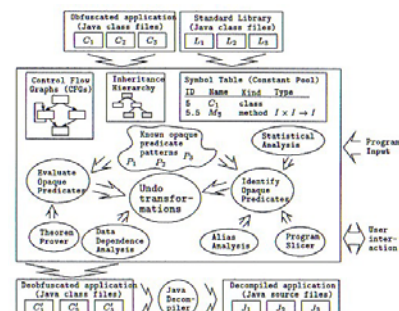
{
  if (P1) { S1; }
  else { S2; }
}

{
  if (True) { S1; }
  else { S2; }
}

{
  if (P1) { S1; }
  else { S2; }
}
    
```

77

## Deoffuscamento e trasformazioni preventive : anatomia di un tool di deoffuscamento



78

## Trasformazioni preventive

- Sono progettate per rendere le tecniche di deoffuscamento conosciute più difficili
  - **trasformazioni preventive inerenti**
  - **trasformazioni preventive mirate**

79

## Trasformazioni preventive inerenti

- Generalmente hanno poca potenza e molta resilienza.
  - esse hanno la capacità di aumentare la resilienza di altre trasformazioni
- Es. riordiniamo un ciclo for per farlo girare all'indietro
- Non c'è dipendenza dai dati
- Problema : Il deoffuscatore può comunque invertire la trasformazione
- Soluzione : aggiungere una falsa dipendenza dei dati

```
for(i=1;i<=10;i++)  
A[i]=i  
└──┬──  
for(i=10;i>=1;i--){  
A[i]=i;  
for(i=1;i<=10;i++)  
A[i]=i  
└──┬──  
int B[50];  
for(i=10;i>=1;i--){  
A[i]=i;  
B[i]*=B[i+1/2]
```

80

## Trasformazioni preventive mirate

- HoseMocha progettato specificatamente per indagare sulle debolezze del decompilatore Mocha
- HoseMocha inserisce istruzioni extra dopo ogni espressione return in ogni metodo del programma sorgente
- Mocha va in crush

81

## Identificare e valutare i costrutti opachi

- E' la parte più difficile del deoffuscamento
- Un costrutto opaco può essere :
  - **Locale** (contenuto in un singolo basic block)
  - **globale** (contenuto in una singola procedura)
  - **interprocedurale** (distribuito lungo tutto il programma)

82

## Identificazione tramite Pattern Matching

- Si esamina un offuscatore e si costruiscono regole di **pattern-matching** che possano identificare i predicati opachi comunemente usati
- l'offuscatore dovrebbe evitare di usare predicati opachi delimitati

83

## Identificazione tramite decomposizione del programma (1)

- Un tool decompone il programma in pezzi maneggevoli chiamati **slice**
- Una slice di un programma P rispetto ad un punto p e ad una variabile v consiste di tutte le espressioni di P che possono aver contribuito al valore di v nel punto p.
- Il tool deve estrarre dal programma offuscato le espressioni degli algoritmi che computano una variabile opaca v

84

## Identificazione tramite decomposizione del programma (1)

- L'offuscatore deve rendere la vita difficile allo slicer
  - Aggiungere parametri alias**
    - due parametri formali che si riferiscono sempre a qualche locazione di memoria
    - Lo slicer è rallentato o è reso impreciso
  - Aggiungere dipendenza delle variabili**
    - I popolari slicer funzionano bene per piccoli slice, ma qualche volta richiedono tempo eccessivo per computare quelli grandi.

