The Role of Trust Management in Distributed Systems Security

(KeyNote)

Darrell Hyatt
Introduction

- For secure distributed systems, ACLs are inadequate
  - Password-based protocols are insecure in a networked environment
  - Centralized access control is inefficient
  - Not flexible enough for future uses
  - Trust policy is localized

- As of this writing, not a whole lot of alternatives
- Except maybe Trust Management
Trust Management

- Suggested by Blaze, Feigenbaum, and Lacy in 1996
- Does a set of credentials prove that the request complies with the local security policy?
- Requires the use of a general-purpose, application-independent algorithm for checking compliance
- Questions to answer when designing:
  - What constitutes “proof of compliance”?
  - Should policies or credentials be fully or only partially programmable?
  - Where are responsibilities divided between the calling application and the TM engine?
- Purpose is to advise applications, not enforce policy
PolicyMaker

- The first tool to be considered a trust-management engine, addressing authorization directly
- Assertions – fully-programmable credentials and policies, paired together as $(f, s)$
  - $f$ – program describing the authority being granted and who it is granted to
  - $s$ – the source of authority (e.g. POLICY)
- Credentials – must be signed & verified prior to use
- Trust root – the final authority source for the request
  - Composed of one or more policy assertions passed by the calling application
How PolicyMaker Works

Calling Application

- policy assertions, $P$
- credential assertions, $C$
- request, $r$

PolicyMaker Engine

proof of compliance
How PolicyMaker Works

- Proof of compliance - credential set $C$ contains a proof that request $r$ complies with policy $P$.
- Assertions can be written in any language that can be interpreted safely by an environment that must evaluate credentials from potentially untrusted sources
  - Early attempts to use AWK for this purpose
  - The engine's host must have an interpreter for the language, limiting some languages
- Calling application is responsible for:
  - all signature verification on credentials & requests
  - gathering all credentials needed to establish proof
- Can not use “negative credentials” like revocations
KeyNote

Design goals:

- Directly authorize action in full (no separation of identification and access control)
- Ease of integration into applications
- Standardization

More responsibility on the TM engine and less on the calling application

Specifies its own assertion language for credentials and policies to be written in

```plaintext
KeyNote-Version: 1
Authorizer: rsa-pkcs1-hex:"1023abcd"
Licensees: dsa-hex:"986512a1" ||
            rsa-pkcs1-hex:"19abcd02"
Comment: Authorizer delegates read access to either of the Licensees
Conditions: ($file == "/etc/passwd" &&
            $access == "read") ->
            {return "ok"}
Signature: rsa-md5-pkcs1-hex:"f00f5673"

Fig. 1. Sample KeyNote assertion
```
How KeyNote Works

Calling Application

KeyNote Evaluator

credentials
policies
requester public keys
action environment

application-defined string
How KeyNote Works

- Action environment – all other relevant information necessary for a trust decision on the request
  - Consists of name/value pairs
  - Identifying these attributes is the trickiest part when integrating KeyNote in new applications
  - Uses DFS search in an attempt to satisfy at least one of the assertions given

- Like PolicyMaker, can not handle revocations
- Unlike PM, specifies its own assertion language
- More work needs to be done regarding credential discovery and the revocation issue
Applications of Trust Management

- **Active Networks (programmable infrastructure)**
  - Keynote used to:
    - Authorize loading code & setting resource limits
    - Assert fine-grain control over what actions loaded code may take on a node
    - Request nodes behind a firewall to execute code for specific tasks

- **Distributed Access Control**
  - Beneficial to implement ACL on top of TM system
    - Easier to implement extension rules
    - Decouples AC policy from distribution and implementation mechanism
Applications of Trust Management

♦ Mobile-code Security
  - Executable content received by a host along with a request for execution → needs verification of trust
  - “a signature has value only to the extent that the verifier can evaluate the signer's trustworthiness.”
  - Trust Management plays two roles:
    • Express trust relationships & the conditions for which certification has meaning
    • Credentials could describe the minimum capability set the code needs to perform its tasks (used to sandbox the code)
Conclusions

♦ Advantages of Trust Management:
  – Direct authorization of security-critical actions
  – Decouples authorization from credential distribution

♦ Areas yet to be addressed:
  – Automated credential discovery
  – Handling of revocation lists
  – Abstracted, human-understandable policy languages
  – Tools to translate credentials to application-native forms