Authorization Schemes & Intrusion Tolerance for Internet Applications

Yves Deswarte
LAAS-CNRS

Who are the intruders?

1: “Outsider”

- Authentication
- Authorization

2: User

- Authentication
- Authorization

3: Privileged User

- Authentication
- Authorization
Outsiders or Insiders: Privilege

- **Theft of privilege**: unauthorized increase in privilege
- **Abuse of privilege**: improper use of authorized operations

- **Outsider**: current privilege does not intersect considered domain
- **Insider**: current privilege intersects considered domain

Authorization

- **Contributes to protection**:  
  - Error detection/confinement  
  - Intrusion prevention/confinement

- **For Internet applications**:  
  - More flexible than "client-server" paradigm  
  - Contributes to privacy: personal information is disclosed only on a "need-to-know" basis
Authorization: reference monitor

Subjects = Active Entities (e.g., process, user, …)
Reference Monitor = Tamperproof Always Invoked Verified Correct
Objects = Information Containers (e.g., file, display, …)

Distributed Authorization? (1)

Reference Monitor

😊 small trusted area, easy administration
😊 bottleneck, single-point-of-failure
Distributed Authorisation? (2)

No bottleneck, no single-point-of-accidental-failure
Mutual trust between TCBs, consistency?

Authorization Scheme for DOOS

[Nicomette & Deswarte, 1997]
MAFTIA Authorization Server

Makes use of MAFTIA middleware:

- Non-confidential information is replicated (atomic multicast)
- Confidential information is shared securely (threshold crypto)
- Global consensus is achieved (majority voting / Byzantine agreement)
- Authorization proofs are distributed to local reference monitors (threshold signatures)

Local protection

- Internet applications: heterogeneous platforms => no modification of user workstations or even servers
- => no trusted security kernel, but JVM
- Local Reference Monitor =
  - A local dispatcher (not trusted)
  - A JavaCard -> security kernel
Security properties

- **Authorization server:**
  - **AS1:** The AS generates only valid authorization proofs
  - **AS2:** It is not possible to prevent AS from generating valid authorization proofs

- **Local reference monitors**
  - **RM1:** Only valid operations will be executed on a non-faulty host
  - **RM2:** It is not possible to prevent non-faulty hosts from executing valid operations

- Assumption 1: no network denial-of-service
- Assumption 2: Java Card tamperproof
**Architecture**

- **consensus, threshold signatures**

**Permissions**

1. Composite Operation Authorization Request

2. Permissions

3. Invocations (with capabilities)

4. Delegated Invocations

```
Permissions(O) = <{Perm(O,O'.m)*;Perm(O,cop)*}>_{SKas}
Perm(O, O'.m) = {O;O'.m(parC);cap(O;O'.m(parC))[vouch(O'.m)]}
Perm(O,cop) = {O;cop(parC);token(O;cop(parC))}
cap(O;O'.m(parC)) = <{O;O'.m;parC;nonce}>_{SKas,PKhost(O')}
vouch(O'.m) = <{Perm(O'.O'.m)*;Perm(O'.cop )*}>_{SKas}
token(O;cop(parC)) = <{O;cop;parC;nonce}>_{SKas,PKkas}
```
Cop Authorization Checks

Symbolic rights: corresponding to the authorization for an object to execute composite operations
(a simple method execution is a particular case of cop)

<table>
<thead>
<tr>
<th>ps1</th>
<th>fs2</th>
<th>f 3</th>
<th>p 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td></td>
<td>PF(this, PRINTER)</td>
<td>PF(FILE, this)</td>
</tr>
<tr>
<td>ps1</td>
<td></td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>fs2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symbolic right rules: to check authorization for composite operations

Permission creation rules: to generate permissions (capabilities and vouchers, tokens) to enable all methods executions

Architecture

Request for cop authorization

consensus, threshold signatures

Security Site

Security Site

Security Site

Security Site

dispatcher1

dispatcher2

Host1

Host2

Oracle O1

Oracle O2

Oracle O3

Oracle O4

PKm
PKas
SK1, PK1, K1

PKm
PKas
SK2, PK2, K2

JavaCard sk1

JavaCard sk2
The dispatcher
- asks sk1 to check the perm. signature
- stores the capabilities and vouchers
  \[ \text{cap}(O1; O3.m(parC)); \text{vouch}(O3.m) \]
- transmits O3.m and parC to O1