```

main() {
  int x=1;
  x = x * 3;
}

 $\xrightarrow{T}$ 

main() {
  int x=1;
  if (P^P) x++;
  x = x + V^0;
  x = x * 3;
}
    
```

85

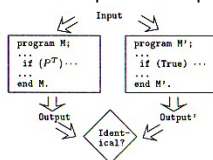
## Analisi Statistica (1)

- qualunque metodo di deoffuscamento che esamina le caratteristiche run-time di una applicazione offuscata
  - deve segnalare qualunque predicato che restituisce sempre lo stesso valore di verità durante un gran numero di test eseguiti
  - non può rimpiazzare alla cieca tali predicati con True o False
    - predicati che si comportano come predicati opachi

86

## Analisi Statistica (2)

- Si ipotizza il valore del predicato opaco potenziale identificato in M
- Si crea M' in cui il predicato opaco potenziale è rimpiazzato dal valore ipotizzato
- Si testano M e M' con gli stessi input
- Gli input devono coprire tutti i percorsi



87

## Contromisure contro l'analisi statistica

- Preferire le trasformazioni che inseriscono i predicati  $P^?$
- progettare predicati opachi in modo che diversi predicati devono essere rotti allo stesso tempo.

```

int k=0;
bool Q1(x) {
  k+=2^31; return (P1^?)
}
bool Q2(x) {
  k-=2^31; return (P2^?)
}

{ S1: ...
  S2: ...
}

 $\xrightarrow{T}$ 

{
  if (Q1(j)^?) S1;
  ...
  if (Q2(k)^?) S2;
}
    
```

Se si rimpiazza un predicato (ma non entrambi) con True, k andrà in overflow

88

## Valutazione attraverso verifica di teoremi

- Problema : Un predicato opaco può essere rotto attraverso la dimostrazione di un teorema.
- Soluzione : usare i teoremi che sappiamo essere difficili da dimostrare o che sappiamo che nessuna prova esiste.
- problema di Collatz.** Una ipotesi dice che il ciclo terminerà sempre. Sebbene non esista alcuna prova di questa ipotesi, è risaputo che il codice termina per tutti i numeri fino a  $7 * 10^{11}$

```

{
  S1;
  S2;
}

 $\xrightarrow{T}$ 

{
  S1;
  n = random(1, 2^32);
  do
    n = ((n%2)!=0)?3*n+1:n/2;
  while (n>1);
  S2;
}
    
```

89

## Sommario

- Protezione del software
- Design di un offuscatore Java
- Classificazione delle trasformazioni offuscanti
- Valutazione delle trasformazioni offuscanti
- Trasformazioni del controllo
- Trasformazioni dei dati
- Valori e predicati opachi
- Deoffuscamento e trasformazioni preventive
- Algoritmi di offuscamento

90

## Algoritmi di offuscamento

- ALGORITMO 1 (Offuscamento del codice)
- ALGORITMO 2 (SelectCode)
- ALGORITMO 3 (SelectTransform)
- ALGORITMO 4 (Done)
- ALGORITMO 5 (Informazioni pragmatiche)
- ALGORITMO 6 (Priorità dell'offuscamento)
- ALGORITMO 7 (Appropriatezza dell'offuscamento)

91

## Algoritmi di offuscamento : ciclo principale

