Enforcing Dynamic Interference Policy

Frédéric Prost

LIG, Université de Grenoble
B. P. 53, F-38041 Grenoble, France
Frederic.Prost@imag.fr

6th. of September 2011

Introduction

- Non-interference as mathematical ground for privacy analyses.

Introduction

- Non-interference as mathematical ground for privacy analyses.
- "Non-interference: who needs it ?"
- Numerous acceptable flows:
  - Cryptography;
  - Password check;
  - etc.

Introduction

- Non-interference as mathematical ground for privacy analyses.
- "Non-interference: who needs it ?"
- Numerous acceptable flows:
  - Cryptography;
  - Password check;
  - etc.

$\Longrightarrow$ Interference policies: data downgrading is allowed in some circumstances.

- The policy is generally static.

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;
  - etc.
- Challenge: to adapt non-interference to fit with dynamic evolution of privacy ?

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;
  - etc.
- Challenge: to adapt non-interference to fit with dynamic evolution of privacy ?
- In our framework we propose:
  1. A "security profile" for each operator: rewrite rules over privacy lattice.

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;
  - etc.
- Challenge: to adapt non-interference to fit with dynamic evolution of privacy ?
- In our framework we propose:
  1. A "security profile" for each operator: rewrite rules over privacy lattice.
  2. Rewrite rules may include actions modifying the policy.

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;
  - etc.
- Challenge: to adapt non-interference to fit with dynamic evolution of privacy ?
- In our framework we propose:
  1. A “security profile” for each operator: rewrite rules over privacy lattice.
  2. Rewrite rules may include actions modifying the policy.
  3. Definition of high/low bisimulation wrt dynamic policy.

Introduction

- A lot of every-day life scenarios involve dynamic evolution of data privacy levels.
  - Pay-per-view;
  - Sealed auctions;
  - etc.
- Challenge: to adapt non-interference to fit with dynamic evolution of privacy ?
- In our framework we propose:
  1. A "security profile" for each operator: rewrite rules over privacy lattice.
  2. Rewrite rules may include actions modifying the policy.
  3. Definition of high/low bisimulation wrt dynamic policy.
  4. Program safety verification by abstract execution on privacy levels.

Plan

Programming framework

Dynamic interference policy

Program safety w.r.t. DIP

Program Verification

Conclusion

WHILE programming language

- Minimalistic programming language:

$$
\begin{array}{lll}
v & ::= & x \mid 0 \mid 1 \mid \mathsf{tt} \mid \mathsf{ff} \mid \ldots \\
t, b & ::= & v \mid f(x_1, \ldots, x_n) \\
P & ::= & x \leftarrow t \mid P; P \mid \\
 & & \mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ P \mid \mathsf{while}\ b\ \mathsf{do}\ P \mid \mathsf{skip}
\end{array}
$$

- It can be seen as an intermediate language:

$$x := f(345, g(x_1, x_2)) \equiv (x_0 \leftarrow 345; x_3 \leftarrow g(x_2, x_3); x \leftarrow f(x_3))$$

- Natural semantics $\langle \mu, P \rangle \rightarrow_{\mathsf{os}} \langle \mu', P' \rangle$

Plan

Programming framework

Dynamic interference policy

Program safety w.r.t. DIP

Program Verification

Conclusion

Interference policy

- Program variables are attributed privacy levels.
- Privacy levels are elements of a lattice $\mathfrak{L}$.
- Interference policies are based on authorised behavior of operators.
- Usually it is done through types but it is too strict. ‘

Interference policy

- Program variables are attributed privacy levels.
- Privacy levels are elements of a lattice $\mathfrak{L}$.
- Interference policies are based on authorised behavior of operators.
- Usually it is done through types but it is too strict. '

$\Longrightarrow$ We use term rewriting system on privacy levels in order to deal with concrete privacy levels used at evaluation time.

