Securing Class Initialization

Keiko Nakata
Joint work with A. Sabelfeld

January 2010

Protecting information confidentiality

Background

Conventional security mechanisms include

- access control
- firewalls
- encryption
- antivirus software
- security mechanisms for Java applets, such as
  - bytecode verifier
  - sandbox

They are useful but not perfect.

Protecting information confidentiality (2)

Background

We want to

- manipulate confidential information
- and communicate with the outside,
- while protecting the confidentiality.

Explicit vs implicit flows

Explicit flow:

$$l := h$$

Implicit flow:

$$h := h \bmod 2;$$
$$l := 0;$$
$$\text{if } h = 0 \text{ then } l := 1 \text{ else skip}$$

Covert channels

- Implicit flows
- Termination channels
- Timing channels
- Probabilistic channels
- Resource exhaustion channels
- Power channels

Noninterference

The attacker may view nonconfidential (public) information, which must not affected by confidential (private) data.

A program satisfies noninterference if the attacker cannot observe any difference between two executions that differ only in their confidential input.

The While language

$$
\begin{array}{rcl}
x, y, z & \in & \textit{Variables} \\
n & \in & \textit{Integers} \\
\sigma & \in & \textit{Variables} \rightarrow \textit{Integers}
\end{array}
$$

$$
\begin{array}{llcl}
\textit{Expressions} & e & ::= & n \mid x \mid e_0 \; \textit{op} \; e_1 \\
\textit{statement} & s & ::= & \text{skip} \mid s_0; s_1 \mid x := e \\
 & & \mid & \text{if } e \text{ then } s_t \text{ else } s_f \mid \text{while } e \text{ do } s_t
\end{array}
$$

Expression evaluation

$$\frac{}{(\sigma, n) \downarrow (\sigma, n)} \quad \frac{}{(\sigma, x) \downarrow (\sigma, \sigma(x))}$$

$$\frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \downarrow (\sigma'', n_1) \quad n_0 \; op \; n_1 = n}{(\sigma, e_0 \; op \; e_1) \downarrow (\sigma'', n)}$$

$$\frac{(\sigma, e_0) \uparrow \sigma'}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma'} \quad \frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \uparrow \sigma''}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma''}$$

$$\frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \downarrow (\sigma'', n_1) \quad n_0 \; op \; n_1 = \bullet}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma''}$$

Operational semantics of statements (1)

$$\frac{(\sigma, e) \downarrow (\sigma', n)}{\langle \sigma, x := e \rangle \rightarrow \langle \sigma'[x \mapsto n], \mathsf{skip} \rangle}$$

$$\frac{}{\langle \sigma, \mathsf{skip}; s \rangle \rightarrow \langle \sigma, s \rangle}$$

$$\frac{\langle \sigma, s_0 \rangle \rightarrow \langle \sigma', s_0' \rangle}{\langle \sigma, s_0; s_1 \rangle \rightarrow \langle \sigma', s_0'; s_1 \rangle} \qquad \frac{\langle \sigma, s_0 \rangle \rightarrow \langle \sigma', \bullet \rangle}{\langle \sigma, s_0; s_1 \rangle \rightarrow \langle \sigma', \bullet \rangle}$$

$$\frac{(\sigma, e) \downarrow (\sigma', n) \quad n \neq 0}{\langle \sigma, \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \rangle \rightarrow \langle \sigma', s_t \rangle}$$

$$\frac{(\sigma, e) \downarrow (\sigma', 0)}{\langle \sigma, \mathsf{if}\ e\ \mathsf{then}\ s_t\ \mathsf{else}\ s_f \rangle \rightarrow \langle \sigma', s_f \rangle}$$

Operational semantics of statements (2)

$$\frac{(\sigma, e) \downarrow (\sigma', n) \quad n \neq 0}{\langle \sigma, \text{while } e \text{ do } s_t \rangle \rightarrow \langle \sigma', s_t; \text{while } e \text{ do } s_t \rangle}$$

$$\frac{(\sigma, e) \downarrow (\sigma', 0)}{\langle \sigma, \text{while } e \text{ do } s_t \rangle \rightarrow \langle \sigma', \text{skip} \rangle}$$

$$\frac{(\sigma, e) \uparrow \sigma'}{\langle \sigma, Q[e] \rangle \rightarrow \langle \sigma', \bullet \rangle}$$

where $Q ::= x := [] \mid \text{if } [] \text{ then } s_t \text{ else } s_f \mid \text{while } [] \text{ do } s_t$

Security lattice

A simple security lattice consisting of only two levels *low* (public) and *high* (secret), with $\mathit{low} \sqsubset \mathit{high}$.
Metavariables ℓ and *pc* range over security levels.

