Evidence-based Audit

Jeff Vaughan
Limin Jia, Karl Mazurak, and Steve Zdancewic

Department of Computer and Information Science
University of Pennsylvania

CSF/LICS Joint Session
June 24, 2008
Our Setting: Distributed Access Control

Diagram showing the relationship between applications, resources, principals, and data in a distributed access control system.
Our Setting: Distributed Access Control

Diagram showing the relationships between applications, resources, and principals in a distributed access control system.
Our Setting: Distributed Access Control

Diagram showing relationships between applications, resources, principals, and data.
Our Setting: Distributed Access Control

- Application
- Resource
- Principal
- Data

Diagram showing nodes connected with lines, indicating access control relationships in a distributed system.
Key Idea: Proofs attest to data integrity.
The Aura Project

- A programming language called Aura
  - A propositional fragment, modeled here by Aura$_0$
  - An ML-like computation language [Jia+ 08]
- A security aware programming model
  - active, potentially malicious principals
  - mutual distrust between applications and principals
  - emphasis on access control and audit
- An implementation including compiler and .Net-based runtime
The Aura Project

- A programming language called Aura
  - A propositional fragment, modeled here by Aura\_0
  - An ML-like computation language [Jia+ 08]
- A security aware programming model
  - active, potentially malicious principals
  - mutual distrust between applications and principals
  - emphasis on access control and audit
- An implementation including complier and .Net-based runtime

Today’s Talk
Analyzing the local security of Aura applications.
Access control, locally.

Application
(Untrusted)

raw-op1
raw-op2

Resource
Access control, locally.
Access control, locally.
Access control, locally.

Application (Untrusted)

raw-op1 raw-op2

Resource
Access control, locally.

Application (Untrusted)

raw-op1  raw-op2

Resource
Access control, locally.
Access control, locally.

Application (Untrusted)

raw-op1

raw-op2

Resource
Access control, locally.
Access control, locally.

Application (Untrusted)

Resource

raw-op1

raw-op2
Access control, locally.

Application (Untrusted)

raw-op1 raw-op2

Resource
Access control, locally.

Application (Untrusted)

raw-op1

raw-op2

Resource
Access control, locally.

Diagram showing an application (untrusted) accessing a resource with raw-ops 1 and 2.
Access control, locally.

Application (Untrusted)

raw-op1
raw-op2

Resource
Access control, locally.
Jargon needed for this talk

**Institutional Policy** Human-level rules about principals, data values, and resources.

**Bad Access** A system state change violating institutional policy.

**Formal Rules** Machine-level encoding of policy.
Jargon needed for this talk

**Institutional Policy**  Human-level rules about principals, data values, and resources.

**Bad Access**  A system state change violating institutional policy.

**Formal Rules**  Machine-level encoding of policy.

**Foreshadowing**  Users care about *institutional policy*, but technology tries to enforces *formal rules*. 
## Why do reference monitors allow bad accesses to occur?

<table>
<thead>
<tr>
<th>Problem 1</th>
<th>The trusted computing base’s implementation may be buggy.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 2</td>
<td>The formal rules language may be too impoverished to express institutional policy.</td>
</tr>
<tr>
<td>Problem 3</td>
<td>The system may be configured with incorrect formal rules.</td>
</tr>
</tbody>
</table>

And many other reasons not addressed here...
Common application design exacerbates these problems.
Common application design exacerbates these problems.

- 
  -C <directive> (initialization time)
  (Untrusted)

- --disable-cgi (compile time)

- .htaccess files
- /etc/httpd.conf
- External Rules
- Log

-raw-op1
- raw-op2

Resource
Common application design exacerbates these problems.
Problem 1

The trusted computing base’s implementation may be buggy.

Aura Solution

Trust only a small, generic *kernel* that has no application-specific functionality.

[Saltzer+ 75], [Bauer+ 99], [Jia+ 08]
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.

if (op2 △ ▪) is well-typed then (forward △ to resource; log {△, ▪}; ... ) else skip
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
In Aura, a lightweight kernel protects resources.
Example: Types for a remote procedure call server.