Static interference policy

- For each operator f we consider f_{DIP}.
- $\Sigma_{DIP} = (\mathcal{V}_{DIP}, \mathcal{V} \cup \Omega_{DIP} \cup \mathfrak{L})$
- Encryption policy, $\mathcal{D}$:

$$
\begin{aligned}
&\mathsf{encrypt}_{DIP}(\pi_{128}, \overline{x}) \to \pi_1 \\
&\mathsf{encrypt}_{DIP}(\pi_{256}, \overline{x}) \to \bot \\
&\mathrm{SPY}_{DIP} \to \bot \\
&\mathrm{PIN}_{DIP} \to \top \\
&\ldots
\end{aligned}
$$

- In the program: $\mathrm{SPY} \leftarrow \mathsf{encrypt}(K, \mathrm{PIN})$
  The security level of $\mathsf{encrypt}(K, \mathrm{PIN})$ is computed using rules of $\mathcal{D}$.

Dynamicity

- Privacy levels may change during computation.

Dynamicity

- Privacy levels may change during computation.
- Rewriting rules with actions : $l \to r; \mathsf{a}$

$$\mathsf{a} ::= x \mapsto \pi \mid \overline{x} \mapsto \pi \mid \overline{x} \mapsto \overline{y} \mid x \mapsto \overline{y} \mid \mathsf{a}; \mathsf{a}$$

Dynamicity

- Privacy levels may change during computation.
- Rewriting rules with actions : $l \rightarrow r; \mathsf{a}$

$$\mathsf{a} ::= x \mapsto \pi \mid \overline{x} \mapsto \pi \mid \overline{x} \mapsto \overline{y} \mid x \mapsto \overline{y} \mid \mathsf{a}; \mathsf{a}$$

- The interference policy changes through the evaluation of operator security level computation:

$$\langle t[\sigma(l)], \mathcal{D} \rangle \rightsquigarrow \langle t[\sigma(r)], \mathcal{D}' \rangle$$

Three strikes, out

- Aim: to control the number of password checks before blocking the system.
- Program command: ckpwd(g, PWD)

Three strikes, out

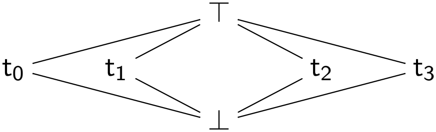
- Aim: to control the number of password checks before blocking the system.
- Program command: ckpwd(g, PWD)
- For each operator f of arity n we consider f_{DIP} of arity $2n$.

$\Longrightarrow$ distinction between the name of a program variable and its privacy level.

- Privacy level of ckpwd(g, PWD) is computed by the evaluation of:

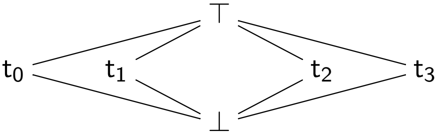
$$\mathrm{ckpwd}_{DIP}(\pi_g, g, \pi_{pwd}, \mathrm{PWD})$$

Three strikes, out



$$
\begin{array}{ll}
\mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_0, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_1 & \mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_1, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_2 \\
\mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_2, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_3 & \mathsf{ckpwd}(\bot, (\overline{g}, \mathsf{t}_3, \overline{p}) \to \top \\
\mathsf{ckok}(\overline{x}, \overline{y}) \to \bot; \overline{y} \to \mathsf{t}_0 & g \to \bot \\
\mathrm{PIN}_1 \to \mathsf{t}_0 & \mathrm{PIN}_2 \to \mathsf{t}_0 \\
\ldots &
\end{array}
$$

Three strikes, out



$\mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_0, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_1$ $\quad$ $\mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_1, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_2$

$\mathsf{ckpwd}(\bot, \overline{g}, \mathsf{t}_2, \overline{p}) \to \bot; \overline{p} \to \mathsf{t}_3$ $\quad$ $\mathsf{ckpwd}(\bot, (\overline{g}, \mathsf{t}_3, \overline{p}) \to \top$

$\mathsf{ckok}(\overline{x}, \overline{y}) \to \bot; \overline{y} \to \mathsf{t}_0$ $\quad$ $g \to \bot$

$\mathrm{PIN}_1 \to \mathsf{t}_0$ $\quad$ $\mathrm{PIN}_2 \to \mathsf{t}_0$

$\ldots$

In the program the evaluation modifies the DIP:
if $\mathsf{ckpwd}(g, \mathrm{PIN}_1)$ then blah else next_try

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \to \pi$ in $\mathcal{D}$.

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \rightarrow \pi$ in $\mathcal{D}$.
2. For each rule $l \rightarrow r$ such that l is in $\mathcal{V}$ then r is in $\mathfrak{L}$.

