A Secure Software Architecture Description Language

Jie Ren, Richard N. Taylor
Department of Informatics
University of California, Irvine

Software Security Assurance Tools, Techniques, and Metrics
November 7, 2005

Outline

- Background and insight
  - Architecture and Security
  - Software connectors
- Secure xADL
  - xADL
  - Access control models
  - XACML-based policy
- Case study: secure coalition
- Conclusion and future work

Main Goal

- Integrate security and software architecture
  - Integrate
  - Architecture level
  - Security: integrity through access control
  - Software engineering perspective: how to express, check, and enforce

Re-architecting boosts security!

Table 1. Secure by design.

POTENTIAL PROBLEM	PROTECTION MECHANISM	DESIGN PRINCIPLES
The underlying `dll (ntdll.dll)` was not vulnerable because…	Code was made more conservative during the Security Push.	Check precondition
Even if it were vulnerable…	Internet Information Services (IIS) 6.0 is not running by default on Windows Server 2003.	Secure by default
Even if it were running…	IIS 6.0 does not have WebDAV enabled by default.	Secure by default
Even if Web-based Distributed Authoring and Versioning (WebDAV) had been enabled…	The maximum URL length in IIS 6.0 is 16 Kbytes by default (> 64 Kbytes needed for the exploit).	Tighten precondition, secure by default
Even if the buffer were large enough…	The process halts rather than executes malicious code due to buffer-overrun detection code inserted by the compiler.	Tighten postcondition, check precondition
Even if there were an exploitable buffer overrun…	It would have occurred in w3wp.exe, which is running as a network service (rather than as admininstrator).	Least privilege

(Data courtesy of David Aucsmith.)

Wing, IEEE Security & Privacy, 2003

Traditional SA

- Component-based Software Engineering
- Software Architecture
  - Structure
  - Behavior
    - Process algebra (Wright), labeled transition system (Darwin)

Connectors

- Should they be first class citizens?
  - Capture and reuse
- Existing work
  - Taxonomy: Mehta 2000
  - Assembly Language: Mehta 2004
  - Constructions: Lopes 2003
  - Transformation: Spitznagel 2001
- No rich security
  - Dependability: Spitznagel 2004

Our Approach

- Describe and enforce Architectural Access Control
  - Combine software architecture and security research
  - Based on the extensible xADL language
  - Adopt an integrated access control model: classic, role-based, trust management
  - Utilize XACML

Overview of xADL

- XML-based extensible architecture description language
- Component and connector
- Types
- Signatures and interfaces
- Sub-architecture
- Design-time and run-time
- Tool support: ArchStudio
- Extensible: configuration, execution

Unified Access Control

- Classic Access Control
  - Subject, object, operation
- Role-based Access Control
  - Use role as an indirection
- Role-based Trust Management
  - Trust management: attributes
  - Inspired by Professor Ninghui Li’s work
  - Trust relationship between roles of different domains

Secure xADL

- Concepts for Architectural Access Control
  - subject, principal, resource, privilege, safeguard, and policy
- Integrate with xADL
- The first effort to model these security concepts directly in an architectural description language

Subject

- A ***subject*** is the user on whose behalf software executes.
- Missing from traditional software architecture:
  - All of its components and connectors execute under the same subject,
  - The subject can be determined at design time,
  - It will not change during runtime, either advertently or intentionally
  - Even if there is a change, it has no impact on the software architecture.

Principal

- A subject can take multiple ***principals***, which are possessed credentials.
- Classic access control: subjects
- RBAC: roles
- Trust management: keys, certificates,

Resource

- A ***resource*** is an entity whose access should be protected.
- Passive: files, sockets, etc.
- Active: components, connectors

Privilege

- *Permissions* describes a possible operation on an object.
- ***Privilege*** describes what permissions a component possess depending on the executing subjects.
- Privilege escalation vulnerabilities
- Two types of privileges:
  - Traditional: read file, open sockets, etc.
  - Architectural: instantiation, connection, message routing, introspection

Safeguard

- ***Safeguards*** are permissions that are required to access the interfaces of the protected components and connectors.
- Architectural access control check

Policy

- A ***policy*** specifies what privileges a subject should have to access resources protected by safeguards.
- Numerous existing studies in the security community.
- We focus on software engineering applicability for architectural modeling.
- XACML
  - XML-based
  - Extensible: RBAC profile
  - Tool support