The RPC server resource has a single raw operation

\[ \text{raw-rpc} : \text{string} \Rightarrow \text{string} \]

The kernel exposes an extended signature of propositions.

\[ \text{OkToRPC} : \text{string} \rightarrow \text{Prop} \]
\[ \text{DidRPC} : \text{string} \rightarrow \text{string} \rightarrow \text{Prop} \]

Applications are written against the kernel interface.

\[ \text{rpc} : (x : \text{string}) \Rightarrow (\text{Kernel says} \ \text{OkToRPC} \ x) \Rightarrow \{y : \text{string}; \text{Kernel says} \ \text{DidRPC} \ x \ y\} \]
The RPC server resource has a single raw operation

$$\text{raw-rpc} : \text{string} \Rightarrow \text{string}$$

The kernel exposes an extended signature of propositions.

$$\text{OkToRPC} : \text{string} \to \text{Prop}$$

$$\text{DidRPC} : \text{string} \to \text{string} \to \text{Prop}$$

Applications are written against the kernel interface.

$$\text{rpc} : (x : \text{string}) \Rightarrow (\text{Kernel says OkToRPC } x) \Rightarrow \{y : \text{string}; \text{Kernel says DidRPC } x \ y\}$$

The kernel interface and extended signature may be mechanically generated.
Problem 2

The formal rules language may be too impoverished to express institutional policy.

Aura Solution

Use dependent DCC with signature objects to specify rules.

[Abadi+ 06], [Fournet+ 07], [Bengtson+ 08] . . .
Aura’s **says** modality represents affirmation. (1/2)

\[
\Gamma \vdash P : \text{Prop} \\
\Gamma \vdash A \text{ says } P : \text{Prop}
\]

The proposition “Principal A affirms proposition \( P \).”

\[
\Gamma \vdash P : \text{Prop} \\
\Gamma \vdash \text{sign}(A, P) : A \text{ says } P
\]

A’s signature on \( P \). \( P \) might be unprovable.
Aura’s \textit{says} modality represents affirmation. (1/2)

\[ \Gamma \vdash P : \text{Prop} \]
\[ \Gamma \vdash A \text{ says } P : \text{Prop} \]

\[ \Gamma \vdash P : \text{Prop} \]
\[ \Gamma \vdash \text{sign}(A, P) : A \text{ says } P \]

The proposition “Principal $A$ affirms proposition $P$.”

A’s signature on $P$. $P$ might be unprovable.
Aura’s **says** modality represents affirmation. (2/2)

\[
\frac{\Gamma \vdash p : P}{\Gamma \vdash \text{return}[A]p : A \text{ says } P}
\]

\[
\frac{\Gamma \vdash p : A \text{ says } P}{\frac{\Gamma, x : P \vdash q : A \text{ says } Q}{\frac{\Gamma \vdash \text{bind } x = p \text{ in } q : A \text{ says } Q}}}
\]

A affirms proven propositions.

A affirms the result of hypothetical reasoning. We can “reason from A’s point of view.”

(These are standard monad rules.)
### Partial syntax

```plaintext
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$ ::=</td>
<td><strong>string</strong></td>
<td><strong>prin</strong></td>
</tr>
<tr>
<td></td>
<td>$x$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$t$ <strong>says</strong> $t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(x : t) \rightarrow t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(x : t) \Rightarrow t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>${x : t ; t}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t \ t$</td>
<td></td>
</tr>
</tbody>
</table>
```

Syntactic separation of computation and logical arrows stop arbitrary resource-effects from polluting the logic.
Dependent types allow for expressive rules.

Example (Bob acts for Alice)

| Alice says | ((P: Prop) → Bob says P → P) |
Dependent types allow for expressive rules.

Example (Bob acts for Alice)

Alice says \((P: \text{Prop}) \rightarrow \text{Bob says } P \rightarrow P\)

Example (Bob acts for Alice only regarding validity)

Alice says \((x: \text{string}) \rightarrow \text{Bob says } \text{valid } x \rightarrow \text{valid } x\)
Dependent types allow for expressive rules.

