A Formal Specification of the MIDP 2.0 Security Model

Santiago Zanella Béguelin[1] Gustavo Betarte[2] Carlos Luna[2]

[1]Everest Project, INRIA Sophia Antipolis
INRIA–Microsoft Research Joint Laboratory

[2]InCo, Universidad de la República, Uruguay

Workshop on Formal Aspects in Security and Trust, 2006

Outline

What is a Mobile Device?

Defining characteristics

- portable
- scarce resources (compared with other platforms)
- communicated
- stores personal information
- subscribed to pay-per-use services

Some Examples

Cell Phones

Personal Digital Assistants

The Problem

What a secure mobile device should enforce:

- Data confidentiality and integrity
- Cost control
- Availability

...even in the presence of malicious applications

The Problem

What a secure mobile device should enforce:

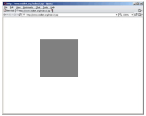
- Data confidentiality and integrity
- Cost control
- Availability

...even in the presence of malicious applications

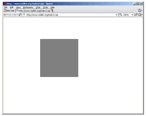
The Problem
A possible scenario

If the device supports loading of executable code after issuance...

A



B

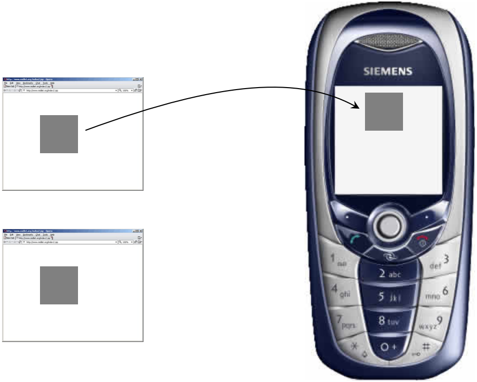


The Problem
A possible scenario

If the device supports loading of executable code after issuance...

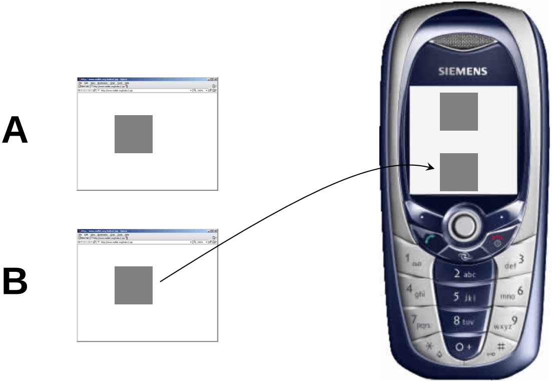
A

B



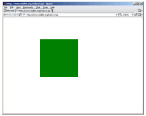
The Problem
A possible scenario

If the device supports loading of executable code after issuance...

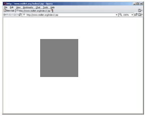


The Problem
A possible scenario

If the device supports loading of executable code after issuance...

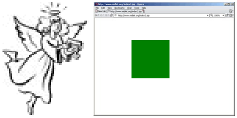



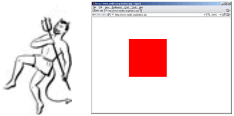
B



The Problem
A possible scenario

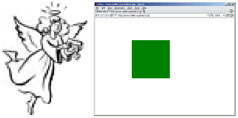
If the device supports loading of executable code after issuance...

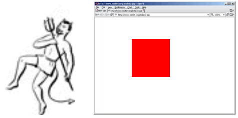




The Problem
A possible scenario

If the device supports loading of executable code after issuance...