A security environment Γ is a finite mapping from variables to their security levels.

Two states σ and σ' are low-equivalent, $\sigma =_{\mathit{low}} \sigma'$, if they agree on low variables . Formally, $\sigma =_{\mathit{low}} \sigma'$ if for any x such that $\Gamma(x) = \mathit{low}$, $\sigma(x) = \sigma'(x)$.

Termination-insensitive noninterference

A statement s satisfies termination-insensitive noninterference if, for any low-equivalent states σ_0 and σ_1, $\langle \sigma_0, s \rangle \downarrow \sigma_0'$ and $\langle \sigma_1, s \rangle \downarrow \sigma_1'$ imply $\sigma_0' =_{low} \sigma_1'$.

Termination-sensitive noninterference

A statement s satisfies termination-sensitive noninterference if, for any low-equivalent states σ_0 and σ_1, we have

- $\langle \sigma_0, s \rangle \downarrow \sigma_0'$ and $\langle \sigma_1, s \rangle \downarrow \sigma_1'$ and $\sigma_0' =_{low} \sigma_1'$;
- or, $\langle \sigma_0, s \rangle \uparrow \sigma_0'$ $\langle \sigma_1, s \rangle \uparrow \sigma_1'$ and $\sigma_0' =_{low} \sigma_1'$;
- or, both diverge.

Termination-sensitive vs insensitive

Conventional systems enforce termination-insensitive noninterference.

(Relatively) liberal security conditions, which are easy to check.

Programs satisfying TIN do not reliably leak the secret in polynomial time in the size of the secret.

If the secret is uniformly distributed, the attacker's advantage is negligible.

We adopt termination-insensitive noninterference.

Security type system

$$\frac{}{pc \vdash \text{skip}} \qquad \frac{pc \vdash e \quad \Gamma(e) \sqsubseteq \Gamma(x) \quad pc \sqsubseteq \Gamma(x)}{pc \vdash x := e}$$

$$\frac{pc \vdash s_0 \quad pc \vdash s_1}{pc \vdash s_0; s_1}$$

$$\frac{pc \vdash e \quad pc \sqcup \Gamma(e) \vdash s_t \quad pc \sqcup \Gamma(e) \vdash s_f}{pc \vdash \text{if } e \text{ then } s_t \text{ else } s_f}$$

$$\frac{pc \vdash e \quad pc \sqcup \Gamma(e) \vdash s_t}{pc \vdash \text{while } e \text{ do } s_t}$$

Soundness

Proposition

If $pc \vdash s$ then s satisfies TI noninterference.

Exception handling

Insecure:

$l := 1;$
try $h' := 1/h; l := 0$ catch skip

Secure:

$l := 1;$
try $h' := 1/h$ catch skip;
$l := 0$

Insecure, magnified:

$i := 0;$
while $i < n$ do
  $l_i := 0;$
  try $h' := 1/(h \,\&\, 2^i); l_i := 1$ catch skip;
  $i := i + 1$

While with try-catch

$$
\begin{array}{rcl}
x, y, z & \in & \textit{Variables} \\
n & \in & \textit{Integers} \\
\sigma & \in & \textit{Variables} \rightarrow \textit{Integers}
\end{array}
$$

$$
\begin{array}{lrcl}
\textit{Expressions} & e & ::= & n \mid x \mid e_0 \; \textit{op} \; e_1 \\
\textit{statement} & s & ::= & \text{skip} \mid s_0; s_1 \mid x := e \\
 & & \mid & \text{if } e \text{ then } s_t \text{ else } s_f \mid \text{while } e \text{ do } s_t \\
 & & \mid & \text{try } s_0 \text{ catch } s_1
\end{array}
$$