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \to \pi$ in $\mathcal{D}$.
2. For each rule $l \to r$ such that l is in $\mathcal{V}$ then r is in $\mathfrak{L}$.
3. $\mathcal{D}$ introduces no junk into $\mathfrak{L}$. I.e., for all ground terms, t, over $\Sigma \cup \mathfrak{L}$, the normal form of t, is in $\mathfrak{L}$.

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \to \pi$ in $\mathcal{D}$.
2. For each rule $l \to r$ such that l is in $\mathcal{V}$ then r is in $\mathfrak{L}$.
3. $\mathcal{D}$ introduces no junk into $\mathfrak{L}$. I.e., for all ground terms, t, over $\Sigma \cup \mathfrak{L}$, the normal form of t, is in $\mathfrak{L}$.
4. $\mathcal{D}$ introduces no confusion into $\mathfrak{L}$. I.e., $\forall \tau_1, \tau_2 \in \mathfrak{L}, \tau_1 \neq \tau_2 \implies \tau_1 \not\xrightarrow{*} \tau_2$.

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \to \pi$ in $\mathcal{D}$.
2. For each rule $l \to r$ such that l is in $\mathcal{V}$ then r is in $\mathfrak{L}$.
3. $\mathcal{D}$ introduces no junk into $\mathfrak{L}$. I.e., for all ground terms, t, over $\Sigma \cup \mathfrak{L}$, the normal form of t, is in $\mathfrak{L}$.
4. $\mathcal{D}$ introduces no confusion into $\mathfrak{L}$. I.e., $\forall \tau_1, \tau_2 \in \mathfrak{L}, \tau_1 \neq \tau_2 \implies \tau_1 \not\overset{*}{\to} \tau_2$.
5. functions in Σ are monotonic w.r.t. privacy levels: $\forall \pi_i, \pi_i' \in \mathfrak{L}, \pi_i \sqsubseteq \pi_i' \implies nf^{\mathcal{D}}(f(\pi_1, \ldots, \pi_n)) \sqsubseteq nf^{\mathcal{D}}(f(\pi_1', \ldots, \pi_n'))$.

Dynamic interference policy

Definition

A DIP, $\mathcal{D}$, is a confluent terminating rewrite system with actions with:

1. For every $x \in \mathcal{V}$ there is a rule $x \to \pi$ in $\mathcal{D}$.
2. For each rule $l \to r$ such that l is in $\mathcal{V}$ then r is in $\mathfrak{L}$.
3. $\mathcal{D}$ introduces no junk into $\mathfrak{L}$. I.e., for all ground terms, t, over $\Sigma \cup \mathfrak{L}$, the normal form of t, is in $\mathfrak{L}$.
4. $\mathcal{D}$ introduces no confusion into $\mathfrak{L}$. I.e., $\forall \tau_1, \tau_2 \in \mathfrak{L}, \tau_1 \neq \tau_2 \implies \tau_1 \not\xrightarrow{*} \tau_2$.
5. functions in Σ are monotonic w.r.t. privacy levels: $\forall \pi_i, \pi'_i \in \mathfrak{L}, \pi_i \sqsubseteq \pi'_i \implies nf^{\mathcal{D}}(f(\pi_1, \ldots, \pi_n)) \sqsubseteq nf^{\mathcal{D}}(f(\pi'_1, \ldots, \pi'_n))$.

- $\langle t, \mathcal{D} \rangle \rightsquigarrow^* \langle \pi^{\mathcal{D}}(t), \overline{\mathcal{D}}^t \rangle$

Privacy level of a term wrt $\mathcal{D}$

- to compute the privacy level of $f(x, y)$ we consider

$$t = f_{DIP}(nf^{\mathcal{D}}(x), x, nf^{\mathcal{D}}(y), y)$$

- The evaluation of this term in $\mathcal{D}$ gives the privacy level and a new interference policy:

$$\langle t, \mathcal{D} \rangle \rightsquigarrow^* \langle \pi^{\mathcal{D}}(t), \overline{\mathcal{D}}^t \rangle$$

Plan

Program safety

- Traditionally: a program is safe if every modification of a value above π cannot be observed below π:

$$\langle \mu_1, P \rangle \rightarrow^*_{\mathsf{os}} \mu'_1$$
$$\langle \mu_2, P \rangle \rightarrow^*_{\mathsf{os}} \mu'_2$$
$$\mu'_1 \equiv_\pi \mu'_2$$