First Solution
Removing the cause

Either

- Don’t allow users to download code

  but they love to do so
  (and it’s a big market opportunity)

- Don’t allow downloaded code to access sensitive APIs

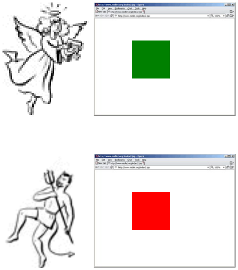
  but many useful applications must do so
  (e.g. synchronization, news push)

Roughly, MIDP 1.0 used this last solution (a sandbox model)

Second solution

Establish a security policy

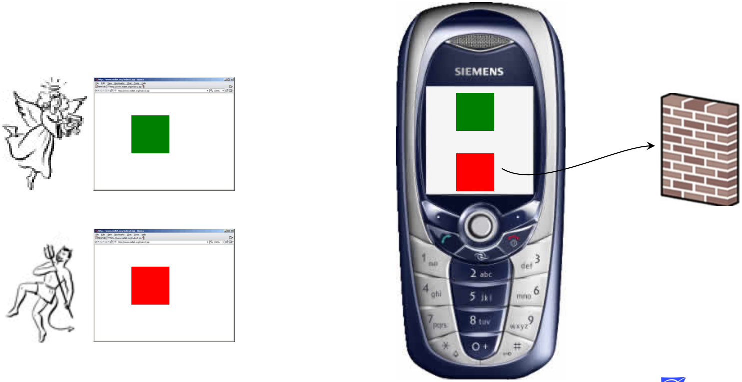
A security policy is a mapping from a set of properties that characterize code to a set of access permissions granted to that code



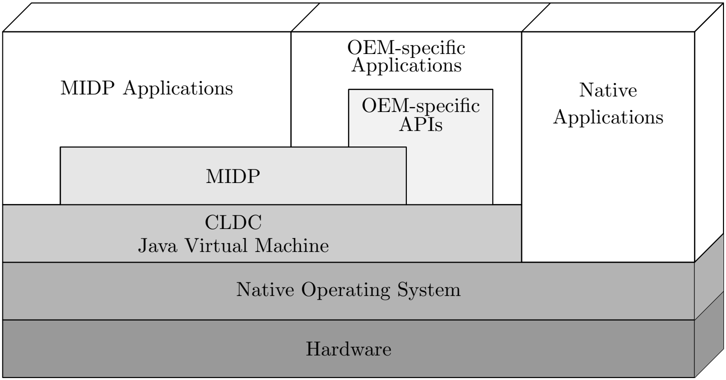
Second solution

Establish a security policy

A security policy is a mapping from a set of properties that characterize code to a set of access permissions granted to that code



Layered J2ME - MIDP architecture



- Users may only download MIDP applications
- MIDP applications access resources through restricted interface

MIDP Security Model

- In MIDP 1.0, *sandbox*-like model
- In MIDP 2.0, model based on *protection domains*

Protection Domain

- It's an abstraction of the context of execution of a piece of code
- Restricts access to sensitive functions
- In MIDP 2.0, each application belongs to a suite and each suite is bound to a unique Protection Domain

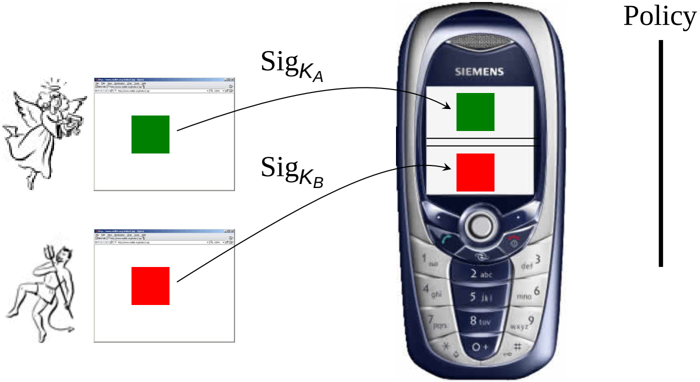
MIDP Security Model

- In MIDP 1.0, *sandbox*-like model
- In MIDP 2.0, model based on *protection domains*