Operational semantics for exception handling

$$\frac{}{\langle \sigma, \text{try skip catch } s \rangle \rightarrow \langle \sigma, \text{skip} \rangle}$$

$$\frac{\langle \sigma, s_0 \rangle \rightarrow \langle \sigma', s_0' \rangle}{\langle \sigma, \text{try } s_0 \text{ catch } s_1 \rangle \rightarrow \langle \sigma', \text{try } s_0' \text{ catch } s_1 \rangle}$$

$$\frac{\langle \sigma, s_0 \rangle \rightarrow \langle \sigma', \bullet \rangle}{\langle \sigma, \text{try } s_0 \text{ catch } s_1 \rangle \rightarrow \langle \sigma', s_1 \rangle}$$

Typing of expressions for exception handling

$$\frac{}{pc \vdash n : low} \qquad \frac{}{pc \vdash x : low}$$

$$\frac{pc \vdash e_0 : \ell_0 \quad pc \vdash e_1 : \ell_1}{pc \vdash e_0 \; op \; e_1 : \ell_0 \sqcup \ell_1 \sqcup \Gamma(e_0) \sqcup \Gamma(e_1)}$$

Typing of statements for exception handling

$$\frac{}{pc \vdash \text{skip} : \mathit{low}} \qquad \frac{pc \vdash e : \ell \quad \Gamma(e) \sqsubseteq \Gamma(x) \quad pc \sqsubseteq \Gamma(x)}{pc \vdash x := e : \ell}$$

$$\frac{pc \vdash s_0 : \ell_0 \quad pc \sqcup \ell_0 \vdash s_1 : \ell_1}{pc \vdash s_0; s_1 : \ell_1 \sqcup \ell_2}$$

$$\frac{\begin{array}{c} pc \vdash e : \ell \\ \Gamma(e) \sqcup \ell \sqcup pc \vdash s_t : \ell_0 \quad \Gamma(e) \sqcup \ell \sqcup pc \vdash s_f : \ell_1 \end{array}}{pc \vdash \text{if } e \text{ then } s_t \text{ else } s_f : \ell \sqcup \ell_0 \sqcup \ell_1}$$

$$\frac{pc \vdash e : \ell \quad \Gamma(e) \sqcup \ell \sqcup pc \vdash s_t : \ell'}{pc \vdash \text{while } e \text{ do } s_t : \ell \sqcup \ell'}$$

$$\frac{pc \vdash s_0 : \ell_0 \quad pc \sqcup \ell_0 \vdash s_1 : \ell_1}{pc \vdash \text{try } s_0 \text{ catch } s_1 : \ell_1}$$

Soundness

Proposition

If $pc \vdash s : \ell$ then s satisfies TI noninterference.

Lazy class initialization

In Java, a class is loaded, linked and initialized lazily on demand when the class is actively used for the first time.

Class loading constitutes one of the most compelling features of the Java platform, allowing for dynamically loading software components on the Java platform.

The rest of the talk discusses security implications of class initialization.

Exception during initialization

$$\text{class } C \{\ g = 1\ \}$$
$$\text{class } D \{\ f = 1/C.g\ \}$$

with fields g and f low.

P_0 : $C.g := 0$;
if $h = 0$ then new D else skip

P_1 : new D;
$C.g := 0$;
if $h = 0$ then new D else skip

Persistence of initialization failure

$$\text{class } C \{\ g = 1\ \}$$
$$\text{class } D \{\ f = 1/C.g\ \}$$

with fields g and f low.

