Tight Enforcement of Information-Release Policies for Dynamic Languages

Aslan Askarov and Andrei Sabelfeld
(presented by Lindsey Kuper)
Motivation

• Securing applications written in dynamically typed languages is an important problem.

• We want to do better than noninterference.

• We want to handle dynamic languages.
  • In particular, we want to handle \texttt{eval}.

• We want tight enforcement: everything that the policy allows and nothing it doesn’t.

• \textit{Can we have our cake and eat it too?}
Security model
Security model

• Think of an attacker as trying to discover the contents of memory $m$. 
Security model

- Think of an attacker as trying to discover the contents of memory $m$.
- There is some finite set, say, $M$, of things that $m$ might be. At first, this set is the attacker’s knowledge $k$. As the attacker learns more, $k$ shrinks. (It’s helpful to think of $k$ as uncertainty.)
Security model

• Think of an attacker as trying to discover the contents of memory $m$.

• There is some finite set, say, $M$, of things that $m$ might be. At first, this set is the attacker’s knowledge $k$. As the attacker learns more, $k$ shrinks. (It’s helpful to think of $k$ as uncertainty.)

• Under noninterference, $k$ would have to always remain equal to $M$...
Security model

- Think of an attacker as trying to discover the contents of memory $m$.
- There is some finite set, say, $M$, of things that $m$ might be. At first, this set is the attacker’s knowledge $k$. As the attacker learns more, $k$ shrinks. (It’s helpful to think of $k$ as uncertainty.)
- Under noninterference, $k$ would have to always remain equal to $M$...
- But under a model that allows declassification, we just have to ensure that $k$ remains at least as large as some set $p$ representing our security policy.
Security model

• Think of an attacker as trying to discover the contents of memory \( m \).

• There is some finite set, say, \( M \), of things that \( m \) might be. At first, this set is the attacker’s knowledge \( k \). As the attacker learns more, \( k \) shrinks. (It’s helpful to think of \( k \) as uncertainty.)

• Under noninterference, \( k \) would have to always remain equal to \( M \)...  

• But under a model that allows declassification, we just have to ensure that \( k \) remains at least as large as some set \( p \) representing our security policy.

• Events that may cause \( k \) to shrink form a sequence of low events \( \ell \).
Security model

• But under a model that allows declassification, we just have to ensure that $k$ remains at least as large as some set $p$ representing our security policy.

• Events that may cause $k$ to shrink form a sequence of low events $\vec{\ell}$. 
But under a model that allows declassification, we just have to ensure that $k$ remains at least as large as some set $p$ representing our security policy.

Events that may cause $k$ to shrink form a sequence of low events $\ell$. 
Security model

• But under a model that allows declassification, we just have to ensure that $k$ remains at least as large as some set $p$ representing our security policy.

• Events that may cause $k$ to shrink form a sequence of low events $\ell$. 
Security model

- But under a model that allows declassification, we just have to ensure that $k$ remains at least as large as some set $p$ representing our security policy.

- Events that may cause $k$ to shrink form a sequence of low events $\ell$. 
Security model
Security model
Security model
Security model

- A low event $\ell$ is a change in the attacker’s ability to distinguish memories (i.e., an opportunity for $k$ to shrink).

- Three kinds of low events:
  - $(x, v)$, an assignment of $v$ to low variable $x$
  - $\downarrow$, the program ends
  - $\epsilon$, the empty low event

- An escape hatch is a declassified program expression by which knowledge about the contents of memory can escape to the attacker. $E$ is a set of escape hatches.
The end
Definition of termination-sensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \ I(E) \ m \} \]

\[ k(c, i_L, \vec{\ell}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \rightarrow_{\vec{\ell}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{\ell} \) and initial low memory \( i_L \), denoted \( TSec(c, i_L, \vec{\ell}) \), if for all memories \( m \in k(c, i_L, \vec{\ell}) \) that produce \( \vec{\ell} \) we have:

\[ \forall i . 1 \leq i \leq n . p(m, E_i) \subseteq k(c, m_L, \vec{\ell}_i) \]

where \( \vec{\ell}_i \) is the \( i \)-prefix of \( \vec{\ell} \), \( \vec{\ell} = \vec{\ell}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{\ell}_i \).
Definition of termination-sensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \ I(E) \ m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \xrightarrow{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TSec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \) we have:

\[ \forall i \cdot 1 \leq i \leq n \cdot p(m, E_i) \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-sensitive security

\[ p(m, E) = \{m' \mid m'_L = m_L \land m' \text{ I}(E) m\} \]

\[ k(c, i_L, \vec{l}) = \{m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \xrightarrow{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TSec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \) we have:

\[ \forall i. 1 \leq i \leq n \cdot p(m, E_i) \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-sensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \ I(E) m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \xrightarrow{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TSec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \) we have:

\[ \forall i. \ 1 \leq i \leq n . \ p(m, E_i) \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
If termination-sensitivity is too strong a notion of security...