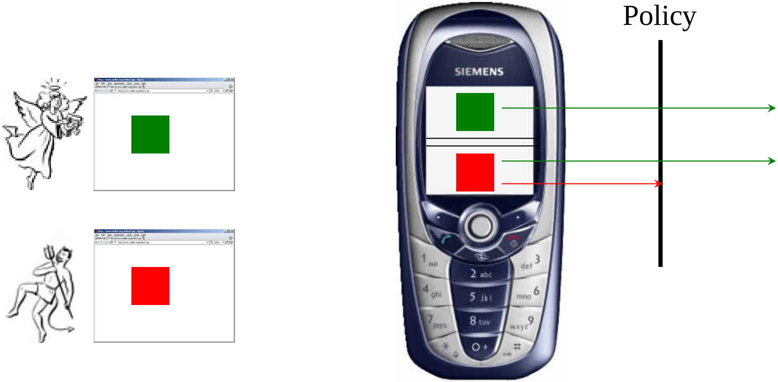
Protection Domain

- It's an abstraction of the context of execution of a piece of code
- Restricts access to sensitive functions
- In MIDP 2.0, each application belongs to a suite and each suite is bound to a unique Protection Domain

Protection Domains in Practice



Protection Domains in Practice



MIDP 2.0 Security Model

$$\text{Protected function} \longrightarrow \text{Permission}$$

A Protection Domain determines:

- A set of permissions granted unconditionally
- A set of permissions that could be granted with explicit user authorization, together with a mode that specifies its validity

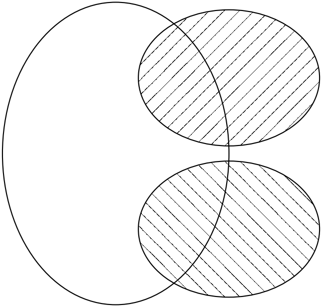
blanket	until the removal of the suite
session	for the current session
oneshot	for a single use

$$\mathit{oneshot} \leq_m \mathit{session} \leq_m \mathit{blanket}.$$

The specified mode is an upper bound

Permissions Acquired by a Suite

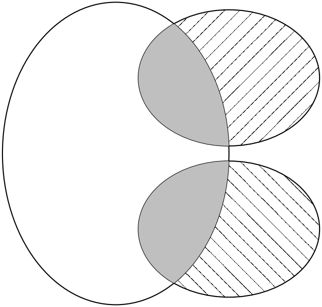
A suite declares at installation time the permissions it requires



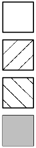
- ☐ Permissions required by the suite
- ▨ Permissions granted unconditionally
- ▧ Permissions granted by an explicit user authorization

Permissions Acquired by a Suite

A suite declares at installation time the permissions it requires



Acquired = Requested ∩
(Unconditionally granted ∪
Granted by user authorization)



Permissions required by the suite
Permissions granted unconditionally
Permissions granted by an explicit user authorization
Acquired permissions

New Problems

Issues

- Does the security model enforce the security policy?
- Do implementations conform to the model?
- How do other operations interfere with the model?

New Problems

Issues

- Does the security model enforce the security policy?
- Do implementations conform to the model?
- How do other operations interfere with the model?
- What is exactly the security model?

Outline

Remarks

- Formalized in the Calculus of Inductive Constructions
- Developed with the Coq proof assistant
- Abstract higher-order specification

The Calculus of Inductive Constructions

CIC

CIC is an extension of the simple-typed lambda calculus with:

- Polymorphic types $[(\lambda\, x\, .\, x) : A \to A]$
- Higher-order types $[A \to A : * : \Box]$
- Dependent types $[(\lambda\, a : A\, .\, f\ a) : (\forall\, a : A\, .\, B_a)]$

- Implemented in Coq
  Type checker + Proof assistant
- Can encode higher-order predicate logic
- Inductive definitions
- Curry-Howard isomorphism $\begin{array}{ccc} \text{types} & \leftrightarrow & \text{propositions} \\ \text{terms} & \leftrightarrow & \text{proofs} \end{array}$

Formalizing the state of the device

State components relevant to the security model:

- installed suites
- current session (if it exists)
  - current suite
  - permissions granted or revoked in *session* mode
- permissions granted or revoked for the session in *blanket* mode

$$State := \{ \begin{array}{ll} suite & : Suite \to Prop, \\ session & : option\ SessionInfo, \\ granted, revoked & : SuiteID \to Permission \to Prop \end{array} \}$$