P_2 : $C.g := 0;$
if $h = 0$ then (try new D catch skip) else skip;
$C.g := 1;$
new D

P_3 : $C.g := 0;$
if $h = 0$ then (try new D catch skip) else skip;
$C.g := 1;$
try new D; $l := 1$ catch $l := 0$

Impact of class hierarchy

class C_0 { $g = 1$ }
class D_0 { $f = C_0.g{+}{+}$ }
class D_1 extends D_0 {}

P_4 : new C_0;
if $h = 0$ then new D_1 else skip

class hierarchy + persistence of initialization failure

class C_0 { $g = 1$ }
class D_0 { $f = 1/C_0.g$ }
class D_1 extends D_0 {}

P_5 : $C_0.g := 0$;
if $h = 0$ then (try new D_1 catch skip) else skip;
$C_0.g := 1$;
try new D_0; $l := 1$ catch $l := 0$

Syntax

$$
\begin{array}{rcl}
CT & \in & \textit{Class names} \to \textit{Class definitions} \\
\sigma & \in & \textit{Variables} \to \textit{Integers} \cup \textit{Class names} \to \textit{Class object}
\end{array}
$$

$$
\begin{array}{lrcl}
\textit{Expressions} & e & ::= & n \mid x \mid e_0 \; op \; e_1 \mid C.f \\
\textit{Statements} & s & ::= & \text{skip} \mid s_0; s_1 \mid x := e \mid C.f := e \\
 & & \mid & \text{if } e \text{ then } s_t \text{ else } s_f \mid \text{while } e \text{ do } s_t \\
\textit{Class definitions} & CL & ::= & \text{class } C \, \{f_0 = e_0, \ldots, f_k = e_k\} \\
\textit{Programs} & P & ::= & (CT, s) \\
\textit{Class object} & o & ::= & \{f_0 = n_0, \ldots, f_k = n_k\} \mid \bullet \mid \circ
\end{array}
$$

Expression evaluation

$$\frac{}{(\sigma, n) \downarrow (\sigma, n)} \quad \frac{}{(\sigma, x) \downarrow (\sigma, \sigma(x))}$$

$$\frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \downarrow (\sigma'', n_1) \quad n_0 \; op \; n_1 = n}{(\sigma, e_0 \; op \; e_1) \downarrow (\sigma'', n)}$$

$$\frac{(\sigma, e_0) \uparrow \sigma'}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma'} \quad \frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \uparrow \sigma''}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma''}$$

$$\frac{(\sigma, e_0) \downarrow (\sigma', n_0) \quad (\sigma', e_1) \downarrow (\sigma'', n_1) \quad n_0 \; op \; n_1 = \bullet}{(\sigma, e_0 \; op \; e_1) \uparrow \sigma''}$$

$$\frac{\rho(\sigma, C) \downarrow \sigma'}{(\sigma, C.f) \downarrow (\sigma', \sigma'(C.f))} \quad \frac{\rho(\sigma, C) \uparrow \sigma'}{(\sigma, C.f) \uparrow \sigma'}$$

Operational semantics of statements

$$\frac{(\sigma, e) \downarrow (\sigma', n) \quad \rho(\sigma', C) \downarrow \sigma''}{\langle \sigma, C.f := e \rangle \rightarrow \langle \sigma''[C.f \mapsto n], \mathsf{skip} \rangle}$$

$$\frac{(\sigma, e) \uparrow \sigma'}{\langle \sigma, C.f := e \rangle \rightarrow \langle \sigma', \bullet \rangle} \qquad \frac{(\sigma, e) \downarrow (\sigma', n) \quad \rho(\sigma', C) \uparrow \sigma''}{\langle \sigma, C.f := e \rangle \rightarrow \langle \sigma'', \bullet \rangle}$$

Class initialization

$$\frac{\sigma(C) = \{f_0 = n_0, \ldots, f_k = n_k\}}{\rho(\sigma, C) \downarrow \sigma} \qquad \frac{\sigma(C) = \bullet}{\rho(C, \sigma) \uparrow \sigma}$$

$$\frac{\begin{array}{c}\sigma(C) = \circ \quad CT(C) = \text{class } C\ \{f_0 = e_0, \ldots, f_k = e_k\} \\ \langle \sigma[C \mapsto \{f_0 = 0, \ldots, f_k = 0\}], C.f_0 := e_0; \ldots; C.f_k := e_k \rangle \downarrow \sigma'\end{array}}{\rho(\sigma, C) \downarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma(C) = \circ \quad CT(C) = \text{class } C\ \{f_0 = e_0, \ldots, f_k = e_k\} \\ \langle \sigma[C \mapsto \{f_0 = 0, \ldots, f_k = 0\}], C.f_0 := e_0; \ldots; C.f_k := e_k \rangle \uparrow \sigma'\end{array}}{\rho(C, \sigma) \uparrow \sigma'[C \mapsto \bullet]}$$

Low equivalence on states

Two states σ and σ' are *low-equivalent*, written $\sigma =_{low} \sigma'$, if they agree on low variables and fields and on class objects.