- ...we can define a notion of termination-insensitive security that ignores information leaks caused by termination.

- Before, we demanded that $p$, which represents the knowledge that the policy releases, was a subset of $k$, which represents the attacker's knowledge.

- Now, we just require the part of $p$ that contains progress knowledge to be a subset of $k$. 
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' I(E) m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \xrightarrow{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TISec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \), we have: \( \forall i . 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} k(c, m_L, \vec{l}_{i-1} \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \mathbin{I(E)} m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \longrightarrow_{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( \text{TISec}(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \), we have: \( \forall i . 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} k(c, m_L, \vec{l}_{i-1} \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \ I(E) \ m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \rightarrow_{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TISec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \), we have: \( \forall i . \ 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} \ k(c, m_L, \vec{l}_{i-1}' \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' I(E) m \} \]
\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land \langle c, m, \emptyset \rangle \xrightarrow{\vec{l}} \langle c', m', E' \rangle \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TISec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \), we have: \( \forall i \cdot 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} k(c, m_L, \vec{l}_{i-1} \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = m_L \land m' \xrightarrow{I(E)} m \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \land (c, m, \emptyset) \xrightarrow{\vec{l}} (c', m', E') \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{l} \) and initial low memory \( i_L \), denoted \( TISec(c, i_L, \vec{l}) \), if for all memories \( m \in k(c, i_L, \vec{l}) \) that produce \( \vec{l} \), we have: \( \forall i . 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} k(c, m_L, \vec{l}_{i-1} \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Definition of termination-insensitive security

\[ p(m, E) = \{ m' \mid m'_L = E \} \]

\[ k(c, i_L, \vec{l}) = \{ m \mid m_L = i_L \} \]

A program \( c \) is secure of low events \( \vec{l} \) and initial state \( TISec(c, i_L, \vec{l}) \), if for all \( m \) that produce \( \vec{l} \), we have: \( \forall i . 1 \leq i \leq n \).

\[ p(m, E_i) \cap \bigcup_{\vec{l}'} k(c, m_L, \vec{l}_{i-1} \vec{l}') \subseteq k(c, m_L, \vec{l}_i) \]

where \( \vec{l}_i \) is the \( i \)-prefix of \( \vec{l} \), \( \vec{l} = \vec{l}_n \) for some \( n \), and \( E_i \) is extracted from the configuration that generated the last event in \( \vec{l}_i \).
Enforcing termination-insensitive security

• A purely dynamic enforcement mechanism is all we need for this definition of security.

• We define the semantics of a monitor, which executes in sync with our program, with a set of monitor events.

• Our little language:

$$e ::= n \mid s \mid x \mid e \ op \ e$$

$$c ::= \text{skip} \mid x := e \mid x := \text{declassify}(e) \mid c; c$$

$$\mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c \mid \text{eval}(e)$$
Monitor events

• *nop*: program performs a **skip**
• *a(x, e)*: program assigns the value of *e* to *x*
• *d(x, e, m)*: program declassifies *e* into the variable *x*
in the context of memory *m*
• *b(e, c₁; c₂)*: program branches on *e* and is about to
enter *c₁* or *c₂*
• *we(e)*: program enters or skips a **while** loop with
guard *e* or runs **eval(e)**
• *f*: program has finished a conditional, loop, or **eval**
Monitor semantics for termination-insensitive enforcement

\[
\begin{align*}
\langle i, \text{st} \rangle & \xrightarrow{\text{nop}} \langle i, \text{st} \rangle \quad \frac{\text{lev}(e) \sqsubseteq \Gamma(x) \quad \text{lev}(\text{st}) \sqsubseteq \Gamma(x)}{\langle i, \text{st} \rangle \xrightarrow{a(x,e)} \langle i, \text{st} \rangle} \\
\langle i, \text{hd} : \text{st} \rangle & \xrightarrow{f} \langle i, \text{st} \rangle \quad \frac{\text{m}(e) = i(e) \quad \text{lev}(\text{st}) \sqsubseteq \Gamma(x)}{\langle i, \text{st} \rangle \xrightarrow{d(x,e,m)} \langle i, \text{st} \rangle} \\
\langle i, \text{st} \rangle & \xrightarrow{b(e,c)} \langle i, \text{lev}(e) : \text{st} \rangle \quad \langle i, \text{st} \rangle \xrightarrow{w(e)} \langle i, \text{lev}(e) : \text{st} \rangle
\end{align*}
\]