Higher-order specification (notice predicates in the state)

Formalizing the state of the device

State components relevant to the security model:

- installed suites
- current session (if it exists)
  - current suite
  - permissions granted or revoked in *session* mode
- permissions granted or revoked for the session in *blanket* mode

$$
\mathit{State} := \left\{ \begin{array}{ll} \mathit{suite} & : \mathit{Suite} \rightarrow \mathit{Prop}, \\ \mathit{session} & : \mathit{option}\ \mathit{SessionInfo}, \\ \mathit{granted}, \mathit{revoked} & : \mathit{SuiteID} \rightarrow \mathit{Permission} \rightarrow \mathit{Prop} \end{array} \right\}
$$

Higher-order specification (notice predicates in the state)

Events

- Session start (*start*);
- Session end (*terminate*);
- Authorization request by the current suite (*request*);
- Suite installation (*install*);
- Suite removal (*remove*).

Their behavior is specified by means of pre- and postconditions.

Example (Session start)

$$
\begin{array}{l}
Pre\ s\ (start\ id) = \\
\quad s.session = None\ \wedge \exists\, ms : Suite, s.suite\ ms \wedge ms.id = id
\end{array}
$$

$$
\begin{array}{l}
Pos\ s\ s'\ r\ (start\ id) = r = None\ \wedge s \equiv_{session} s' \wedge \\
\quad \exists\, ses', s'.session = ses'\ \wedge ses'.id = id\ \wedge \\
\qquad \forall\, p : Permission, \neg ses'.granted\ p\ \wedge \neg ses'.revoked\ p
\end{array}
$$

State transition relation $\hookrightarrow$:

$$\frac{\neg Pre\ s\ e}{s \xhookrightarrow{e/None} s}\ npre \qquad \frac{Pre\ s\ e \quad Pos\ s\ s'\ r\ e}{s \xhookrightarrow{e/r} s'}\ pre$$

$s \xhookrightarrow{e/r} s'$: "the execution of the event e in state s results in a new state s' and produces a response r"

Sessions

$$s_0 \xhookrightarrow{\mathit{start\ id}/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{\mathit{terminate}/r_n} s_n$$

A session is determined by

- a suite identifier *id*
- an initial state s_0
- a sequence of steps $\langle e_i, s_i, r_i \rangle$ $(i = 1, \ldots, n)$ s.t.

1. $e_1 = \mathit{start\ id}$
2. $\mathit{Pre}\ s_0\ e_1$
3. $\forall\, i \in \{2, \ldots, n-1\}, e_i \neq \mathit{terminate}$
4. $e_n = \mathit{terminate}$
5. $\forall\, i \in \{1, \ldots, n\}, s_{i-1} \xhookrightarrow{e_i/r_i} s_i$

Sessions

$$s_0 \xhookrightarrow{\mathit{start\ id}/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{\mathit{terminate}/r_n} s_n$$

A session is determined by

- a suite identifier *id*
- an initial state s_0
- a sequence of steps $\langle e_i, s_i, r_i \rangle$ $(i = 1, \ldots, n)$ s.t.

1. $e_1 = \mathit{start\ id}$
2. $\mathit{Pre}\ s_0\ e_1$
3. $\forall\ i \in \{2, \ldots, n-1\}, e_i \neq \mathit{terminate}$
4. $e_n = \mathit{terminate}$
5. $\forall\ i \in \{1, \ldots, n\}, s_{i-1} \xhookrightarrow{e_i/r_i} s_i$

Sessions

$$s_0 \xhookrightarrow{start\ id/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{terminate/r_n} s_n$$

A session is determined by

- a suite identifier *id*
- an initial state s_0
- a sequence of steps $\langle e_i, s_i, r_i \rangle$ ($i = 1, \ldots, n$) s.t.

