A Reusable, Secure Reference Monitor based on the Aura Programming Language

Luke Zarko
A MOTIVATING EXAMPLE
Knitter is a Web service that allows users to post short messages to one another.

Knitter will release a public API; clients who wish to perform privileged actions must provide evidence.

This evidence can be logged for later auditing.
Basic Login and Making Posts

- Knitter provides an insecure RPC to get the user ID for a particular name.

Client \(\xrightarrow{\text{getUserID "Alice"}}\) Server

\(1234\)
Users need to get evidence that they know the password to their account

Client

matchPassword
user:1234
hash:md5("secret")

Server

md5("secret") matches user 1234
Evidence

- Evidence may be built from assertions
  - Digital signatures prevent forging

Signed assertion

\[ \text{md5(“secret”) matches user 1234} \]

Signed policy

\[ \text{if md5 matches, then login OK} \]
Basic Login and Making Posts

Knitter accepts evidence that the password matches in its database as evidence that a principal may log in as a user.

This promotion is ideally done with no additional round trips to the service backend.

Client

\[
\text{if md5 matches, then login OK} + \text{md5(“secret”) matches user 1234}
\]
Knitter accepts evidence that the password matches in its database as evidence that a principal may log in as a user.

This promotion is ideally done with no additional round trips to the service backend.

Client

login(1234) OK
The user can make posts as a particular Knitter user with evidence that he is logged in.

- If login OK, then can post

+ login(1234) OK

- login OK

  can post (1234)
The OpenLogin system allows single sign-on for many Web services.

Knitter allows evidence from OpenLogin to be used as evidence for Knitter login (provided there is a map between an OLID and a Knitter UID).

- **OpenLogin for User OLID(x) OK**
- **OLID(x) maps to UID(x)**
- **If OpenLogin OK and OLID maps to UID, then login OK**
Separating Policy from Mechanism

- Normally, Knitter would implement these security features at the same level as their database.
- This project advocates a different method: a separate piece of software called a reference monitor that can manage security.
  - ...and will allow us to separate the policy of Knitter access from its mechanism.
- This software can be used with different policies.
  - ...so the size of the trusted computing base shrinks.
The AURA policy language is due to Jia, Vaughan, Mazurak, Zhao, Zarko, Schorr and Zdancewic.
The Aura Log

- All actions on the reference monitor are automatically logged for later auditing.
- Rather than reinvent the wheel, I use XML and XQuery to represent and query the tree-structured evidence.

\[ \text{md5(“secret”) matches user 4} \]
Related Work

► KeyNote
  ▶ Many of the same goals (small, reusable)
  ▶ Based on execution of programs on strings (that may be signed to become credentials)
  ▶ Acts as an ‘advisor’ to hosting applications

► PolicyMaker
  ▶ Superset of KeyNote
  ▶ Signature verification is not built in
  ▶ Runs programs in a shared tuple space until an accept message is emitted
**Related Work**

- **REFEREE**
  - Can fetch credentials at runtime
  - Verification of signatures is left to policy

- **Logic Programming (Fournet, Gordon and Maffeiis)**
  - Specifies authorization policy in a high-level logic language
  - Static reasoning possible
  - Exposes crypto primitives
  - Erases proofs
Weaknesses

- The Aura runtime and language do not support revocation
  - Active area of research; in the short term, timestamps
- Some policies may be sensitive to replay attacks
  - Use encrypted RPC links
The AURA reference monitor provides

- An expressive language for policy
- A reusable security system
- A method for audit with powerful query tools

In contrast to other approaches

- AURA requires strict mediation
- AURA benefits from a strong static typing model
- Signatures in AURA are handled transparently
- Evidence in AURA may be logged for later audit
Questions?
The user can make posts as a particular Knitter user with evidence that he is logged in.

Client

Server

posted

post user:1234
text: “hello”
Access Control Policy

- A system’s *access control policy* determines who (which *principals*) may perform actions on protected resources.
- Various mechanisms are used today to implement access control policy
  - UNIX permission bits
  - Windows NT ACLs
  - Pharos printer kiosks
  - Bespoke security features on Web backends
Access Control Policy

- Each of these mechanisms ask you to trust a different system
  - We believe CETs is doing their job, so we believe transitively that UNIX, NT and Pharos are correct

- Ideally, we could shrink the size of the trusted computing base—the components that must be trusted to enforce a given security policy
  - You wouldn’t write your own crypto, so why write your security mechanism?

EVIDENCE

- Evidence may be combined without communicating with signing parties

![Login (1234) OK]

Not signed, but *well-typed* and constructed from signed evidence
Distributing Aura Services

- OpenLogin distributes two main files to provide access to its service:
  - A *public key*, used to check signatures on its evidence
  - A *policy description file*, which contains names and descriptions of the types of evidence and RPCs
Knitter wants to allow advertisers to befriend particular Knitter users.

So they simply add a new way to build evidence that a principal can add a friend:

- if an affiliate, then for all UIDs, can add a friend.