Model Driven Security: from UML Models to Access Control Infrastructures

Prof. David Basin
Jürgen Doser
Torrsten Lodderstedt
outline:

• problem domain / problem solving
• approach
• example
• bottom line
common software engineering process

requirements

design

implementation

verification

maintenance
security requirements

requirements

design

implementation

verification

maintenance
security requirements

- requirements
- design
- implementation
- implementation
- verification
- maintenance
development of security requirements

- very late ad hoc integration of implemented security mechanisms
- hard to keep track of security requirements through development

▶ different representations of system / security
problem solving

- one representation for system and security
- manual implementation is ambiguous: remove ambiguity
outline:

• problem domain / problem solving
• approach
• example
• bottom line
MDA: Model Driven Architecture

- specify system in abstract model
- apply transformation functions
- result: system specified in target platform
  e.g. EJB, .NET ...
  ( only architecture, no business logic )
simplified example:
poseidon UML Class Diagram to Java Class
MDS: Model Driven Security

• specify system and security together in an abstract model

• apply transformation functions

• result:

  security aware system
  specified in target platform
  e.g. EJB, .NET ...
  ( only architecture, no business logic )
... but how to build a model?

- modeling language

  - abstract syntax
  - concrete syntax
  - semantics
  - transformation functions
... but how to build a modeling language for MDS?
modeling language combination schema
modeling language combination schema

system design modeling
language
modeling language combination schema

system design modeling language

security modeling language
modeling language combination schema

system design modeling language
dialect
security modeling language
modeling language combination schema

system design modeling language  dialect  security modeling language

security design language
modeling language combination schema

system and security
modeled with security design language

<<Permission>>
permission_name
<<ClassMethodAction>>..........
outline:

• problem domain / problem solving
• approach
• example
• bottom line
example
Briefing with “M”

• I need mi6 to get a new system

• I like my cars: protect them with RBAC

• I want everything deployed as EJBs
Role Based Access Control

users

user_1

user_2

...

user_n

permissions

perm_1

perm_2

perm_3

perm_4

...

perm_n
Role Based Access Control

- users
  - user_1
  - user_2
  - ...
  - user_n

- roles
  - role_A
  - role_B

- permissions
  - perm_1
  - perm_2
  - perm_3
  - perm_4
  - ...
  - perm_n

UA → role_A → PA
EJB: Enterprise Java Beans

- **Enterprise JavaBeans™ (EJB)** is a managed, server-side component architecture for modular construction of enterprise applications.
<method-permission>
    <role-name>employee</role-name>
    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>findByPrimaryKey</method-name>
    </method>

    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>getEmployeeInfo</method-name>
    </method>

    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>updateEmployeeInfo</method-name>
    </method>
</method-permission>
mi6 - car access policy
modeling language combination schema

system design modeling language

dialect

security modeling language

security design language
modeling language combination schema

mi6UML

system design modeling language

dialect

security modeling language

SecureUML

security design language

Securemi6UML
system: protected resources
a car modeled with Securemi6UML

<<Entity>>

Car

manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]

change_wheel ( int )::void
refill_oil( int )::void
open( )::void
role and entity

<<Role>>
serviceAgent

<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]
change_wheel ( int )::void
refill_oil( int )::void
open( )::void
permissions as association class

<<Role>>
serviceAgent

<<Permission>>

<<Entity>>
Car

- manufacturer_name : String
- model_name : String
- mpg : int
- oil_level : int
- wheels : wheel[]
- change_wheel ( int )::void
- refill_oil( int )::void
- open( )::void
role: serviceAgent - permission I

```
<<Role>> serviceAgent

<<Permission>>

<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute

<<Entity>>

Car

manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel ( int )::void
refill_oil( int )::void
open( )::void
```
MaintainingAnalysis
<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute

MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute

<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]

change_wheel ( int )::void
refill_oil( int )::void
open( )::void

role: serviceAgent - permission II
role: simpleAgent - permission I

<<Role>> simpleAgent

<<Permission>>

<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute

<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]
change_wheel ( int )::void
refill_oil( int )::void
open( )::void