The stack, \( \text{st} \), handles implicit flow: \( \text{hd} : \text{st} \) pops the top security level, \( \text{lev}(e) : \text{st} \) pushes.
Monitor semantics for termination-insensitive enforcement

\[\langle i, st \rangle \xrightarrow{nop} \langle i, st \rangle\]

\[
\frac{lev(e) \sqsubseteq \Gamma(x) \quad lev(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{a(x,e)} \langle i, st \rangle}
\]

\[
\langle i, hd:st \rangle \xrightarrow{f} \langle i, st \rangle\]

\[
\frac{m(e) = i(e) \quad lev(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{d(x,e,m)} \langle i, st \rangle}
\]

\[
\langle i, st \rangle \xrightarrow{b(e,c)} \langle i, lev(e):st \rangle \quad \langle i, st \rangle \xrightarrow{we(e)} \langle i, lev(e):st \rangle
\]

The stack, \( st \), handles implicit flow:

- \( hd : st \) pops the top security level,
- \( lev(e) : st \) pushes.
Monitor semantics for termination-insensitive enforcement

The stack, $st$, handles implicit flow:

- $hd : st$ pops the top security level,
- $lev(e) : st$ pushes.
Monitor semantics for termination-insensitive enforcement

\[
\begin{align*}
\langle i, st \rangle & \xrightarrow{n_{op}} \langle i, st \rangle \\
\frac{\text{lev}(e) \sqsubseteq \Gamma(x) \quad \text{lev}(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{a(x,e)} \langle i, st \rangle} \\
\langle i, st \rangle & \xrightarrow{f} \langle i, st \rangle \\
\frac{m(e) = i(e) \quad \text{lev}(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{d(x,e,m)} \langle i, st \rangle} \\
\langle i, st \rangle & \xrightarrow{b(e,c)} \langle i, \text{lev}(e) : st \rangle \\
\langle i, st \rangle & \xrightarrow{we(e)} \langle i, \text{lev}(e) : st \rangle
\end{align*}
\]

The stack, \( st \), handles implicit flow:
\( \text{hd} : st \) pops the top security level, \( \text{lev}(e) : st \) pushes.
Monitor semantics for termination-insensitive enforcement

\[ \langle i, st \rangle \xrightarrow{\text{nop}} \langle i, st \rangle \]

\[ \frac{\text{lev}(e) \sqsubseteq \Gamma(x) \quad \text{lev}(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{a(x,e)} \langle i, st \rangle} \]

\[ \langle i, hd: st \rangle \xrightarrow{f} \langle i, st \rangle \]

\[ \frac{m(e) = i(e) \quad \text{lev}(st) \sqsubseteq \Gamma(x)}{\langle i, st \rangle \xrightarrow{d(x,e,m)} \langle i, st \rangle} \]

\[ \langle i, st \rangle \xrightarrow{b(e,c)} \langle i, \text{lev}(e): st \rangle \]

\[ \langle i, st \rangle \xrightarrow{we(e)} \langle i, \text{lev}(e): st \rangle \]

The stack, \( st \), handles implicit flow:

\( hd : st \) pops the top security level,

\( \text{lev}(e) : st \) pushes.
Monitor semantics for termination-insensitive enforcement

The stack, $st$, handles implicit flow:

- $hd : st$ pops the top security level,
- $lev(e) : st$ pushes.
Monitor semantics for termination-insensitive enforcement

The stack, $st$, handles implicit flow:

- $hd : st$ pops the top security level,
- $lev(e) : st$ pushes.
Enforcing termination-sensitive security

• More difficult to enforce than termination-insensitive security. For example:
  • if \( h \) then \( l := 1 \) else skip
  • while \( h \) do skip

• We’ll need to do static analysis (not much, but just enough)
  • We’ll update the monitor semantics to accommodate this change
Primitive calls for static analysis

- Things we might want to know about an expression $e$ statically:
  - $\text{vars}(e)$, the set of variables in $e$
  - $\text{noeval}(e)$ holds if no $\text{eval}$ statements occur in $e$
  - $\text{noloop}(e)$ holds if no $\text{while}$ loops occur in $e$
  - $\text{upd}(e)$, the set of variables assigned to in $e$
  - $\text{lev}(c)$, lowest level of a variable assigned in $c$
Updated monitor semantics for termination-sensitive enforcement

\[ \langle st, U \rangle \xrightarrow{nop} \langle st, U \rangle \quad \langle hd: st, U \rangle \xrightarrow{f} \langle st, U \rangle \]

\[
\frac{\text{lev}(st) = L \implies \text{lev}(e) \subseteq \text{lev}(x)}{\langle st, U \rangle \xrightarrow{a(x,e)} \langle st, U \cup \{x\} \rangle}
\]