```

WHILE NOT Done(A) DO
  S := SelectCode(A);
  T := SelectTransform(S);
  A := Apply(T,S);
END;

```

- **SelectCode** restituisce il prossimo codice oggetto sorgente che deve essere offuscato.
- **SelectTransform** restituisce la trasformazione che dovrebbe essere usata per offuscare il particolare codice oggetto sorgente.
- **Apply** applica le trasformazioni al codice oggetto sorgente
- **Done** determina quando il richiesto livello di offuscamento è stato ottenuto.

92

## Algoritmi di offuscamento : strutture dati

- Per ogni codice oggetto sorgente S e per ogni routine M
  - Ps(S) è l'insieme dei costrutti del linguaggio che il programmatore ha usato in S.
    - usato per trovare le trasformazioni offuscanti appropriate per S
  - A(S) = {T<sub>1</sub> ↦ V<sub>1</sub>, ..., T<sub>n</sub> ↦ V<sub>n</sub>} è un mapping tra le trasformazioni T<sub>i</sub> e i valori V<sub>i</sub>
  - R(M) è il rango del tempo di esecuzione di M
    - R(M) = 1 se è speso più tempo per eseguire M che le altre routine

93

## Algoritmi di offuscamento : funzioni di qualità

- Restituiscono informazioni di tipo numerico riguardanti ogni trasformazione
  - T<sub>res</sub>(S) restituisce una misura della resilienza della trasformazione T quando è applicata al codice oggetto sorgente S.
  - T<sub>pot</sub>(S) restituisce una misura della potenza della trasformazione T quando è applicata al codice oggetto sorgente S
  - T<sub>cost</sub>(S) restituisce una misura del tempo di esecuzione e dell'overhead di spazio aggiunto da T a S.
  - P<sub>t</sub> mappa ogni trasformazione T nell'insieme dei costrutti del linguaggi che T aggiungerà all'applicazione

94

## Algoritmi di offuscamento : ALGORITMO 1 (OFFUSCAMENTO DEL CODICE)

- Input
  - Un'applicazione A costituita dai file C<sub>1</sub>, C<sub>2</sub>,...
  - Le librerie standard L<sub>1</sub>, L<sub>2</sub>,...
  - {T<sub>1</sub>, T<sub>2</sub>,...}
  - P<sub>t</sub>, T<sub>pot</sub>(S), T<sub>res</sub>(S), T<sub>cost</sub>(S)
  - Un insieme di dati di input I = {I<sub>1</sub>, I<sub>2</sub>,...} ad A
  - AcceptCost > 0
    - Max overhead accettabile
  - ReqObf > 0
    - Quantità di offuscamento richiesta
- Output : Un'applicazione A' offuscata

95

## Algoritmi di offuscamento : ALGORITMO 1 (OFFUSCAMENTO DEL CODICE)

1. Caricare l'applicazione C<sub>1</sub>, C<sub>2</sub>,... da offuscare
  - a) caricare file contenenti codice sorgente, oppure
  - b) caricare file contenenti codice oggetto
2. Caricare il codice contenuto nei file delle librerie L<sub>1</sub>, L<sub>2</sub>,...
3. Costruire una rappresentazione interna dell'applicazione
  - (a) un grafo di controllo del flusso (CFG) per ogni routine A.
  - (b) un call-graph per le routine in A
  - (c) un grafo di ereditarietà per le classi in A.
4. Costruire R(M) e Ps(S) usando l'Algoritmo 5, I(S) usando l'Algoritmo 6, e A(S) usando l'Algoritmo 7.

96

### Algoritmi di offuscamento :

#### ALGORITMO 1 (OFFUSCAMENTO DEL CODICE)

5. Applicare le Trasformazioni offuscanti all'applicazione  
**REPEAT**  
T ← SelectTransform(S,A);  
Applica T ad S ed aggiorna le strutture dati rilevanti del punto 3;  
**UNTIL** Done(ReqObf, AcceptCost, S, T, I)
6. Ricostituire il codice oggetto sorgente offuscato in una nuova applicazione offuscata X

97

### Algoritmi di offuscamento :

#### ALGORITMO 2 (SelectCode)

- Input
  - Il mapping della priorità di offuscamento I come computato dall'Algoritmo 6
- Output : Un codice oggetto sorgente S
- I mappa ogni oggetto sorgente S in I(S).
  - trattiamo I come una coda a priorità
  - selezioniamo S in modo da massimizzare I(S).

98

### Algoritmi di offuscamento :

#### ALGORITMO 2 (SelectTransform)

- Input
  - a) Un codice oggetto sorgente S.
