Verification of Access Control Systems

Masoud Koleini
University of Nottingham

November 2015
ACCESS CONTROL
AN OVERVIEW
Access Control is the process of mediating the requests for accessing data in a system and determining whether the request should be granted or denied.

Authentication and Authorization? included in the definition of access control
Some features of ACS

- Support for fine-grained and course-grained specifications
- Separation of duties (SoD)
  - Having more than one person to complete a task
- Delegation of authority
  - Possibility of passing authorization between subjects
- Least privilege
  - Each individual should have the minimum required permissions to perform a task
Implementation of an access control system can be divided into three design layers: [NIST, 2006]

1. Access control (security) policy
2. Access control model
3. Access control mechanism
Access control: policy

- High level description of conditions and rules under which user or process can access resources in the system
- Can be expressed in policy languages
  - XACML, SecPAL, DynPAL, PoliVer, …
<Rule RuleId="rul_self-appraisal-sam" Effect="Permit">
  <Description>Sam can view the resource</Description>
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            view
          </AttributeValue>
          <ActionAttributeDesignator>urn:oasis:names:tc:xacml:1.0:action:action-id</ActionAttributeDesignator>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Sam
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>

[Code (modified) from EMC website]
<Policy PolicyId="pol_self-appraisal" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            self-appraisal
          </AttributeValue>
          <ResourceAttributeDesignator>
            urn:oasis:names:tc:xacml:1.0:resource:resource-id
          </ResourceAttributeDesignator>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule RuleId="rul_self-appraisal-sam" Effect="Permit">
  </Rule>
</Policy>

[Code (modified) from EMC website]
XACML Engine

1. Policy
   - Policy Administration Point
   - 1. Policy

2. Access request
   - Access Requester
   - 2. Access request

3. request
   - Policy Enforcement Point
   - 3. request

4. Attribute query
   - Context Handler
   - 4. Attribute query

5a. Subject attributes
   - Policy Information Point
   - 5a. Subject attributes

5b. Environment attributes
   - Resources
   - 5b. Environment attributes

5c. Resource attributes
   - Environment
   - 5c. Resource attributes

6. Attribute
   - Policy Information Point
   - 6. Attribute

7. Request
   - Policy Decision Point
   - 7. Request

8. Response context
   - Context Handler
   - 8. Response context

9. Response [Permit/Deny/NA]
   - Policy Enforcement Point

10. Obligations
    - Obligations Service
    - 10. Obligations

[OASIS XACML V3.0 (simplified)]
Policy language: PoliVer

AccessControlSystem StudentInformationSystem

Predicate lecturer(agent: Agent), student(agent: Agent),
    demonstrator_of(demonstrator: Agent, student: Agent),
    higher(senior: Agent, junior: Agent), mark(student: Agent);

Action assignAsDem(d:Agent, s:Agent):
    {+demonstrator_of(d,s)} ← lecturer(user) & higher(d,s);

Action resignAsDem(d:Agent, s:Agent):
    {-demonstrator_of(d,s)} ← demonstrator_of(d,s) & user=d;

Action MarkStudent(s: Agent):
    {+mark(s)} ← lecturer(user) | demonstrator_of(user, s);

Read higher(s, j) ← true;
Read student(s) ← true;
Read lecturer(l) ← true;
Read mark(s) ← true;

End
Access control model

(2) Access control model
- Formal presentation of how policy is enforced
- Link between policy and mechanism

Some well-known models:
- Discretionary access control
- Mandatory Access Control
- Role Based Access Control
Discretionary access control (DAC)

- Owner of an object decides about permissions
- Security policy can be expressed in access control matrix

Example:

- File permissions in Unix-like operating systems
  - `rwxr-xr-x` Alice group1 file
- Access permissions in Role-based access control
DAC Weaknesses

- **Unauthorized information flow**
  
  granting read access is transitive: Alice grants Bob read access to her file, Bob copies the content of the file into another file of his own and allows some unauthorized users to access the content of Alice’s file.

- **End user complete control over permissions**

  Cannot reflect organization's security policy requirements

- **Malicious or flawed software**
Mandatory access control (MAC)

A central authority enforces authorization rules to the subjects and objects.

