Software Foundations of Security & Privacy 15316 Spring 2017 Lecture 9: Authentication and Authorization

Matt Fredrikson, Jean Yang
mfredrik@cs.cmu.edu

February 14, 2017

Access control in a nutshell

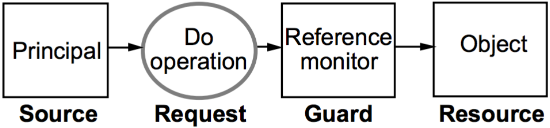

Image Credit: Butler Lampson, Martín Abadi, Michael Burrows

The reference monitor needs to answer two questions:

- *"Who said this?"* and *"Who is trusted to access this?"*
- **Authentication**: obtaining the source of the request
- **Authorization**: interpreting and deciding the access rule

Authentication

Requests arrive over a **channel**

- Network connection
- Pipe/IPC
- System call
- User input

Monitor must associate a **principal** to the channel

- **Centralized**: Requester, monitor, and policy reside on the same system under control of a single authority
- **Distributed**: Requester, monitor, and policy reside on multiple systems, with multiple authorities at differing levels of trust

Centralized model

Authentication: how do we name and identify principals?

- Login names, user IDs, process IDs, ...
- System maps process and login contexts to principals

Authorization: how do we decide who gets access?

- Access control list
- Stored in reference monitor, or with the object
- Maps principals to access rights

For the most part, this is a solved problem

- Just need to trust the operating system
- System implements all channels, maintains the policy
- Knows who is responsible for each request

Distributed model

Things are harder in a distributed system

- **Autonomy**: Path between the object and requesting principal may involve machines with differing levels of trust. We might want policies that take this into account.
- **Size & heterogeneity**: Number of users, objects, and authorities may be large, and have inconsistent interfaces and security models. Must scale to meet the demand.
- **Fault tolerance**: Remote parts of the system may become inaccessible, but we'd like to maintain as much functionality as possible.

Logic to the rescue

Mathematical formalism for authentication and authorization

All systems make assumptions about authority and trust

- Logic forces us to state these assumptions precisely
- Provides sound rules for working out the consequences

To do this, we'll make some assumptions:

- Correctness of underlying operating system
- Hardware implementation is correct and secure
- Secure encryption primitives

Logic due to Reiter, influenced by Lamport, Abadi, Burrows

Fundamentals: principals, statements, and objects

Principals exist to make **statements**

`mfredrik` *says* “`syeom` *speaks for* `15316-spring17-staff`”

- “`syeom` *speaks for* `15316-spring17-staff`” is a statement
- `mfredrik`, `syeom`, `15316-spring17-staff` are principals

Anything that a principal says is a statement

- requests, delegations, trust relationships, ...

Objects are the entities protected by authorization requests

Principals

Principals are named entities, groups, and channels

- People (`mfredrik`, `jyang2`, `syeom`)
- Machines (`linux2.cs.cmu.edu`, ...)
- Groups (`coursestaff`, `students`, `administrators`)
- Channels (`128.2.220.63`, AES key `#574897`)

Formally, a principal is either an identifier or a key:

$$p \quad ::= \quad \textbf{key}(s) \mid \textit{identifier} \mid p.s$$

where s is a string

Statements

Statements, where s is a string and p is a principal

$$
\begin{array}{rcl}
\phi & ::= & \textbf{action}(s) \\
 & | & p \textbf{ says } \phi \\
 & | & p \textbf{ speaksfor } p \\
 & | & s \textbf{ signed } \phi \\
 & | & \textbf{delegates}(p, p, s) \\
 & | & \phi \rightarrow \phi \\
 & | & \phi \wedge \phi
\end{array}
$$

s **signed** ϕ

What does this correspond to in reality?

A **digital signature scheme** is a triple $\langle G, S, V \rangle$

- The **key generator** G takes a key length n and outputs a public/private key pair (pk, sk)
- The **signing algorithm** S takes a private key sk and message m, and outputs a signature σ:

$$\sigma \leftarrow S(sk, m)$$

- The **verifier** V takes a public key pk, message m, and signature σ, and outputs either 0 or 1:

$$V(pk, m, \sigma) \mapsto \{0, 1\}$$

Signatures: correctness

Correctness of a signature scheme

$\langle G, S, V \rangle$ is a correct digital signature scheme if with all but negligible probability over the key pairs (ps, sk) output by G, it holds that:

$$V(pk, m, S(sk, m)) = 1$$

In other words, the verifier accepts valid signatures.

Is this enough?

Signatures: security

Let $\Pi = \langle G, S, V \rangle$ be a signature scheme

Define the following experiment $\mathsf{Forge}(\Pi, \mathcal{A}, n)$:

1. Run $(pk, sk) \leftarrow G(n)$.
2. Give $\mathcal{A}$ pk, and let it run $S(sk, \cdot)$. Let Q be the set of queries $\mathcal{A}$ gives to $S(sk, \cdot)$.
3. $\mathcal{A}$ then outputs (m, σ).
4. $\mathcal{A}$ wins when both $V(pk, m, \sigma) = 1$ and $m \notin Q$.