AdmireCar
CompositeAction

CompositeAction: read_specs

<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute
CompositeAction

CompositeAction: read_all

<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute
CompositeAction

CompositeAction: read_all

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute
action hierarchy

CompositeAction: read_specs

<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute

CompositeAction: read_all

<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute
mi6 - car access policy

mi6 agents

special agents

simple agents

service agents

special cars

common cars
```java
// extend Car entity

<<Entity>>

**Car**

- manufacturer_name : String
- model_name : String
- mpg : int
- oil_level : int
- wheels : wheel[]

- change_wheel ( int )::void
- refill_oil( int )::void
- open( )::void
- go_for_a_ride()::void
```
model

MaintainingAnalysis

<<ClassAction>> Car: read_all

<<Permission>>

MaintainingWork

<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute

<<Permission>>

Car

manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel ( int )::void
refill_oil( int ):::void
open( ):::void

DriveCommonCar

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute

<<Permission>>
MDS: access control decisions

- declarative access control (static)  
  ⇒ Permissions

- programmatic access control (dynamic)  
  ⇒ AuthorizationConstraints
programmatic access control

permission-name
permissions

authorization constraint
<OCL expression>
extend Car Entity

<table>
<thead>
<tr>
<th>&lt;&lt;Entity&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
</tr>
</tbody>
</table>

| manufacturer_name : String |
| model_name : String       |
| mpg : int                 |
| oil_level : int           |
| wheels : wheel[]          |
| class : [ common | special ] |

| change_wheel ( int )::void |
| refill_oil( int )::void   |
| open( )::void             |
| go_for_a_ride()::void     |
simpleAgent: may only drive common cars
mi6 - car access policy
specialAgent: “may” only drive super cars

```
self.Car_class = special
```

<<Role>>
specialAgent

<<Permission>>

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<ClassMethodAction>> Car_act_secret_gadgets: execute

<<Entity>>

Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]
class : [ common | special ]
change_wheel ( int ) : void
refill_oil( int ) : void
open( ) : void
go_for_a_ride() : void
act_secret_gadgets() : void
specialAgents: don’t do carsharing
change request

- I must reduce the CO2 emissions of our car fleet
- no car below 20 mpg may be used from now on
specialAgent

<<Role>>

simpleAgent

<<Role>>

specialAgent

<<Role>>

MaintainingAnalysis

<<ClassAction>> Car: read_all

<<Permission>>

MaintainingWork

<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute

<<Permission>>

self.Car_class = common

DriveCommonCar

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute

<<Permission>>

self.Car_class = special
self.Car_owner = caller.name

DriveSpecialCar

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<ClassMethodAction>> Car_act_secret_gadgets: execute

<<Permission>>

self.Car_mpg > 20

Car

manufacturer_name: String
model_name: String
mpg: int
oil_level: int
wheels: wheel[]
class: [ common | special ]
owner: String
change_wheel ( int ):void
refill_oil( int ):void
open( ):void
go_for_a_ride():void
act_secret_gadgets():void

<<Entity>>
specialAgents may drive any car
copy / paste simpleAgent permissions

self.Car_class = special
self.Car_owner = caller.name

DriveSuperCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<ClassMethodAction>> Car_act_secret_gadgets: execute

<<Entity>>
car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[]
class : [ common | special ]
owner : String
change_wheel ( int ):void
refill_oil( int ):void
open( ):void
go_for_a_ride():void
act_secret_gadget( int ):void

self.Car_class = common

DriveCommonCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute

<<Permission>>

self.Car_mpg > 20
use role hierarchy

<<Role>>
serviceAgent

<<Role>>
simpleAgent

<<Role>>
specialAgent

<<Entity>>
Car

- manufacturer_name : String
- model_name : String
- mpg : int
- oil_level : int
- wheels : wheel[]
- class : [ common | special ]
- owner : String
- change_wheel ( int )::void
- refill_oil( int )::void
- open( )::void
- go_for_a_ride()::void
- act_secret_gadget( int )::void

self.Car_owner = caller
self.Car_class = special
self.Car_class = common
self.Car_mpg > 20
MDS: Model Driven Security

- mi6 as model
- cars as protected resources
- RBAC based security policies
- empty EJB stubs + code implementing security mechanisms
summary:

• roles | permissions | entities
• composite actions
• action hierarchy
• authorization constraints
• role hierarchy
## conventional approach vs. MDS