1. $e_1 = start\ id$
2. $Pre\ s_0\ e_1$
3. $\forall\ i \in \{2, \ldots, n-1\}, e_i \neq terminate$
4. $e_n = terminate$
5. $\forall\ i \in \{1, \ldots, n\}, s_{i-1} \xhookrightarrow{e_i/r_i} s_i$

Sessions
Inductive definition

$$s_0 \xhookrightarrow{start\ id/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{terminate/r_n} s_n$$

$$\frac{Pre\ s_0\ (start\ id) \quad s_0 \xhookrightarrow{start\ id/r_1} s_1}{PSession\ s_0\ ([\,] \frown \langle start\ id, s_1, r_1\rangle)}\ pses\_start$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad e \neq terminate \quad last.s \xhookrightarrow{e/r} s'}{PSession\ s_0\ (ss \frown last \frown \langle e, s', r\rangle)}\ pses\_app$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad last.s \xhookrightarrow{terminate/r} s'}{Session\ s_0\ (ss \frown last \frown \langle terminate, s', r\rangle)}\ ses\_term$$

Sessions
Inductive definition

$$s_0 \xhookrightarrow{start\ id/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{terminate/r_n} s_n$$

$$\frac{Pre\ s_0\ (start\ id) \quad s_0 \xhookrightarrow{start\ id/r_1} s_1}{PSession\ s_0\ ([\,] \frown \langle start\ id, s_1, r_1\rangle)}\ pses\_start$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad e \neq terminate \quad last.s \xhookrightarrow{e/r} s'}{PSession\ s_0\ (ss \frown last \frown \langle e, s', r\rangle)}\ pses\_app$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad last.s \xhookrightarrow{terminate/r} s'}{Session\ s_0\ (ss \frown last \frown \langle terminate, s', r\rangle)}\ ses\_term$$

Sessions
Inductive definition

$$s_0 \xhookrightarrow{start\ id/r_1} s_1 \xhookrightarrow{e_2/r_2} s_2 \xhookrightarrow{e_3/r_3} \cdots \xhookrightarrow{e_{n-1}/r_{n-1}} s_{n-1} \xhookrightarrow{terminate/r_n} s_n$$

$$\frac{Pre\ s_0\ (start\ id) \quad s_0 \xhookrightarrow{start\ id/r_1} s_1}{PSession\ s_0\ ([\,] \frown \langle start\ id, s_1, r_1\rangle)}\ pses\_start$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad e \neq terminate \quad last.s \xhookrightarrow{e/r} s'}{PSession\ s_0\ (ss \frown last \frown \langle e, s', r\rangle)}\ pses\_app$$

$$\frac{PSession\ s_0\ (ss \frown last) \quad last.s \xhookrightarrow{terminate/r} s'}{Session\ s_0\ (ss \frown last \frown \langle terminate, s', r\rangle)}\ ses\_term$$

Outline

Methodology

- The formal specification defines a theory
- Properties of the security model are theorems
- We state and prove some of these theorems with the help of Coq

Some Proved Theorems

Proofs are omitted, sorry

State validity is an invariant

$$\forall\ (s\ s' : State)\ (e : Event)\ (r : Response)$$
$$Valid\ s \rightarrow s \xhookrightarrow{e/r} s' \rightarrow Valid\ s'$$

A state is valid if (among other things)

- Suite identifiers are unique;
- The current suite is an installed suite;
- Granted permissions are consistent with corresponding protection domains and application descriptors;
- Permissions required as critical by a suite are not forbidden by its protection domain

Some Proved Theorems

More theorems

Revocation of permissions is correctly enforced

Whenever a permission is revoked in *session* mode, subsequent authorization requests are refused

Generalization of invariants

- Sufficient and necessary conditions for invariants
- Theorem: one-step invariants remain true once established

Outline

Why Should We Care?

Remarks

- We have a higher-order specification
- Transition relation defined implicitly
- Coq program extraction mechanism cannot be used

What is the pay off of refinement?