Formally, $\sigma =_{low} \sigma'$ if the following three conditions hold:

- for any x such that $\Gamma(x) = low$, $\sigma(x) = \sigma'(x)$;
- $uninitialized(\sigma) = uninitialized(\sigma')$, $initialized(\sigma) = initialized(\sigma')$, and also $failed(\sigma) = failed(\sigma')$;
- for any C and f such that $\Gamma(C, f) = low$ and C is in $initialized(\sigma)$, $\sigma(C.f) = \sigma'(C.f)$.

Noninterference

Definition

A statement s satisfies termination-insensitive noninterference if, for any low-equivalent states σ_0 and σ_1, $\langle \sigma_0, s \rangle \downarrow \sigma_0'$ and $\langle \sigma_1, s \rangle \downarrow \sigma_1'$ imply $\sigma_0' =_{low} \sigma_1'$.

Typing of expressions

$$\frac{}{pc \vdash n : \delta \hookrightarrow \delta} \qquad \frac{}{pc \vdash x : \delta \hookrightarrow \delta}$$

$$\frac{pc \vdash e_0 : \delta \hookrightarrow \delta_0 \qquad pc \vdash e_1 : \delta \hookrightarrow \delta_1}{pc \vdash e_0 \ op \ e_1 : \delta \hookrightarrow \delta_0 \cup \delta_1}$$

$$\frac{}{low \vdash C.f : \delta \hookrightarrow \delta \cup \{C\}} \qquad \frac{C \in \delta}{high \vdash C.f : \delta \hookrightarrow \delta}$$

Typing of statements (1)

$$\frac{}{pc \vdash \mathsf{skip} : \delta \hookrightarrow \delta} \qquad \frac{pc \vdash e : \delta \hookrightarrow \delta' \quad \Gamma(e) \sqsubseteq \Gamma(x) \quad pc \sqsubseteq \Gamma(x)}{pc \vdash x := e : \delta \hookrightarrow \delta'}$$

$$\frac{low \vdash e : \delta \hookrightarrow \delta' \quad \Gamma(e) \sqsubseteq \Gamma(C.f)}{low \vdash C.f := e : \delta \hookrightarrow \delta' \cup \{C\}}$$

$$\frac{C \in \delta \quad high \vdash e : \delta \hookrightarrow \delta' \quad \Gamma(e) \sqsubseteq \Gamma(C.f) \quad high \sqsubseteq \Gamma(C.f)}{high \vdash C.f := e : \delta \hookrightarrow \delta'}$$

$$\frac{pc \vdash s_0 : \delta_0 \hookrightarrow \delta_1 \quad pc \vdash s_1 : \delta_1 \hookrightarrow \delta_2}{pc \vdash s_0; s_1 : \delta_0 \hookrightarrow \delta_2}$$

Typing of statements (2)

$$\frac{pc \vdash e : \delta \hookrightarrow \delta' \quad pc \sqcup \Gamma(e) \vdash s_t : \delta' \hookrightarrow \delta_0 \quad pc \sqcup \Gamma(e) \vdash s_f : \delta' \hookrightarrow \delta_1}{pc \vdash \text{if } e \text{ then } s_t \text{ else } s_f : \delta \hookrightarrow \delta_0 \cap \delta_1}$$

$$\frac{pc \vdash e : \delta \hookrightarrow \delta' \quad pc \sqcup \Gamma(e) \vdash s_t : \delta' \hookrightarrow \delta''}{pc \vdash \text{while } e \text{ do } s_t : \delta \hookrightarrow \delta'}$$

Soundness

Proposition

If $pc \vdash s : \emptyset \hookrightarrow \delta$ *then* s *satisfies TI noninterference.*