Security for Mobile Objects

Will Harwood
Introduction

*A mobile object is some code*
Introduction

A mobile object is some code that carries a state
Introduction

A mobile object is some code that carries a state that lives on a host
Introduction

A mobile object is some code that carries a state that lives on a host that visits places.
Introduction

A mobile object is some code that carries a state that lives on a host that visits places which is let in when trusted.
Introduction

A mobile object is some code that carries a state that lives on a host that visits places which is let in when trusted and barred when untrusted.
Introduction

A mobile object is some code that carries a state that lives on a host that visits places which is let in when trusted and barred when untrusted and will refuse to go to untrustworthy places.
Communication

Mobile objects can talk to their friends
Communication

*Mobile objects can talk to their friends but only by co-operation of the hosts*
Communication

*Mobile objects can talk to their friends but only by co-operation of the hosts. In fact they expose their keys to their host.*
Mobile objects can talk to their friends but only by co-operation of the hosts. In fact they expose their keys to their host. Which means a host can fake communication from an object even when the object has left.
Movement, Secrets and Promises

*Mobile objects carry secrets and promises between hosts*
Movement, Secrets and Promises

Mobile objects carry secrets and promises between hosts
State variables constitute communication channels between hosts
Movement, Secrets and Promises

Mobile objects carry secrets and promises between hosts
State variables constitute communication channels between host
Use of these channels is governed by a policy
Movement, Secrets and Promises

Mobile objects carry secrets and promises between hosts
State variables constitute communication channels between hosts
Use of these channels is governed by a policy
Mobile objects carry secrets and promises between hosts
State variables constitute communication channels between host
Use of these channels is governed by a policy
What's Really Going On?
What's Really Going On?

code

policy

state
What's Really Going On?

code

policy

state

© 1998 ANSA Consortium
What's Really Going On?

code

policy

state
What's Really Going On?

code

policy

hosts

vars

state
Access Control Matrices

- Access control matrices define a set of secure transitions in a system
- A system is secure if it starts in a secure initial state and only applies secure transitions
- A state is secure if it was obtained starting from the secure initial state and applying a sequence of secure transitions
Secure Transition System

In a secure state and transition pre condition holds, go to new secure state.
Problem

- In general it is not possible to look at a given state and know that it is secure without looking at how that state was constructed.
Solutions

- Find a way of imposing the pre conditions without the consent of the hosts
  - cryptographically enforced pre conditions
- Translate pre conditions into post conditions that can be checked at subsequent hosts
  - integrity post conditions
  - history conditions
Cryptographic Enforcement

- Read access to variables can be restricted by encryption of the variables contents. Read access is only granted to those hosts with keys.
- Read access can be separated from write access by use of asymmetric encryption.
Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)
Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)
Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - can use asymmetric encryption if the data is self validating (otherwise use separate signing)
Integrity Post Conditions

- Cryptographic signing can be used to ensure that only legitimate writers have altered a variable
  - Anonymously signed by any writer
  - Can use asymmetric encryption if the data is self validating (otherwise use separate signing)
Promises

- Hosts commit themselves to the value of a variable or group of variables by signing.
- Commitments may be conditional on other hosts commitments remaining valid.
Replays

- **Yesterday**
  - $V$ bound to 1234 signed Will

- **Today**
  - $V$ bound to 5678 signed Will
i.e. object is created with unique object number and signing is based on the static part of the object and the point that the signing happens in the objects history.
Unwinding

Alice → Bob → Cleo → Alice
Piggy-in-the-Middle

Alice → Bob

Bob → Cleo

Cleo → Clone

Clone → Alice
Best Branch

Alice → Bob → Cleo → Dave → Alice
Nested signing can be used to record where an object has been and the host can check that variable signing is consistent with the object’s history provided there are no branches in the object’s path.
Audit and Sequence Server

Audit and sequence server keeps track of choices
A Little Language of Behaviour

- Object movement is described by a finite state machine
  - itineraries: h;k, h+k, h[action], skip
- Each “state” corresponds to a visit to a host where the object performs actions
  - updates: !x, !x, ^x, <update, ..., update>
  - checks: h?x, h?<x,y,z>
  - conditional updates: H?x,k?y → <!!x,!y>
Information Gatherer
Information Gatherer

let itinerary = (P₁ + … + Pₙ)[^stack]; itinerary
+ Skip
in home; itinerary; home
Information Gatherer

let itinerary = (P_1 + \ldots + P_n)[^\text{stack}]; itinerary
+ Skip
in home; itinerary; home

<table>
<thead>
<tr>
<th>Stack</th>
<th>Home</th>
<th>P_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W</td>
<td>W</td>
<td></td>
</tr>
</tbody>
</table>
Lottery

Lottery Registry

Filled Out Ticket

request

Ticket Seller

Client

New Ticket
Lottery Behaviour

$\text{TS} ; \text{Client} ; \text{TR}[<\!\text{TN},!\text{Bet}>] ; \text{Client}[\text{TR}?<\text{TN},\text{Bet}>]$
Lottery Behaviour

\[ TS ; Client ; TR[<!TN,!Bet>] ; Client[TR?<TN,Bet>] \]

<table>
<thead>
<tr>
<th></th>
<th>TN</th>
<th>Bet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TS</strong></td>
<td>W</td>
<td></td>
</tr>
<tr>
<td><strong>Client</strong></td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td><strong>TR</strong></td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>
Payment Protocol

order, amount, purchaser’s account

Purchaser → Merchant

order, amount, purchaser’s account

Merchant → Purchaser’s Bank

order number, Merchant’s account

Purchaser’s Bank → Merchant’s Bank

Merchant’s Bank
## Payment Access Matrix

<table>
<thead>
<tr>
<th></th>
<th>order number</th>
<th>order amount</th>
<th>Purchaser’s account</th>
<th>Merchant’s account</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purchaser</strong></td>
<td>W</td>
<td>W</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td><strong>Merchant</strong></td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>W</td>
</tr>
<tr>
<td><strong>Purchaser’s</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Bank</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merchant’s</strong></td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td><strong>Bank</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Payment Behaviour

\[ P[<<!!order,!!amount,!!PurchaserAccount>>]; \]

\[ M[P?<order,amount,PurchaserAccount> \rightarrow
   <!order,!amount,!!OrderNumber,!!MerchantAccount>>]; \]

\[ PB[M?<order,account,OrderNumber,MerchantAccount>,
   P?<order,amount,PurchaserAccount> \rightarrow
   <!amount,!OrderNumber>>]; \]

\[ MB[M?<amount,OrderNumber,MerchantAccount>,
   PB?<amount,OrderNumber> \rightarrow
   <!amount,!OrderNumber>>]; \]

\[ M[MB?<amount,OrderNumber>] \]