\[
\frac{\text{lev}(e) \subseteq \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset}{\text{lev}(e) = H \implies \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c)}
\]

\[
\frac{\text{vars}(e) \cap U = \emptyset}{\langle st, U \rangle \xrightarrow{d(x,e,m)} \langle st, U \cup \{x\} \rangle}
\]

\[
\frac{\text{lev}(e) = L}{\langle st, U \rangle \xrightarrow{we(e)} \langle L : st, U \rangle}
\]
Updated monitor semantics for termination-sensitive enforcement

\[ \langle st, U \rangle \xrightarrow{\text{nop}} \langle st, U \rangle \quad \langle \text{hd: } st, U \rangle \xrightarrow{f} \langle st, U \rangle \]

\[
\frac{\text{lev}(st) = L \implies \text{lev}(e) \subseteq \text{lev}(x)}{
\langle st, U \rangle \xrightarrow{a(x,e)} \langle \text{st, } U \cup \{x\} \rangle}
\]

\[
\frac{\text{lev}(e) \subseteq \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset}{\langle st, U \rangle \xrightarrow{\text{b(e,c)}} \langle \text{lev(e): st, } U \cup U' \rangle}
\]

\[
\frac{\text{vars}(e) \cap U = \emptyset}{\langle st, U \rangle \xrightarrow{d(x,e,m)} \langle \text{st, } U \cup \{x\} \rangle}
\]

\[
\frac{\text{lev}(e) = L}{\langle st, U \rangle \xrightarrow{\text{we(e)}} \langle L: st, U \rangle}
\]
Updated monitor semantics for termination-sensitive enforcement

\[
\begin{align*}
\langle st, U \rangle \xrightarrow{\text{nop}} & \langle st, U \rangle \\
\langle hd : st, U \rangle \xrightarrow{f} & \langle st, U \rangle \\
\end{align*}
\]

\[
\begin{align*}
\text{lev}(st) = L \implies & \text{lev}(e) \subseteq \text{lev}(x) \\
\langle st, U \rangle \xrightarrow{a(x,e)} & \langle st, U \cup \{x\} \rangle \\
\text{lev}(e) \subseteq & \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset \\
\text{lev}(e) = H \implies & \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c) \\
\langle st, U \rangle \xrightarrow{b(e,c)} & \langle \text{lev}(e) : st, U \cup U' \rangle \\
\text{vars}(e) \cap & U = \emptyset \\
\langle st, U \rangle \xrightarrow{d(x,e,m)} & \langle st, U \cup \{x\} \rangle \\
\text{lev}(e) = L \quad & \langle st, U \rangle \xrightarrow{\text{we}(e)} \langle L : st, U \rangle
\end{align*}
\]
Updated monitor semantics for termination-sensitive enforcement

\[\langle st, U \rangle \xrightarrow{\text{nop}} \langle st, U \rangle\]
\[\langle \text{hd: } st, U \rangle \xrightarrow{f} \langle st, U \rangle\]

\[
\frac{\text{lev}(st) = L \implies \text{lev}(e) \subseteq \text{lev}(x)}{
\langle st, U \rangle \xrightarrow{a(x,e)} \langle st, U \cup \{x\} \rangle}
\]

\[
\begin{align*}
\text{lev}(e) \subseteq \text{lev}(c) & \quad \text{lev}(e) = L \implies U' = \emptyset \\
\text{lev}(e) = H \implies \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c)
\end{align*}
\]

\[
\langle st, U \rangle \xrightarrow{b(e,c)} \langle \text{lev}(e) : st, U \cup U' \rangle
\]

\[
\begin{align*}
\text{vars}(e) \cap U = \emptyset & \quad \text{lev}(e) = L \\
\langle st, U \rangle \xrightarrow{d(x,e,m)} \langle st, U \cup \{x\} \rangle & \quad \langle st, U \rangle \xrightarrow{we(e)} \langle L : st, U \rangle
\end{align*}
\]
Updated monitor semantics for termination-sensitive enforcement

\[ \langle st, U \rangle \xrightarrow{\text{nop}} \langle st, U \rangle \]
\[ \langle \text{hd: } st, U \rangle \xrightarrow{f} \langle st, U \rangle \]

\[
\begin{align*}
\text{lev}(st) = L & \implies \text{lev}(e) \subseteq \text{lev}(x) \\
\langle st, U \rangle & \xrightarrow{a(x,e)} \langle st, U \cup \{x\} \rangle
\end{align*}
\]

\[\begin{align*}
\text{lev}(e) \subseteq \text{lev}(c) & \quad \text{lev}(e) = L \implies U' = \emptyset \\
\text{lev}(e) = H & \implies \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c)
\end{align*}\]