Security of a signature scheme

Π is secure if for all polynomial-time adversaries $\mathcal{A}$ and key lengths n,

$$\Pr[\mathsf{Forge}(\Pi, \mathcal{A}, n) \text{ wins}] \leq \mathsf{negl}(n)$$

for some negligible function negl.

Signatures: summary

Signature schemes are widely used in authorization

- Pay attention to the way that security is carefully defined
- Gives a rigorous justification for s **signed** ϕ

This isn't a class about cryptography

- We won't cover signature schemes in more detail
- You aren't expected to memorize this definition
- The details of the scheme aren't important to us
- As long as we have one that satisfies security

Associating trust: **speaksfor**

A **speaksfor** B

When A says something, we believe that B says it too

- `128.2.220.63` **speaksfor** `mfredrik`
- AES key `#574897` **speaksfor** `jyang2`
- `syeom` **speaksfor** `coursestaff`

speaksfor formalizes indirection for statements

- Some principals can't communicate directly with others
- Principals often have several others speaking for them
- Roles may have a rotating set of principals speak for them

Provable statements

Write **judgements** to denote statements that are provably true

$$\vdash s$$

This means: **statement s is provable without assumptions**

$$P \vdash s$$

This means: s **is provable using assumptions given in** P

We use **inference rules** to prove things about statements

Inference rules

says Introduction 1 (Says-I1)

$$\frac{s \textbf{ signed } \phi}{\textbf{key}(s) \textbf{ says } \phi}$$

Intuitively, this rule:

- Creates a principal **key**(s) from a *key string* s
- Establishes **key**(s) said ϕ given the appropriate signature

Inference rules

says Introduction 2 (Says-I2)

$$\frac{\phi}{p \textbf{ says } \phi}$$

What does this rule say?

- If ϕ is established to be true, then p says it
- Maybe p didn't say it, but we can proceed as though it did
- Basically, principals will say true things

Inference rules

says Introduction 3 (Says-I3)

$$\frac{p \textbf{ says } (p.s \textbf{ says } \phi)}{p.s \textbf{ says } \phi}$$

This rule might seem counterintuitive

- $p.s$ is the “principal that p calls s”.
- p can name s to be whomever it wants
- p can just find someone who says ϕ, and call that person s
- We must accept what p says about the person it calls s

Inference rules

says Implication (Says-Impl)

$$\frac{p \textbf{ says } (\phi_1 \rightarrow \phi_2) \qquad p \textbf{ says } \phi_1}{p \textbf{ says } \phi_2}$$

Recall modus ponens from propositional calculus

- Says-Impl is modus ponens over **says**
- We take principals at their word
- To the logical conclusion

Inference rules

speaksfor Elimination 1 (Speaksfor-E1)

$$\frac{p_1 \textbf{ says } (p_2 \textbf{ speaksfor } p_1) \qquad p_2 \textbf{ says } \phi}{p_1 \textbf{ says } \phi}$$

speaksfor Elimination 2 (Speaksfor-E2)

$$\frac{p_1 \textbf{ says } (p_2 \textbf{ speaksfor } p_1.s) \qquad p_2 \textbf{ says } \phi}{p_1.s \textbf{ says } \phi}$$

These rules deal with broad delegations of authority

- When p_1 says p_2 speaks for her...
- Then anything p_2 says is attributable to p_1

Inference rules

delegates Elimination (Delegate-E)

$$\frac{p_1 \textbf{ says delegates}(p_1, p_2, s) \qquad p_2 \textbf{ says action}(s)}{p_1 \textbf{ says action}(s)}$$

This rule allows more fine-grained delegation of authority

- **delegates** allows p_1 to let p_2 speak-for her
- But only with regard to selected actions

Example: Certification authorities

Let's use the rules to reason about a certification authority

A certification authority is a named principal *CA*

For our purposes, it issues statements of the form:

$$K_{\textsf{\textit{CA}}}\ \textbf{signed}\ (\textbf{key}(K_A)\ \textbf{speaksfor}\ \textbf{key}(K_{\textsf{\textit{CA}}}).A)$$

This statement is called a **certificate**

- Usually, K_CA is a public key known to everyone
- It could also be a symmetric key
- If so, need to ensure that K_CA not used as public identifier
- In practice, use secure hash functions to do this

Example: Proof

Suppose we have:

1. K_{CA} **signed** $($**key**(K_A) **speaksfor key**$(K_{CA}).A)$
2. K_A **signed action**$(\texttt{read}, \texttt{foo.txt})$