**Example (Bob acts for Alice)**

Alice says \(((P : \text{Prop}) \rightarrow \text{Bob says } P \rightarrow P)\)

**Example (Bob acts for Alice only regarding validity)**

Alice says \(((x : \text{string}) \rightarrow \text{Bob says } \text{valid } x \rightarrow \text{valid } x)\)

**Example (Kernel allows RPC calls on strings endorsed by Alice)**

Kernel says \(((x : \text{string}) \rightarrow \text{Alice says } \text{valid } x \rightarrow \text{OkToRPC } x)\)
Problem 3

The system may be configured with incorrect formal rules.

Aura Solution

Consistently log runtime proof objects for later analysis.

[Wee 95], [Cederquist+ 05]
When something unexpected happens: look at the log.

- System design guarantees a one-to-one correspondence between log entries and resource state changes.
- If a Alice’s signature does not appear in a log entry, she could not have caused the associated action.
- Proofs get convoluted, but proof reduction can restore clarity.

Example (A convoluted proof)

$$\lambda x. \lambda y. y \ (\text{sign}(\text{Alice, } P)) \ (\text{sign}(\text{Bob, } Q))$$

Here Alice’s signature is “irrelevant.”
Reduction relation includes special cases for `bind`.

### R-Bind-Specious

\[
\frac{x \notin \text{fv}(t_2)}{\text{bind } x = t_1 \text{ in } t_2 \rightarrow t_2}
\]

Drops unused hypotheses. Most interesting when `t_1` is a signature.

### R-Bind-Commute

\[
\frac{y \notin \text{fv}(t_3)}{\text{bind } x = (\text{bind } y = t_1 \text{ in } t_2) \text{ in } t_3 \rightarrow \text{bind } y = t_1 \text{ in } \text{bind } x = t_2 \text{ in } t_3}
\]

Commutation rule that can enable further reductions.

...plus standard \(\beta\) and structural rules.
Example (The same convoluted proof)

\[(\lambda x. \lambda y. y) \langle \text{sign}(\text{Alice}, P) \rangle \langle \text{sign}(\text{Bob}, Q) \rangle \rightarrow^* \text{sign}(\text{Bob}, Q)\]

Example (Reducing by special bind rules)

\[
\text{bind } x = (\text{bind } y = (f \text{ sign}(\text{Bob}, Q))) \\
\text{in } \text{sign}(\text{Alice}, P) \\
\rightarrow^* \text{sign}(\text{Alice}, P)
\]
Theorem (Subject Reduction)

If $p \rightarrow p'$ and $\Gamma \vdash p : s$ then $\Gamma \vdash p' : s$.

Theorem (Confluence)

If $p \rightarrow^* p_1$, and $p \rightarrow^* p_2$, then there exists $p_3$ such that $p_1 \rightarrow^* p_3$ and $p_2 \rightarrow^* p_3$.

Theorem (Strong Normalization)

If $\Gamma \vdash p : s$, then $p$ is strongly normalizing (SN). That is, all reduction sequences starting with $p$ halt.
$\text{Fact}$

The Calculus of Constructions with dependent pairs (CoC) is SN.

$\text{Proof Idea}$

Show Aura reductions can be simulated in terminating system based on CoC.

$$
\begin{align*}
& \text{Aura} \quad \text{reduction} \quad \text{CoC} \\
& p_0 \to \quad [\cdot] \quad p_1 \to \quad [\cdot] \quad p_2 \to \quad \ldots \\
& t_0 \quad \to \quad t_1 \quad \to \quad t'_1 \quad \to \quad t_2 \quad \to \quad \ldots
\end{align*}
$$
Aura is a framework for proof carrying authorization and audit.

Aura includes
- a small and generic trusted computing base,
- an expressive authorization logic, and
- a principled audit methodology.
Take home message

Aura is a framework for proof carrying authorization and audit.

Aura includes
- a small and generic trusted computing base,
- an expressive authorization logic, and
- a principled audit methodology.