\[\begin{align*}
\langle st, U \rangle & \xrightarrow{b(e,c)} \langle \text{lev}(e) : st, U \cup U' \rangle \\
\text{vars}(e) \cap U & = \emptyset \\
\langle st, U \rangle & \xrightarrow{d(x,e,m)} \langle st, U \cup \{x\} \rangle \\
\text{lev}(e) = L & \implies \langle st, U \rangle \xrightarrow{we(e)} \langle L : st, U \rangle
\end{align*}\]
Updated monitor semantics for termination-sensitive enforcement

\[\langle st, U \rangle \xrightarrow{\text{nop}} \langle st, U \rangle\]

\[\langle hd: st, U \rangle \xrightarrow{f} \langle st, U \rangle\]

\[
\begin{align*}
\text{lev}(st) = L & \implies \text{lev}(e) \subseteq \text{lev}(x) \\
\langle st, U \rangle & \xrightarrow{a(x,e)} \langle st, U \cup \{x\} \rangle
\end{align*}
\]

\[
\begin{align*}
\text{lev}(e) & \subseteq \text{lev}(c) \\
\text{lev}(e) = L & \implies U' = \emptyset \\
\text{lev}(e) = H & \implies \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c) \\
\langle st, U \rangle & \xrightarrow{b(e,c)} \langle \text{lev}(e) : st, U \cup U' \rangle
\end{align*}
\]

\[
\begin{align*}
\text{vars}(e) \cap U = \emptyset & \\
\langle st, U \rangle & \xrightarrow{d(x,e,m)} \langle st, U \cup \{x\} \rangle
\end{align*}
\]

\[
\begin{align*}
\text{lev}(e) = L & \\
\langle st, U \rangle & \xrightarrow{we(e)} \langle L : st, U \rangle
\end{align*}
\]
Updated monitor semantics for termination-sensitive enforcement

\[ \langle st, U \rangle \xrightarrow{\text{nop}} \langle st, U \rangle \quad \langle \text{hd}: st, U \rangle \xrightarrow{f} \langle st, U \rangle \]

\[
\frac{\text{lev}(st) = L \Rightarrow \text{lev}(e) \subseteq \text{lev}(x)}{
\langle st, U \rangle \xrightarrow{a(x,e)} \langle st, U \cup \{x\} \rangle}
\]

\[
\begin{align*}
\text{lev}(e) \subseteq \text{lev}(c) & \quad \text{lev}(e) = L \Rightarrow U' = \emptyset \\
\text{lev}(e) = H & \Rightarrow \text{noeval}(c) \land \text{noloop}(c) \land U' = \text{upd}(c)
\end{align*}
\]

\[ \langle st, U \rangle \xrightarrow{\text{b}(e,c)} \langle \text{lev}(e): st, U \cup U' \rangle \]

\[
\frac{\text{vars}(e) \cap U = \emptyset}{\langle st, U \rangle \xrightarrow{d(x,e,m)} \langle st, U \cup \{x\} \rangle}
\]

\[
\frac{\text{lev}(e) = L}{\langle st, U \rangle \xrightarrow{\text{we}(e)} \langle L : st, U \rangle}
\]
Support for input/output

• Our goal: a notion of security that can accept programs that \texttt{eval} their input.

• We’ll have to deal with more than just initial memory $i$.

• We’ll also need to update our notions of attacker knowledge and indistinguishability.
Support for input/output

• Communication channels $\hat{L}$ and $\hat{H}$, each comprising an input stream and an output stream

• New language primitives for input and output

• Two new kinds of low events:
  • $(I, x, v)$, when input $v$ is received into variable $x$
  • $(0, v)$, when value $v$ is output

• Escape hatches become pairs $(e, r)$, where $r$ is the length of the input history
Support for input/output

Indistinguishability relation:

\[(m_1, \hat{H}_1) I(E, \hat{L}, hist) (m_2, \hat{H}_2) \Leftrightarrow \forall (e, r) \in E . m_1^{hist[r]}(e) = m_2^{hist[r]}(e)\]

where

- \(hist = (ch_n, x_n):(ch_{n-1}, x_{n-1}):\ldots:(ch_1, x_1)\),
- \(r \leq n\),
- \(hist[r] = (ch_r, x_r):(ch_{r-1}, x_{r-1}):\ldots:(ch_1, x_1)\),
- \(m_j^{hist[r]} = memupd(m, \hat{L}, \hat{H}_j, hist[r]), j = 1, 2\)