- What to do with the policy:

$$\mathsf{encrypt}(\pi_{1024}, \overline{K}, \top, \overline{p}) \rightarrow \bot$$

making possible program as:

$$\textsc{spy} \leftarrow \mathsf{encrypt}(\mathit{key}_{1024}, \textsc{pin})$$

Program safety

- Traditionally: a program is safe if every modification of a value above π cannot be observed below π:

$$\begin{array}{l} \langle \mu_1, P\rangle \rightarrow^*_{\mathsf{os}} \mu'_1 \\ \langle \mu_2, P\rangle \rightarrow^*_{\mathsf{os}} \mu'_2 \\ \mu'_1 \equiv_\pi \mu'_2 \end{array}$$

- What to do with the policy:

$$\mathsf{encrypt}(\pi_{1024}, \overline{K}, \top, \overline{p}) \rightarrow \bot$$

making possible program as:

$$\mathrm{SPY} \leftarrow \mathsf{encrypt}(key_{1024}, \mathrm{PIN})$$

$\Longrightarrow$ Use an alternate operational semantics in which declared leaks are treated specificaly.

- Notion of declassified operational semantics.

$$\langle \mu_1, P\rangle \xRightarrow{\mu_d} \langle \mu'_1, P'\rangle$$

declassifying terms

- A term is declassifying if its privacy level is lower than one of its arguments.
- Such terms will be subjected to specific rules in the declassified operational semantics.

Definition (Declassifying terms and assignments)

$t = f(x_1, \ldots, x_n)$ is declassifying wrt $\mathcal{D}$, written $\mathcal{D} \vdash f(x_1, \ldots, x_n) \downarrow$ if:

$$\pi^{\mathcal{D}}(t) \sqsubseteq (\bigsqcup_{i=1}^{n} \pi^{\mathcal{D}}(t_i))$$

Declassified evaluation

- $\langle P, \mu \rangle \xrightarrow{\mu_d} \langle P', \mu', \mathcal{D}' \rangle$
- Declassifying assignment:

$$\frac{\begin{array}{l}\mathcal{D} \vdash f_{DIP}((\pi^{\mathcal{D}}(x), x) \downarrow \\ \quad [\![f(x)]\!] \mu_d = v\end{array} \qquad \langle f(\pi^{\mathcal{D}}(x), x), \mathcal{D} \rangle \rightsquigarrow^* \langle \pi, \overline{\mathcal{D}}^{f(\pi^{\mathcal{D}}(x), x)} \rangle}{\langle y \leftarrow f(x), \mu, \mathcal{D} \rangle \xrightarrow{\mu_d} \langle \mathsf{skip}, \mu[y := v], \overline{\mathcal{D}}^{f(\pi^{\mathcal{D}}(x), x)} \rangle} [\mathsf{AS}$$

High/low bisimulation and DIPs

Definition (Bisimulation)

A π-bisimulation is a symetric relation $\mathcal{R}$ on couples formed by a program and a DIP such that:

If $\langle P_1, \mathcal{D}_1 \rangle \mathcal{R} \langle P_2, \mathcal{D}_2 \rangle$ and

$\langle \mu_1, P_1, \mathcal{D}_1 \rangle \stackrel{\mu_1}{\Longrightarrow} \langle \mu_1', P_1', \mathcal{D}_1' \rangle$

and $\mu_1 \simeq_{\pi}^{\mathcal{D}_1 \sqcup \mathcal{D}_2} \mu_2$

High/low bisimulation and DIPs

Definition (Bisimulation)

A π-bisimulation is a symetric relation $\mathcal{R}$ on couples formed by a program and a DIP such that:

$$\begin{array}{l} \text{If } \langle P_1, \mathcal{D}_1\rangle \mathcal{R} \langle P_2, \mathcal{D}_2\rangle \text{ and} \\ \\ \langle \mu_1, P_1, \mathcal{D}_1\rangle \xRightarrow{\mu_1} \langle \mu_1', P_1', \mathcal{D}_1'\rangle \\ \\ \text{and } \mu_1 \simeq_\pi^{\mathcal{D}_1 \sqcup \mathcal{D}_2} \mu_2 \end{array} \implies \begin{cases} \exists P_2', \mathcal{D}_2' \text{ and } \mu_2' \text{ s.t.} \\ \\ \langle \mu_2, P_2, \mathcal{D}_2\rangle \xRightarrow{\mu_1}{}^{*} \langle \mu_2', P_2', \mathcal{D}_2'\rangle \\ \\ \text{and } \mu_1' \simeq_\pi^{\mathcal{D}_1' \sqcup \mathcal{D}_2'} \mu_2' \\ \\ \text{and } \langle P_1', \mathcal{D}_1'\rangle \mathcal{R} \langle P_2', \mathcal{D}_2'\rangle \end{cases}$$