<table>
<thead>
<tr>
<th>conventional approach</th>
<th>MDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>low level</td>
<td>arbitrary level of abstraction</td>
</tr>
<tr>
<td>policy format: XML</td>
<td>model elements (UML)</td>
</tr>
<tr>
<td>copy - paste / wildcards</td>
<td>hierarchy / composite container</td>
</tr>
<tr>
<td>running code from day 1</td>
<td>time intensive modeling, business logic comes later</td>
</tr>
</tbody>
</table>
remember:

<<Permission>>
permission_name

<<ClassMethodAction>>

... ...

system
model driven security offers:
- common representation for system and security
- general language composition schema
- arbitrary levels of abstraction
- unambiguous target code generation
- semantics as basis for model checking
bottom line:

- model driven security drawbacks:
  - modeling needs time and skills
    (reduce needed skills: tool development process, system development process)
  - new composite actions / action hierarchies
    ⇒ change the dialect
    ⇒ recomposition of language
    (can be solved with macros)
  - modifying the model ⇒ apply transformation functions again
    (can be solved with dedicated IDE or business logic stored outside of bean)
  - (“code generator” needed)
SecureUML

- modeling language

<table>
<thead>
<tr>
<th>abstract syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>concrete syntax</td>
</tr>
<tr>
<td><strong>semantics</strong></td>
</tr>
<tr>
<td>transformation functions</td>
</tr>
</tbody>
</table>
role based access control

users
  u
  ...
  user_n

roles
  role_1
  ...

permissions
  perm
  ...
  perm_n

actions
  a_1
  a_2
  ...
  a_n

UA
PA
AA

UA
PA
AA

Wednesday, December 17, 2008
role based access control

\[
RBAC_{\text{simple}} = \{ \\
(u, a_1) \in Users \times Actions | \\
\exists role_1 \in Roles, perm \in Permissions . \\
(u, role_1) \in UA \land \\
(role_1, perm) \in PA \land \\
(perm, a_1) \in AA
\}
\]
adding subjects

Subject

CompositeContainer

User

Group
adding subjects

\[ \text{RBAC}_{w, \text{subjects}} = \{ (u, a_1) \in \text{Users} \times \text{Actions} \mid \exists \text{sub} \in \text{Subjects}, \text{role}_1 \in \text{Roles}, \text{perm} \in \text{Permissions}, a_1 \in \text{Actions} . \]
\[ (\text{sub}, \text{role}_1) \in \text{UA} \land \]
\[ \text{sub} \geq \text{Subjects} u \land \]
\[ (\text{role}_1, \text{perm}) \in \text{PA} \land \]
\[ (\text{perm}, a_1) \in \text{AA} \} \]
adding role hierarchy

\[
\text{RBAC}_{w, \text{roleH.}} = \{ \\
(u, a_1) \in \text{Users} \times \text{Actions} | \exists \text{sub} \in \text{Subjects}, \text{role}_1, \text{role}_2 \in \text{Roles}, \text{perm} \in \text{Permissions}, a_1 \in \text{Actions} . \\
\text{sub} \geq_{\text{Subjects}} u \land \\
\text{role}_1 \geq_{\text{Roles}} \text{role}_2 \land \\
(\text{role}_2, \text{perm}) \in \text{PA} \land \\
(\text{perm}, a_1) \in \text{AA} \\
\}
\]
adding composite actions

\[ \text{RBAC}_{w, \text{comp}A.} = \{ \]
\[ (u, \ a_1) \in Users \times Actions | \]
\[ \exists sub \in Subjects, \ role_1, \ role_2 \in Roles, \ perm \in Permissions, \ a_2 \in Actions . \]
\[ \text{(sub, role}_1) \in UA \land \]
\[ \text{sub} \geq Subjects \ u \land \]
\[ \text{role}_1 \geq Roles \ role_2 \land \]
\[ a_2 \geq Actions \ a_1 \land \]
\[ (\text{role}_2, \ perm) \in PA \land \]
\[ (perm, \ a_2) \in AA \}
\]
SecureUML

- modeling language

<table>
<thead>
<tr>
<th>abstract syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>concrete syntax</td>
</tr>
<tr>
<td>semantics</td>
</tr>
<tr>
<td>transformation functions</td>
</tr>
</tbody>
</table>
abstract syntax
abstract syntax SecureUML

source: Security Engineering, Prof. D. Basin