Code and technical reports available from
http://www.cis.upenn.edu/~stevez/sol/aura.html

Or ask us for a demo!
Bonus Slides

- Typing Rules for Pair and Arrow
- Strong Normalization Proof
Typing rules for arrow and pair.

\[
\frac{\Sigma; \Gamma \vdash t_1 : k_1 \quad \Sigma; \Gamma, x : t_1 \vdash t_2 : k_2}{\Sigma; \Gamma \vdash (x : t_1) \rightarrow t_2 : k_2}
\]

\[
k_1 \in \{\text{Kind}^P, \text{Prop}, \text{Type}\} \quad k_2 \in \{\text{Prop, Type}\}
\]

\[
\frac{\Sigma; \Gamma \vdash t_1 : k_1 \quad \Sigma; \Gamma, x : t_1 \vdash t_2 : k_2}{\Sigma; \Gamma \vdash \{x : t_1; t_2\} : k_1}
\]
Translation from Aura to CoC erases says

<table>
<thead>
<tr>
<th>Definition</th>
<th>≈</th>
</tr>
</thead>
<tbody>
<tr>
<td>([A \text{ says } P])</td>
<td>([P])</td>
</tr>
<tr>
<td>([\text{return@}[A]p])</td>
<td>([p])</td>
</tr>
<tr>
<td>([\text{bind } x = p: P \text{ in } q])</td>
<td>((\lambda x: P . [q]) [p])</td>
</tr>
<tr>
<td>([\text{sign}(A, P)])</td>
<td>(x \text{ fresh})</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>$[A \text{ says } P]<em>\Delta \approx [P]</em>\Delta$</td>
<td></td>
</tr>
<tr>
<td>$[\text{return}@[A]p]<em>\Delta \approx [p]</em>\Delta$</td>
<td></td>
</tr>
<tr>
<td>$[\text{bind } x = p: P \text{ in } q]<em>\Delta \approx (\lambda x:[P]</em>\Delta. [q]<em>\Delta) [p]</em>\Delta$</td>
<td></td>
</tr>
<tr>
<td>$[\text{sign}(A, P)]_\Delta \approx \Delta(\text{sign}(A, P))$</td>
<td></td>
</tr>
</tbody>
</table>

Where $\Delta$ maps signatures to unique fresh variables. (We’ll treat the $\Delta$’s implicitly from now on.)
Translation from Aura to CoC erases \textit{says}

\textbf{Definition}

\[
\begin{align*}
[A \; \text{says} \; P]_\Delta & \approx [P]_\Delta \\
[\text{return}@[A]p]_\Delta & \approx [p]_\Delta \\
[\text{bind} \; x = p: P \; \text{in} \; q]_\Delta & \approx (\lambda x : [P]_\Delta . [q]_\Delta) \; [p]_\Delta \\
[\text{sign}(A, P)]_\Delta & \approx \Delta(\text{sign}(A, P))
\end{align*}
\]

Where $\Delta$ maps signatures to unique fresh variables. (We’ll treat the $\Delta$’s implicitly from now on.)

\textbf{Lemma}

\textit{If $p$ is a well typed term in Aura}_0, \textit{then—for an appropriate context—}[p] \textit{is well typed in CoC.}
Constructions needs a new reduction to simulate bind.

**Definition (CC’ reduction)**

The CC’ reduction relation augments the standard Calculus of Construction reduction relation with

\[
(\lambda x: t. t_1)((\lambda y: s. t_2)u) \rightarrow (\lambda y: s. ((\lambda x: t. t_1)t_2))u
\]

\(\beta'\) reductions simulate R-Bind-Commutte reductions:

\[
\begin{align*}
\text{bind } x &= (\text{bind } y = t_1 \text{ in } t_2) \text{ in } t_3 \\
\text{bind } y &= t_1 \text{ in bind } x = t_2 \text{ in } t_3
\end{align*}
\]
Aura_0 is strongly normalizing.

Lemma

Well typed CoC terms are SN under CC' reduction.

[Lemma 05]

Lemma

If p is a well typed Aura_0 term and p \rightarrow p'. Then 
\langle p \rangle \rightarrow^{+}_{CC'} \langle p' \rangle using one or more steps.

Proof of strong normalization.

Imagine p is an looping Aura_0 term. Then by the second lemma we can build a term which loops according to CC'. This contradicts the first lemma.