Program safety w.r.t. a DIP

- The union of two π-bisimulation is a π-bisimulation.
- The biggest π-bisimulation is written $\simeq$ and is the union of all π-bisimulation.

Program safety w.r.t. a DIP

- The union of two π-bisimulation is a π-bisimulation.
- The biggest π-bisimulation is written $\simeq$ and is the union of all π-bisimulation.

Definition (Safe program)

A program P is safe with relation DIP $\mathcal{D}$, written $\mathcal{D} \models P$, if for all privacy level π $\langle P, \mathcal{D} \rangle \simeq^{\pi} \langle P, \mathcal{D} \rangle$.

Plan

Abstract execution principle (1)

Abstract execution principle (1)

- Idea: to execute the program on $\mathfrak{L}$.
- An abstract memory record associates variables with their privacy levels.

Abstract execution principle (1)

- Idea: to execute the program on $\mathfrak{L}$.
- An abstract memory record associates variables with their privacy levels.
- Record of the highest privacy level encountered in if-then-else and while guards to avoid indirect leaks, e.g.:

$$\text{if } \mathrm{PIN} = 0 \text{ then while tt do skip else skip; } \mathrm{SPY} \leftarrow 0$$

- Check assignments wrt $\mathcal{D}$ and :

$$x \leftarrow f(\ldots) \text{ implies } \pi^{\mathcal{D}}(x) \subseteq (\pi^{\mathcal{D}}(f(\ldots)) \sqcup \pi_g)$$

raises a failure if the inequality is not satisfied.

Abstract execution principle (2)

- Moreover evaluation of terms modify the DIP.
- Problem: it is not possible to merge DIPs resulting from the branches of an if-then-else construct.

Abstract execution principle (2)

- Moreover evaluation of terms modify the DIP.
- Problem: it is not possible to merge DIPs resulting from the branches of an if-then-else construct.

$\Longrightarrow$ Creation of a DIP list recording DIP's for each execution paths.

- Fixpoint problem for the while construct.

Abstract execution principle (2)

- Moreover evaluation of terms modify the DIP.
- Problem: it is not possible to merge DIPs resulting from the branches of an if-then-else construct.

  $\Longrightarrow$ Creation of a DIP list recording DIP's for each execution paths.
- Fixpoint problem for the while construct.

  $\Longrightarrow$ Finite number of DIP lists.

Abstract execution result

Theorem

$$\exists \mathcal{L}.(\langle\{\langle \mathcal{D}, \bot\rangle\}, P\rangle \hookrightarrow^{*} \mathcal{L}) \implies \mathcal{D} \models P$$

Abstract execution result

Theorem

$$\exists \mathcal{L}.(\langle \{\langle \mathcal{D}, \bot \rangle\}, P\rangle \hookrightarrow^* \mathcal{L}) \implies \mathcal{D} \models P$$

- Converse implication does not hold:

$$\text{if } \textsc{pin} = 0 \text{ then } \textsc{spy} \leftarrow 1 \text{ else } \textsc{spy} \leftarrow 1$$

this safe program raises a failure in the abstract operational semantics.

Plan

Conclusion

- We have introduced dynamicity aspects in interference policies.
  - Definition of privacy levels.
  - Definition of program safety wrt DIP.
  - Sound program analysis algorithm.

$\Longrightarrow$ Encompass more real life scenarios than traditional noninterference based methods.

Conclusion

- We have introduced dynamicity aspects in interference policies.
  - Definition of privacy levels.
  - Definition of program safety wrt DIP.
  - Sound program analysis algorithm.

$\Longrightarrow$ Encompass more real life scenarios than traditional noninterference based methods.

- Dynamic aspects are limited to the evolution of data privacy levels.
- Extension in presence of concurrency ?