“When in doubt, add another environment to your relation.”
- Stevie Strickland
Definition of termination-sensitive security in the presence of I/O

\[ k(c, i_L, \hat{L}, \vec{\ell}) = \{ (m, \hat{H}) \mid m_L = i_L \land \langle c, m, \emptyset, \hat{L}, \hat{H}, \epsilon \rangle \rightarrow \vec{\ell} \langle c', m', E', \hat{L}', \hat{H}', \text{hist}' \rangle \} \]

\[ p(m, \hat{L}, \hat{H}, E, \text{hist}) = \{ (m', \hat{H}') \mid m_L = m'_L \land (m, \hat{H}) I(E, \hat{L}, \text{hist}) (m', \hat{H}') \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{\ell} \), initial low-memory \( i_L \), and initial low-communication environment \( \hat{L} \), denoted \( TSec(c, i_L, \hat{L}, \vec{\ell}) \), if for all environments \( (m, \hat{H}) \in k(c, i_L, \hat{L}, \vec{\ell}) \) that produce low events \( \vec{\ell} \) we have:

\[ \forall i. 1 \leq i \leq n . p(m, \hat{L}, \hat{H}, E_i, \text{hist}_i) \subseteq k(c, m_L, \hat{L}, \vec{\ell}_i) \]

where \( \vec{\ell}_i \) is the i-prefix of \( \vec{\ell} \), \( \vec{\ell} = \vec{\ell}_n \) for some \( n \), and \( E_i \) and \( \text{hist}_i \) are extracted from the configuration that generated the last event in \( \vec{\ell}_i \).
Definition of termination-sensitive security in the presence of I/O

\[
k(c, i_L, \hat{L}, \ell) = \{ (m, \hat{H}) \mid m_L = i_L \land \langle c, m, \emptyset, \hat{L}, \hat{H}, \epsilon \rangle \rightarrow \langle c', m', E', \hat{L}', \hat{H}', \text{hist}' \rangle \}\]

\[
p(m, \hat{L}, \hat{H}, E, \text{hist}) = \{ (m', \hat{H}') \mid m_L = m'_L \land (m, \hat{H}) I(E, \hat{L}, \text{hist}) (m', \hat{H}') \}\]

A program \( c \) is secure with respect to a sequence of low events \( \ell \), initial low-memory \( i_L \), and initial low-communication environment \( \hat{L} \), denoted \( TSec(c, i_L, \hat{L}, \ell) \), if for all environments \( (m, \hat{H}) \in k(c, i_L, \hat{L}, \ell) \) that produce low events \( \ell \) we have:

\[
\forall i. 1 \leq i \leq n . p(m, \hat{L}, \hat{H}, E_i, \text{hist}_i) \subseteq k(c, m_L, \hat{L}, \ell_i)
\]

where \( \ell_i \) is the \( i \)-prefix of \( \ell \), \( \ell = \ell_n \) for some \( n \), and \( E_i \) and \( \text{hist}_i \) are extracted from the configuration that generated the last event in \( \ell_i \).
Definition of termination-sensitive security in the presence of I/O

\[ k(c, i_L, \hat{L}, \hat{\ell}) = \{ (m, \hat{H}) \mid m_L = i_L \land \langle c, m, \emptyset, \hat{L}, \hat{H}, \epsilon \rangle \xrightarrow{\ell} \langle c', m', E', \hat{L}', \hat{H}', \text{hist}' \rangle \} \]

\[ p(m, \hat{L}, \hat{H}, E, \text{hist}) = \{ (m', \hat{H}') \mid m_L = m'_L \land (m, \hat{H}) I(E, \hat{L}, \text{hist}) (m', \hat{H}') \} \]

A program \( c \) is secure with respect to a sequence of low events \( \hat{\ell} \), initial low-memory \( i_L \), and initial low-communication environment \( \hat{L} \), denoted \( TSec(c, i_L, \hat{L}, \hat{\ell}) \), if for all environments \( (m, \hat{H}) \in k(c, i_L, \hat{L}, \hat{\ell}) \) that produce low events \( \hat{\ell} \) we have:

\[ \forall i. 1 \leq i \leq n . \ p(m, \hat{L}, \hat{H}, E_i, \text{hist}_i) \subseteq k(c, m_L, \hat{L}, \hat{\ell}_i) \]

where \( \hat{\ell}_i \) is the i-prefix of \( \hat{\ell} \), \( \hat{\ell} = \hat{\ell}_n \) for some \( n \), and \( E_i \) and \( \text{hist}_i \) are extracted from the configuration that generated the last event in \( \hat{\ell}_i \).
Definition of termination-sensitive security in the presence of I/O

\[ k(c, i_L, \hat{L}, \vec{\ell}) = \{(m, \hat{H}) \mid m_L = i_L \land \langle c, m, \emptyset, \hat{L}, \hat{H}, \epsilon \rangle \rightarrow_\vec{\ell} \langle c', m', E', \hat{L}', \hat{H}', \text{hist}' \rangle \} \]

\[ p(m, \hat{L}, \hat{H}, E, \text{hist}) = \{(m', \hat{H}') \mid m_L = m'_L \land (m, \hat{H}) \xrightarrow{I(E, \hat{L}, \text{hist})} (m', \hat{H}') \} \]