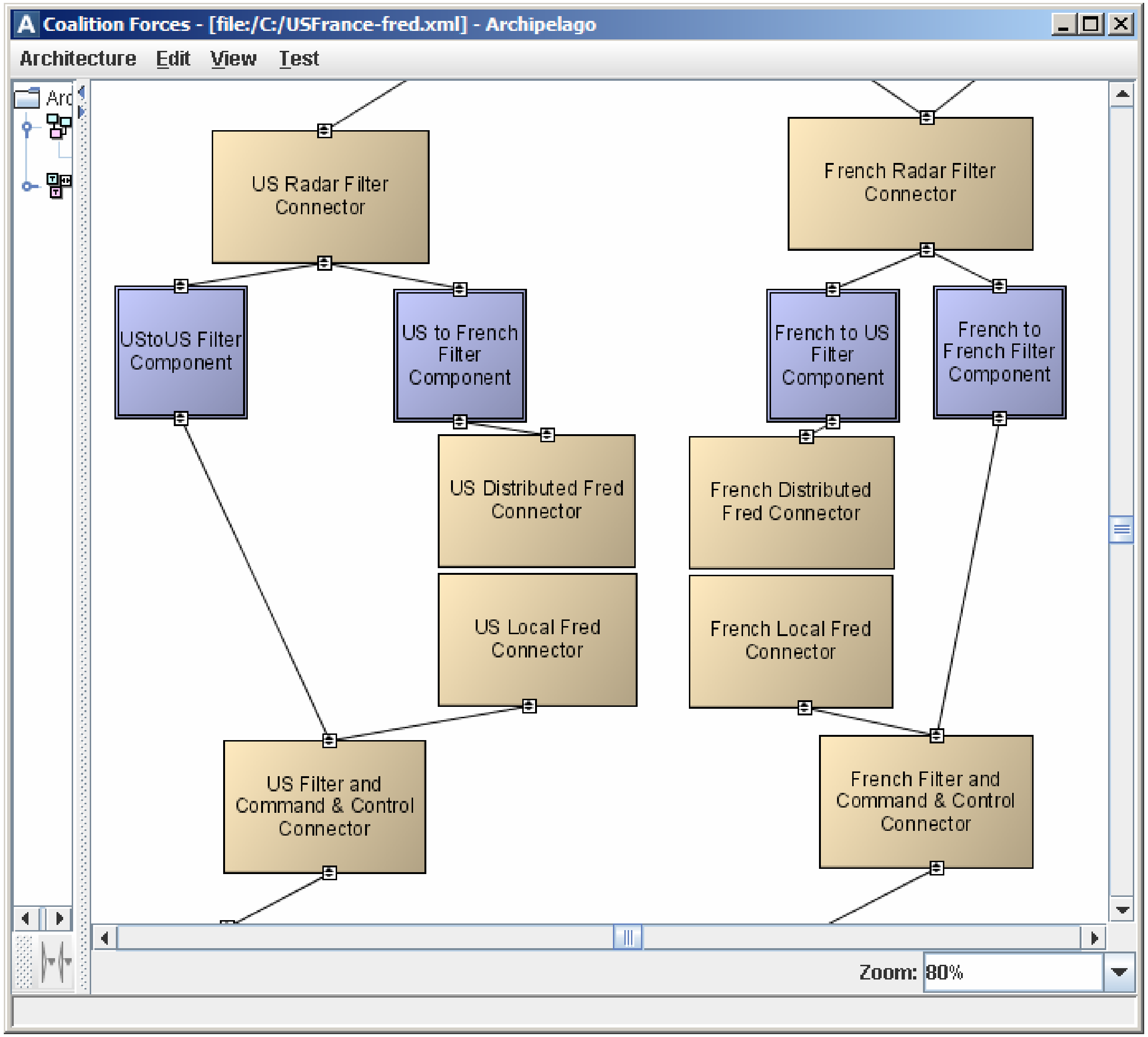
Contexts for Architectural Access Control

- Access control decisions might be based on entities other than the decision maker and the protected resource. These relationships are the *context.*
- Four types of context
  - The nearby components and connectors of the component and the connector
  - The explicitly modeled sub-architecture that contains the component and the connector
  - The type of the component and the connector,
  - The global architecture.
- XACML's combining algorithms supply a framework to combine these contexts