- An executable prototype
- An oracle for testing
- Test case extraction (*black box testing*)

Data Refinement

- For each type T, a concrete type $\overline{T}$ is defined
- $x \sqsubseteq \overline{x}$ is read "x is refined by $\overline{x}$"

Example (Predicates as lists)

Let $P : A \rightarrow Prop$ and $l : list\ \overline{A}$, then $l \sqsubseteq P$ iff

$$(\forall\, a, P\ a \rightarrow \exists\, \overline{a}, \overline{a} \in l \ \wedge\ a \sqsubseteq \overline{a}) \ \wedge$$
$$(\forall\, \overline{a}, \overline{a} \in l \rightarrow \exists\, a, P\ a \ \wedge\ a \sqsubseteq \overline{a})$$

Whenever $A = \overline{A}$ this simplifies to

$$\forall\, a, P\ a \leftrightarrow a \in l$$

Concrete State

$$\overline{State} := \{ \begin{array}{ll} suite & : list\ \overline{Suite}, \\ session & : option\ \overline{SessionInfo}, \\ granted, revoked & : SuiteID \rightarrow list\ Permission \end{array} \}$$

Operation Refinement

$$
\begin{tikzcd}
State & s \arrow[rr, dashed, "\hookrightarrow"] \arrow[d, "\sqsubseteq"'] & & s' \arrow[d, "\sqsubseteq"] \\
\overline{State} & \overline{s} \arrow[rr, "interp"] & & \overline{s'}
\end{tikzcd}
$$

For every state $\overline{s}'$ and response r computed by *interp* there must exist a corresponding abstract state s' refined by $\overline{s}'$ reachable from s by $\hookrightarrow$ with the same response

$$
\begin{aligned}
&\forall\, (s : State)\ \left(\overline{s} : \overline{State}\right)\ (e : Event)\ \left(\overline{e} : \overline{Event}\right)\ (r : Response), \\
&\quad s \sqsubseteq \overline{s} \rightarrow e \sqsubseteq \overline{e} \rightarrow \\
&\quad \text{let } \left(\overline{s'}, r\right) := interp\ \overline{s}\ \overline{e} \text{ in } \exists\, s' : State, s' \sqsubseteq \overline{s'}\ \wedge\ s \xhookrightarrow{e/r} s'
\end{aligned}
$$

Operation Refinement

$$
\begin{tikzcd}
State & s \arrow[r, dashed, hook, "\hookrightarrow"] \arrow[d, "\sqsubseteq"'] & s' \arrow[d, "\sqsubseteq"] \\
\overline{State} & \overline{s} \arrow[r, "interp"] & \overline{s'}
\end{tikzcd}
$$

For every state $\overline{s}'$ and response r computed by *interp* there must exist a corresponding abstract state s' refined by $\overline{s}'$ reachable from s by $\hookrightarrow$ with the same response

$$
\begin{aligned}
&\forall\,(s : State)\left(\overline{s} : \overline{State}\right)(e : Event)\left(\overline{e} : \overline{Event}\right)(r : Response),\\
&\quad s \sqsubseteq \overline{s} \to e \sqsubseteq \overline{e} \to\\
&\quad \text{let } \left(\overline{s'}, r\right) := interp\ \overline{s}\ \overline{e} \text{ in } \exists\, s' : State, s' \sqsubseteq \overline{s'}\ \wedge\ s \xhookrightarrow{e/r} s'
\end{aligned}
$$

Main Contributions

- The first formalization of the MIDP 2.0 security model
- Formal machine-checked verification of the model
- Investigated some aspects unclear in the informal specification
- A refinement methodology

The complete development in Coq may be obtained from

`http://www-sop.inria.fr/everest/personnel/Santiago.Zanella/MIDP`

Ongoing and Future Work

- We have not completed a full refinement
- Relax hypothesis assumed about the model
  - More than one active suite
  - Dynamic security policies in Protection Domains
- Consider extensions to the existing model
  - Hierarchical permissions
  - Multiplicities (Besson et al. – ESORICS'06)

Thank you!

Additional Information

`http://www-sop.inria.fr/everest/personnel/Santiago.Zanella/MIDP`

Santiago.Zanella@inria.fr