A program \( c \) is secure with respect to a sequence of low events \( \vec{\ell} \), initial low-memory \( i_L \), and initial low-communication environment \( \hat{L} \), denoted \( \overline{TSec(c, i_L, \hat{L}, \vec{\ell})} \), if for all environments \( (m, \hat{H}) \in k(c, i_L, \hat{L}, \vec{\ell}) \) that produce low events \( \vec{\ell} \) we have:

\[ \forall i. \ 1 \leq i \leq n. \ p(m, \hat{L}, \hat{H}, E_i, \text{hist}_i) \subseteq k(c, m_L, \hat{L}, \vec{\ell}_i) \]

where \( \vec{\ell}_i \) is the i-prefix of \( \vec{\ell} \), \( \vec{\ell} = \vec{\ell}_n \) for some \( n \), and \( E_i \) and \( \text{hist}_i \) are extracted from the configuration that generated the last event in \( \vec{\ell}_i \).
Updated monitor semantics for enforcement in the presence of I/O

\[
\begin{align*}
\text{lev}(st) &= L \implies U' = U \setminus \{x\} \\
\text{lev}(st) &= H \implies U' = U \\
\langle st, U, ct \rangle &\xrightarrow{\text{in}(x,v)} \langle st, U', \text{lev}(st) \sqcup ct \rangle \\
\text{lev}(st) &= L \implies \text{lev}(e) \sqsubseteq \text{lev}(ch) \\
\langle st, U, ct \rangle &\xrightarrow{\text{out}(ch,e)} \langle st, U, ct \rangle \\
\text{vars}(e) \cap U &= \emptyset \\
\text{lev}(ct) &\sqsubseteq \text{lev}(x) \\
\langle st, U, ct \rangle &\xrightarrow{d(x,e,m)} \langle st, U \cup \{x\}, ct \rangle \\
\text{lev}(e) &\sqsubseteq \text{lev}(c) \\
\text{lev}(e) &= L \implies U' = \emptyset \\
\text{lev}(e) &= H \implies \text{noeval}(c) \land \text{noloop}(c) \\
\text{land}U' &= \text{upd}(c) \land ct' = \text{inputs}(c) \\
\langle st, U, ct \rangle &\xrightarrow{b(e,c)} \langle \text{lev}(e) : st, U \cup U', ct \sqcup ct' \rangle
\end{align*}
\]

Two new monitor events:
\[
\text{in}(x,v)
\]
\[
\text{out}(ch,e)
\]

And updates to two existing events.
Updated monitor semantics for enforcement in the presence of I/O

Two new monitor events:

\[ \text{in}(x, v) \]

\[ \text{out}(ch, e) \]

And updates to two existing events.

\[
\begin{align*}
&\text{lev}(st) = L \Rightarrow U' = U \setminus \{x\} \\
&\text{lev}(st) = H \Rightarrow U' = U \\
&\langle st, U, ct \rangle \xrightarrow{\text{in}(x, v)} \langle st, U', \text{lev}(st) \sqcup ct \rangle
\end{align*}
\]

\[
\begin{align*}
&\text{lev}(st) = L \Rightarrow \text{lev}(e) \sqsubseteq \text{lev}(ch) \\
&\langle st, U, ct \rangle \xrightarrow{\text{out}(ch, e)} \langle st, U, ct \rangle
\end{align*}
\]

\[
\begin{align*}
&\text{vars}(e) \cap U = \emptyset \quad \text{lev}(ct) \sqsubseteq \text{lev}(x) \\
&\langle st, U, ct \rangle \xrightarrow{\text{d}(x, e, m)} \langle st, U \cup \{x\}, ct \rangle
\end{align*}
\]

\[
\begin{align*}
&\text{lev}(e) \sqsubseteq \text{lev}(c) \quad \text{lev}(e) = L \Rightarrow U' = \emptyset \\
&\text{lev}(e) = H \Rightarrow \text{noeval}(c) \land \text{noloop}(c) \\
&\text{land}U' = \text{upd}(c) \land ct' = \text{inputs}(c)
\end{align*}
\]

\[
\langle st, U, ct \rangle \xrightarrow{b(e, c)} \langle \text{lev}(e) : st, U \cup U', ct \sqcup ct' \rangle
\]
Updated monitor semantics for enforcement in the presence of I/O

Two new monitor events:

\[ \text{in}(x, v) \]

\[ \text{out}(ch, e) \]

And updates to two existing events.