  - b) La mappa di appropriatezza computata dall'Algoritmo 7
- Output : Una trasformazione T
- Due aspetti importanti da considerare
  - T deve essere inglobata in modo naturale con il resto del codice in S.
    - Deve avere un alto valore di appropriatezza in A(S)
  - T deve rendere alti livelli di offuscamento con bassi costi di overhead

99

### Algoritmi di offuscamento :

#### ALGORITMO 2 (SelectTransform)

- Restituisci una trasformazione T tale che  
 $T \mapsto V \in A(S)$  e  
$$\frac{\omega_1 T_{pot}(S) + \omega_2 T_{res}(S) + \omega_3 V}{T_{cost}(S)}$$
 è massimizzata  
dove  $\omega_1, \omega_2, \omega_3$  sono costanti definite dall'implementazione

100

### Algoritmi di offuscamento :

#### ALGORITMO 4 (Done)

- Input
  - ReqObf, AcceptCost, S, T, I
- Output
  - a. un ReqObf aggiornato.
  - b. un AcceptCost aggiornato.
  - c. una mappa delle priorità di offuscamento I aggiornata.
  - d. un valore di ritorno booleano che è TRUE se la condizione di terminazione è stata raggiunta.

101

### Algoritmi di offuscamento :

#### ALGORITMO 4 (Done)

- Svolge vari compiti
  - Aggiorna la coda a priorità I
    - La riduzione è basata su una combinazione resilienza/potenza
  - aggiorna anche ReqObf e AcceptCost
  - determina se la condizione di terminazione è stata raggiunta  
 $I(S) \leftarrow I(S) - (\omega_1 T_{pot}(S) + \omega_2 T_{res}(S));$   
 $ReqObf \leftarrow ReqObf - (\omega_3 T_{pot}(S) + \omega_4 T_{res}(S));$   
 $AcceptCost \leftarrow AcceptCost - T_{cost}(S);$   
**RETURN** AcceptCost ≤ 0 OR ReqObf ≤ 0;

102

## Algoritmi di offuscamento :

### ALGORITMO 5 ( INFORMAZIONI PRAGMATICHE )

- Input
  - Un'applicazione A
  - $I = \{I_1, I_2, \dots\}$
- Output :  $R(M)$  ,  $P_s(S)$
- Si computano le informazioni pragmatiche
  1. dinamiche
    - Uso di profiler su I
    - Calcolare  $R(M)$  per ogni routine/basic block
  2. statiche
    - Calcolare  $P_s(S)$

103

## Algoritmi di offuscamento :

### ALGORITMO 5 ( INFORMAZIONI PRAGMATICHE )

- **FOR** S ← ogni codice oggetto sorgente in A **DO**
  - O ← l'insieme di operatori che S usa;
  - C ← l'insieme dei costrutti del linguaggio ad alto livello (WHILE , eccezioni, threads, etc.) che S usa;
  - L ← l'insieme di classi/routine di libreria che S referencia;
  - $P_s(S) \leftarrow O \cup C \cup L$ ;
- **END FOR**

104

## Algoritmi di offuscamento :

### ALGORITMO 6 (PRIORITA' DELL'OFFUSCAMENTO)

- Input
  - Un'applicazione A
  - $R(M)$
- Output :  $I(S)$
- Possibili euristiche per  $I(S)$  possono essere
  - se molto tempo è speso ad eseguire una routine M, allora M è probabilmente una procedura importante che dovrebbe essere pesantemente offuscata
  - il codice complesso è più probabile che contenga importanti segreti commerciali che semplice codice

105

## Algoritmi di offuscamento :

### ALGORITMO 7 (APPROPRIATEZZA DELL'OFFUSCAMENTO)

- Input
  - Un'applicazione A costituita dai file  $C_1, C_2, \dots$
  - $P_i, P_s(S), A(S)$
- Output :  $A(S)$
- **FOR** S ← ogni codice oggetto sorgente in A **DO**
  - FOR** T ← ogni trasformazione **DO**
    - V ← grado di somiglianza tra e ;
    - $A(S) \leftarrow A(S) \cup \{T \rightarrow V\}$ ;
- **END FOR**
- **END FOR**

106

## Altri usi dell'offuscamento

- Controllo dei pirati del software
  - Si crea una nuova versione offuscata dell'applicazione per ogni nuovo acquirente e si registrano i dati di chi ha acquistato ogni versione
    - Se si scopre una versione piratata la si confronta con il proprio database
- Uso illecito
  - un pirata potrebbe oscurare un programma Java legalmente acquistato
    - La versione offuscata potrebbe essere rivenduta
    - È un problema giuridicamente complesso
      - Il codice rivenduto è completamente differente dall'originale

107