Example:
Bell-LaPadula multilevel security model [Bell and LaPadula, 1976]

Simple security:
- Subject $S$ can read object $O$ only if $L(S) \geq L(O)$

*-property:
- Subject $S$ can write object $O$ only if $L(S) \leq L(O)$
Mandatory access control (MAC)

The Chinese Wall Policy [Brewer and Nash, 1989]

COI = Conflict Of Interest
DS = Dataset
O = Object

Policy: There must be no information flow that results in conflict of interest

http://www.cs.cornell.edu/courses/CS5430/2012sp/chinWall.html
Mandatory access control (MAC)

The Chinese Wall Policy [Brewer and Nash, 1989]

**Read**: Subject $S$ can read object $O$ if for all $O'$ that $S$ had access to:
$DS(O) = DS(O')$, or
$CoI(O) \neq CoI(O')$

**Write**: Subject $S$ can write object $O$ if:
$S$ can read $O$, and
$S$ can not read any object $O'$ such that $DS(O) \neq DS(O')$

http://www.cs.cornell.edu/courses/CS5430/2012sp/chinWall.html
Mandatory access control (MAC)

The Chinese Wall Policy [Brewer and Nash, 1989]

What happens without write rule? Information flow

Assume that $S_2$ has read information $DS_2$. Then:
- $S_1$ reads $O_1$ in $DS_1$
- $S_1$ write information of $O_1$ into $O_4$ in $DS_3$
- $S_2$ reads that information from $O_4$

http://www.cs.cornell.edu/courses/CS5430/2012sp/chinWall.html
Role-based access control (RBAC)

This model is introduced to facilitate administration in medium to large systems [Ferraiolo and Kuhn, 1992]

Basic RBAC:

Core entities: users, roles, permissions
Relations: user assignment (UA),
          permission assignment (PA)
Other core components: session
RBAC reference model
Access control: mechanism

(3) Access control mechanism

- Policy enforcement mechanism using system’s acceptable structure

Example: Protection bits in UNIX-like systems (DAC)

- `rwxr-xr-x` `usr1` `grp1` `file1`
- `rwxr-xr--` `usr2` `grp2` `file2`

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usr1</td>
<td>rwx</td>
<td>r--</td>
</tr>
<tr>
<td>Usr2</td>
<td>r-x</td>
<td>rwx</td>
</tr>
</tbody>
</table>
Access control matrix

Formally, each element $P(s, o)$ in matrix represents access rights of subject $s$ over object $o$. [Harison et al. 1976]
ACCESS CONTROL
VERIFICATION
A security policy defines security requirements for a given system [Goguen and Meseguer, 1982]

- Multi-level security (MLS), discretionary access, mandatory access, information-flow, …

Verification is a technique that proves whether a policy holds on a system or not
Formal verification

Two main approaches of formal verification: [Cousot and Cousot, 2010]

1) Deductive methods
   produces formal mathematical correctness using theorem provers or proof assistances

2) Model checking
   Exhaustively explores of all possible behaviors over the state transition system that models program execution
Access control: dynamic vs. static

We can divide access control policies into two categories:

1. **Dynamic policies**: Permissions of the agents depend on the system state and can be changed by the actions of other agents. (PoliVer and DynPAL)

2. **Static policies**: Access decisions doesn’t change the state of the system. (Access Control Matrix)
Model-checking is a technique for verifying a property on a finite state model.

Model-checking contains three tasks: [Huth & Ryan, 2004]

1) Modeling the system
2) Coding the property in the specification language
3) Running the verification algorithm
Model: State Transition System

State transition graph (Kripke structure)

Unwind State Graph to obtain infinite tree

[Clarke et al, 2000]
PoliVer: Policy language to STS

PoliVer has a simple policy language based on read and write actions:

A write rule:

\[ \text{assignReviewer}(x, y, p) : \]
\[ \{ +\text{reviewer}(y, p) \} \leftarrow \text{chair}(x) \land \text{pcMember}(y) \land \neg\text{author}(y, p) \]

A read rule:

\[ \text{reviewer}(x, y, p) \leftarrow \text{chair}(x) \]
PoliVer: Policy language to STS

PoliVer creates ground rules: If Alice, Bob, and Paper or objects in the system, then we have:

\[
\text{assignReviewer}(\text{Alice, Bob, Paper}) : \{+\text{reviewer}(\text{Bob, Paper})\} \leftarrow \text{chair}(\text{Alice}) \land \text{pcMember}(\text{Bob}) \land \neg \text{author}(\text{Paper, Bob})
\]
In a conference paper review system:

The reviewers (\textit{rev}) of a paper should not be able to read other submitted reviews (\textit{submittedR}) before they submit their own.

\[
\{\text{chair}(c)^!, \neg \text{author}(a, p)^*, \text{submittedR}(b, p), \text{rev}(a, p), 
\neg \text{submittedR}(p, a)\} \rightarrow \{a\} : (\langle \text{review}(b, p) \rangle \land \neg \text{submittedR}(a, p) \text{ THEN } \{a, c\} : (\text{submittedR}(a, p)))
\]
∃a: There exists action a
∀p: For all values of p
The model-checking algorithm:

1) Create a state transition system from the policy
2) Start from all the states that satisfy the goal ($st$)
3) Use backward search to find all reachable states $st'$ that transit to $st$ based on policy rules.
4) If initial states $st_{init}$ reached ($st_{init} \subseteq st'\rangle$, output the strategy
5) If $st' \subseteq st$, then no strategy exists.
6) $st = st \cup st'$, then go to 3.
A DEMONSTRATION OF POLIVER
VERIFICATION OF INFORMATION FLOW IN TRUST-MANAGEMENT SYSTEMS
Trust management systems

In trust management systems:

- Authorization rules are specified in a machine-enforceable high-level policy language
- Users requesting access may submit credentials to support their requests

Probing attacks:

1. Attacker submits credentials together with requests
2. Attacker analysis system’s response to gain knowledge about confidential information
What can be detected about policy $A_0$?
A simple probing attack

Alice can detect "Svc says secretAg(Bob)!"
Challenges

1. What does “attack”, “detect”, etc. mean?*
2. What can the attacker (not) detect?
3. How do we automate?
probe

Client

Service

Policy

(A, q)

A_0 \cup A \vdash q?
Available probes

\[(A_1, q_1), (A_2, q_2), (A_3, q_3)\]

Attacker

Service

\[A_0 \cup A_1 \cup q?\]

Policy
Available probes

The attacker can’t distinguish A₀ and A₀' iff for all \((A, q) \in \text{Avail}\), \(A₀ \cup A \vdash q \iff A₀' \cup A \vdash q\)

Policies A₀ and A₀’ are observationally equivalent (\(A₀ \equiv A₀'\)) iff for all \((A, q) \in \text{Avail}\).
A query $p$ is detectable in $A_0$ iff
\[ \forall A_0' \in [A_0] \equiv : A_0' \vdash p \]
A query $p$ is **opaque** in $A_0$ iff

$$\exists A'_0 \in [A_0]: A'_0 \not\models p$$
Challenges

1. What does “attack”, “detect”, etc. mean?
2. What can the attacker (not) detect?*
3. How do we automate?
Is p opaque in A0?

- Policy language: Datalog clauses $p_0 \leftarrow p_1, \ldots, p_n$
- Input: $A_0, Avail, p$
- Output: $p$ is opaque in $A_0$ – or $p$ is detectable in $A_0$
- Sound
  - If algorithm says “$p$ is opaque,” then $p$ is really opaque!
- Complete
  - If $p$ is opaque, then algorithm finds it out
- Always **terminates** with a definite answer

A query $p$ is **opaque** in $A_0$ iff

$$\exists A'_0 \in [A_0] \equiv : A'_0 \not\models p$$
Example 1

What do we learn about $a$ and $b$ in $A_0$?

- $c.
- c \leftarrow a.
- c \leftarrow b.
- c \leftarrow a, b.$
What do we learn about \( c \) and \( a \) in \( A_0 \)?
Challenges

1. What does “attack”, “detect”, etc. mean?
2. What can the attacker (not) detect?
3. How do we automate?
General approach

To find if $q$ (in general a probe) is opaque in $A_0$:

- We will try to construct $A'_0$
- $A'_0$ is an opacity witness if
  - It masquerades as $A_0$ with respect to probes in $\text{Avail}$
  - But it makes $q$ negative
Conclusion

1. What are we protecting against?
   - Information flow properties: opacity and detectability

2. How much is leaked?
   - Algorithm for deciding opacity in Datalog policies
   - Tool with optimizations
Extra Slides