O1 invokes O3.m with \{parameters\}
dispenser1 retrieves \{\text{cap}(O1; O3.m(parC)); \text{vouch}(O3.m)\}, inserts it in the message to disp.2
• dispatcher2: sends \( \text{cap}(O1; O3.m(parC)) \) and \( \text{vouch}(O3.m) \) to sk
• sk2 checks the signatures of \( \text{cap} \) and \( \text{vouch} \)
• sk2 deciphers \( \text{cap} \) and returns the result to d2
• d2 checks \( \text{cap}(O1; O3.m(parC)) \) vs. invocation
• d2 stores the permissions from \( \text{vouch} \)
• d2 invokes O3.m

• O3 requests authorization for \( \text{cop5(par)} \)
• d2 retrieves \( \text{token}(O3; \text{cop5(parC)}) \)
• d2 inserts the token in the message to AS
**Example:**

**U → AS**: \( \text{Request}\{\text{SendPatientMedicalFile}(Pmf_1,V)\} \)

<table>
<thead>
<tr>
<th></th>
<th>HCP Role</th>
<th>( (Pmf(U)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U )</td>
<td>( S\text{P}mf{Pmf(U),\text{this}} )</td>
<td>read, write(^*), ( S\text{P}mf{\text{this.HCPRole}} )</td>
</tr>
<tr>
<td>DBS</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**AS → D_0**: \( \{U;\text{DBS.transferPatientMedicalFile}(Pmf_1,V); \text{Cap}(U;\text{DBS.transferPatientMedicalFile}(Pmf_1,V)); \text{Vouch}(\text{DBS.transferPatientMedicalFile}) \} \)
Example:

\[ U \mapsto DBS : \left\{ \begin{array}{l} \text{RMI}\left( DBS,\text{transferPatientMedicalfile}(Pmf_1, V) \right) ; \\
\text{Cap}(U, DBS,\text{transferPatientMedicalfile}(Pmf_1, V)); \text{Vouch}(DBS, \text{transferPatientMedicalfile}) \end{array} \right\} \]

\[ D_1 \mapsto J_1 : \left\{ \begin{array}{l} \text{Cap}(U, DBS,\text{transferPatientMedicalfile}(Pmf, V)); \text{Vouch}(DBS, \text{transferPatientMedicalfile}) \end{array} \right\} \]

\[ \text{Vouch}(DBS, ...) : \left\{ \begin{array}{l} DBS, \text{Pmf}_1, \text{readPatienMedicalfile} \text{Cap}(DBS, \text{Pmf}_1, \text{readPatientMedicalfile}) ; \\
DBS, MTA_1, \text{sendFilebyMail}(V, \text{Cap}(DBS, MTA_1, \text{sendFilebyMail}(V))) ; \\
\text{Vouch}(MTA_1, \text{sendFilebyMail}) \end{array} \right\} \]

Example:

\[ DBS \mapsto Pmf_1 : \left\{ \begin{array}{l} \text{RMI}(\text{Pmf}_1, \text{readPatientMedicalfile}) ; \\
\text{Cap}(DBS, \text{Pmf}_1, \text{readPatientMedicalfile}) \end{array} \right\} \]

\[ DBS \mapsto tf : \left\{ \begin{array}{l} \text{RMI}(tf, \text{write(contentof Pmf_1)}) \end{array} \right\} \]

\[ DBS \mapsto MTA_1 : \left\{ \begin{array}{l} \text{RMI}(DBS, \text{sendFilebyMail}(tf, V)) ; \\
\text{Cap}(DBS, MTA_1, \text{sendFilebyMail}(V)); \text{Vouch}(MTA_1, \text{sendFilebyMail}) \end{array} \right\} \]
Example:

$MTA_1 \rightarrow AS: \text{Request}(DeliverFilebyMail(*,V));\text{Token}(MTA_1;DeliverFilebyMail(*,V))$

$AS \rightarrow D_1: \{MTA_1;MTA_2.receive(*,V);\text{Cap}(MTA_1;MTA_2.receive(*,V));\text{Vouch}(MTA_2.receive)\}$

$MTA_1 \overset{6}{\rightarrow} MTA_2: \left\{ \begin{array}{c} \text{RMI}(MTA_2.receive(content\ of\ tf,V)); \\ \text{Cap}(MTA_1;MTA_2.receive(*,V));\text{Vouch}(MTA_2.receive) \end{array} \right\}$

Example:

$D_2 \rightarrow JC_2: \{\text{Cap}(MTA_1;MTA_2.receive(*,V));\text{Vouch}(MTA_2.receive)\}$

$MTA_2 \rightarrow VMailbox: \left\{ \begin{array}{c} \text{RMI}(VMailBox.mdeliver(content\ of\ tf)); \\ \text{Cap}(MTA_2;VMailbox.mdeliver(*)) \end{array} \right\}$
http://www.research.ec.org/maftia/