We want to derive:

$$\textbf{key}(K_{CA}).A \textbf{ says action}(\texttt{read}, \texttt{foo.txt})$$

Proof:

3. **key**(K_{CA}) **says** $($**key**(K_A) **speaksfor key**$(K_{CA}).A)$ (Says-I3 on 1)
4. **key**(K_A) **says action**$(\texttt{read}, \texttt{foo.txt})$ (Says-I3 on 2)
5. **key**$(K_{CA}).A$ **says action**$(\texttt{read}, \texttt{foo.txt})$ (Speaksfor-E2)

Example due to Mike Reiter

Authenticating requests

Suppose the reference monitor receives a request:

$$p \textbf{ says action}(s)$$

The monitor has a policy that enumerates:

1. A set of principals P
2. The subset of principals $P_s \subseteq P$ authorized to perform s

The reference monitor needs a proof that:

$$p_a \textbf{ says action}(s), \text{ for some } p_a \in P_s$$

Authenticating requests

There are three basic approaches for doing this.

1. **Push**: The sender of the request collects certificates necessary to prove p_a **says action**(s), and sends them with the request. The monitor then finds a proof, if one exists.
2. **Pull**: The reference monitor searches for a set of certificates sufficient to prove p_a **says action**(s), and constructs the proof.
3. **Proof-carrying**: The sender of the request collects the necessary certificates and constructs the proof of p_a **says action**(s) itself. This might be more efficient than having the server construct every proof.

Example

$$
\dfrac{\begin{array}{c}\texttt{mfredrik}\ \textbf{says}\\ \textbf{delegates}(\texttt{mfredrik}, \texttt{staff}, s)\end{array} \qquad \dfrac{\begin{array}{c}\texttt{mfredrik}\ \textbf{says}\\ (\texttt{sam}\ \textbf{speaksfor}\ \texttt{staff})\end{array} \qquad \texttt{sam}\ \textbf{says action}(s)}{\texttt{staff}\ \textbf{says action}(s)}}{\texttt{mfredrik}\ \textbf{says action}(s)}
$$

Centralized access control list

$$\dfrac{\texttt{mfredrik}\ \textbf{says}\ \textbf{delegates}(\texttt{mfredrik}, \texttt{staff}, s) \qquad \dfrac{\texttt{mfredrik}\ \textbf{says}\ (\texttt{sam}\ \textbf{speaksfor}\ \texttt{staff}) \qquad \texttt{sam}\ \textbf{says action}(s)}{\texttt{staff}\ \textbf{says action}(s)}}{\texttt{mfredrik}\ \textbf{says action}(s)}$$

In a traditional aceess control list implementation

- The highlighted parts are stored in the reference monitor
- They're part of the TCB, and not cryptographically signed

Pull authentication

$$\dfrac{\texttt{mfredrik}\ \textbf{says}\ \textbf{delegates}(\texttt{mfredrik}, \texttt{staff}, s) \qquad \dfrac{\texttt{mfredrik}\ \textbf{says}\ (\texttt{sam}\ \textbf{speaksfor}\ \texttt{staff}) \qquad \texttt{sam}\ \textbf{says action}(s)}{\texttt{staff}\ \textbf{says action}(s)}}{\texttt{mfredrik}\ \textbf{says action}(s)}$$

In pull-authentication scheme

- The red-highlighted parts are retrieved by the monitor
- The blue-highlighted part is sent by the requester
- The rest is computed by the monitor

Push authentication

$$
\dfrac{\texttt{mfredrik}\ \textbf{says}\ \textbf{delegates}(\texttt{mfredrik}, \texttt{staff}, s) \qquad \dfrac{\texttt{mfredrik}\ \textbf{says}\ (\texttt{sam}\ \textbf{speaksfor}\ \texttt{staff}) \qquad \texttt{sam}\ \textbf{says action}(s)}{\texttt{staff}\ \textbf{says action}(s)}}{\texttt{mfredrik}\ \textbf{says action}(s)}
$$

In push-authentication scheme

- The red-highlighted parts are sent by the requester
- The rest is computed by the monitor

Proof-carrying authentication

$$\frac{\texttt{mfredrik}\ \textbf{says}\ \textbf{delegates}(\texttt{mfredrik},\texttt{staff},s) \qquad \dfrac{\texttt{mfredrik}\ \textbf{says}\ (\texttt{sam}\ \textbf{speaksfor}\ \texttt{staff}) \qquad \texttt{sam}\ \textbf{says action}(s)}{\texttt{staff}\ \textbf{says action}(s)}}{\texttt{mfredrik}\ \textbf{says action}(s)}$$

In push-authentication scheme

- The red-highlighted parts are sent by the requester
- The conclusion is **verified** by the monitor
- If the proof checks out, authorization is granted

Next lecture

- More authorization logic
- Application to secure boot, TLS authentication
- Revocation
- Extensions to the logic