\[ \text{vars}(e) \cap U = \emptyset \quad \text{lev}(ct) \sqsubseteq \text{lev}(x) \]

\[ \langle st, U, ct \rangle \xrightarrow{\text{in}(x, v)} \langle st, U', \text{lev}(st) \sqcup ct \rangle \]

\[ \text{lev}(st) = L \implies U' = U \setminus \{x\} \]
\[ \text{lev}(st) = H \implies U' = U \]

\[ \text{lev}(st) = L \implies \text{lev}(e) \sqsubseteq \text{lev}(ch) \]

\[ \langle st, U, ct \rangle \xrightarrow{\text{out}(ch, e)} \langle st, U, ct \rangle \]

\[ \text{lev}(e) \sqsubseteq \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset \]
\[ \text{lev}(e) = H \implies \text{noleval}(c) \land \text{noloop}(c) \]
\[ \text{land}U' = \text{upd}(c) \land ct' = \text{inputs}(c) \]

\[ \langle st, U, ct \rangle \xrightarrow{b(e, c)} \langle \text{lev}(e) : st, U \cup U', ct \sqcup ct' \rangle \]
Updated monitor semantics for enforcement in the presence of I/O

Two new monitor events:

\[ \text{in}(x, v) \]

\[ \text{out}(ch, e) \]

And updates to two existing events.

\[ \text{lev}(st) = L \implies U' = U \setminus \{x\} \]
\[ \text{lev}(st) = H \implies U' = U \]
\[ \langle st, U, ct \rangle \xrightarrow{\text{in}(x,v)} \langle st, U', \text{lev}(st) \sqcup ct \rangle \]

\[ \text{lev}(st) = L \implies \text{lev}(e) \sqsubseteq \text{lev}(ch) \]
\[ \langle st, U, ct \rangle \xrightarrow{\text{out}(ch,e)} \langle st, U, ct \rangle \]

\[ \text{vars}(e) \cap U = \emptyset \quad \text{lev}(ct) \sqsubseteq \text{lev}(x) \]
\[ \langle st, U, ct \rangle \xrightarrow{\text{d}(x,e,m)} \langle st, U \cup \{x\}, ct \rangle \]

\[ \text{lev}(e) \sqsubseteq \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset \]
\[ \text{lev}(e) = H \implies \text{noeval}(c) \land \text{noloop}(c) \]
\[ \text{land}U' = \text{upd}(c) \land ct' = \text{inputs}(c) \]
\[ \langle st, U, ct \rangle \xrightarrow{b(e,c)} \langle \text{lev}(e) : st, U \cup U', ct \sqcup ct' \rangle \]
Updated monitor semantics for enforcement in the presence of I/O

Two new monitor events:

\[ \text{in}(x, v) \]

\[ \text{out}(ch, e) \]

And updates to two existing events.

\[ \text{lev}(st) = L \implies U' = U \setminus \{x\} \]
\[ \text{lev}(st) = H \implies U' = U \]
\[ \langle st, U, ct \rangle \xrightarrow{\text{in}(x, v)} \langle st, U', \text{lev}(st) \sqcup ct \rangle \]
\[ \text{lev}(st) = L \implies \text{lev}(e) \sqsubseteq \text{lev}(ch) \]
\[ \langle st, U, ct \rangle \xrightarrow{\text{out}(ch, e)} \langle st, U, ct \rangle \]
\[ \text{vars}(e) \cap U = \emptyset \quad \text{lev}(ct) \sqsubseteq \text{lev}(x) \]
\[ \langle st, U, ct \rangle \xrightarrow{d(x, e, m)} \langle st, U \cup \{x\}, ct \rangle \]
\[ \text{lev}(e) \sqsubseteq \text{lev}(c) \quad \text{lev}(e) = L \implies U' = \emptyset \]
\[ \text{lev}(e) = H \implies \text{noeval}(c) \land \text{noloop}(c) \]
\[ \text{land}U' = \text{upd}(c) \land ct' = \text{inputs}(c) \]
\[ \langle st, U, ct \rangle \xrightarrow{b(e, c)} \langle \text{lev}(e) : st, U \cup U', ct \sqcup ct' \rangle \]
Relation to localized delimited release

• All programs that satisfy localized delimited release (see: Sabelfeld and Sands, “Dimensions and Principles”) satisfy our definition of termination-insensitive security.

• The converse is not true. For instance:

  \[
  h' := 0; \text{if } h \text{ then } l := \text{declassify}(h') \text{ else } l := 0
  \]
An example

```
1 while l {
2    // get location from a high channel
3    input (user_location, H);
4    // make the location public
5    ploc := declassify (user_location);
6    output(ploc, L);
7    // Get new code that redraws the map
8    input (code, L);
9    eval (code) // run the code
10  }
```
(exit)