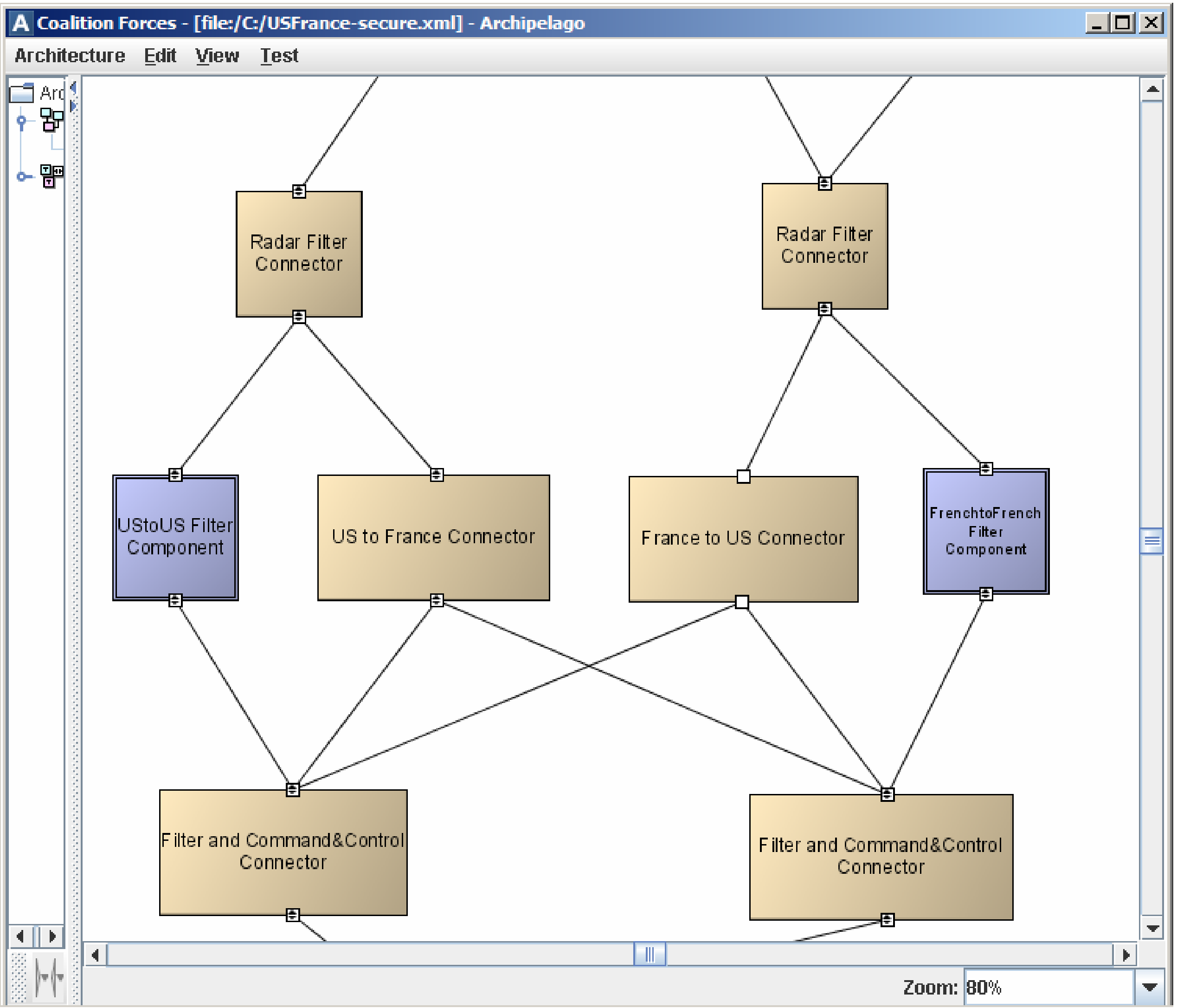
Syntax of Secure xADL

```
<complexType name=”SecurityPropertyType">
  <sequence>
    <element name="subject“ type="Subject"/>
    <element name="principal“ type="Principals"/>
    <element name="privilege“type="Privileges"/>
    <element ref="xacml:PolicySet"/>
  </sequence>
<complexType>
<complexType name="SecureConnectorType">
  <complexContent>
    <extension base="ConnectorType">
      <sequence>
        <element mame="security“
           type="SecurityPropertyType"/>
      <sequence>
    <extension>
  <complexContent>
</complexType>
<!-- similar constructs for component, structure, and
   instance -->
```

Case Study: Secure Coalition



Secure Connector



Architectural Policy: Routing

```
<connector id="UStoFranceConnector">
 <security type="SecurityPropertyType">
  <subject>US</subject>
   <Policy RuleCombiningAlgId="permit-overrides">
    <Rule Effect="Permit">
     <Target>
      <Subject><AttributeValue>UStoFranceConnector
        <AttributeDesignator AttributeId="subject-id"/>
      <Resource><AttributeValue>RouteMessage
        <AttributeDesignator AttributeId="resource-id"/>
      <Action><AttributeValue>RouteMessage
        <AttributeDesignator AttributeId="action-id"/>
     <Condition FunctionId="string-equal">
      <AttributeValue>Aircraft Carrier
      <Apply>
       <AttributeSelector RequestContextPath =
         "//context:ResourceContent/security:routeMessage/
         messages:namedProperty[messages:name='type']/
         messages:value/text()"/>
<Rule RuleId="DenyEverythingElse" Effect="Deny"/>
```

Conclusion

- Background and insight
  - Combine security and software architecture
  - Architectural Access Control
- Approach
  - Extend xADL
  - A unified access control model
  - Subject, principal, resource, privilege, safeguard, and policy
  - XACML as the base policy syntax
- Case study: secure coalition
- Future work
  - Algorithm for architectural access control algorithm
  - Tool support