CERIAS

the center for education and research in information assurance and security

A System for the Specification and Enforcement of Quality-based Authentication Policies

A. Squicciarini, A. Czeskis, E. Bertino, A. Bhargav-Spantzel, M. Almomen

The Problem

There is no current method which permits the basing of access decisions on the system's certainty of a user's identity. This is a problem because not all authentication mechanisms have the same strength. Consider a 3 letter password versus a fingerprint. This could result in the following scenario:

A user authenticated five hours ago with a three letter password can access the same sensitive resources as if he were authenticated two minutes ago with a fingerprint.

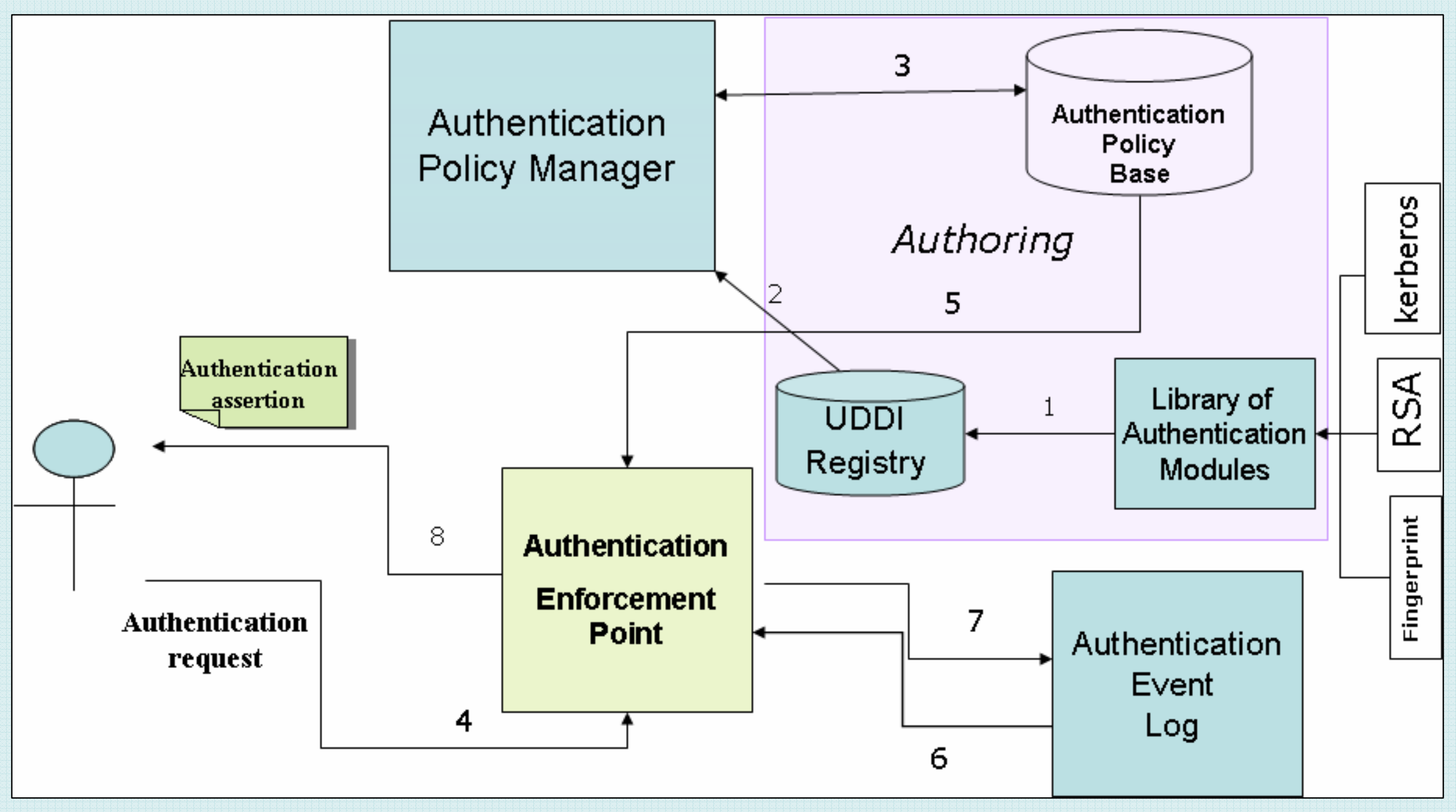
Overview of the Solution

Our solution is based on *authentication policies*. Policies specify:

- How many authentication factors are needed to access a resource.
- Which type of factors (ie. fingerprint, iris scan, password, smartcard,..)
- What time constraints – how long ago was the last authentication?
- Constraints on the authorities by which credentials used for authentication have to be provided.

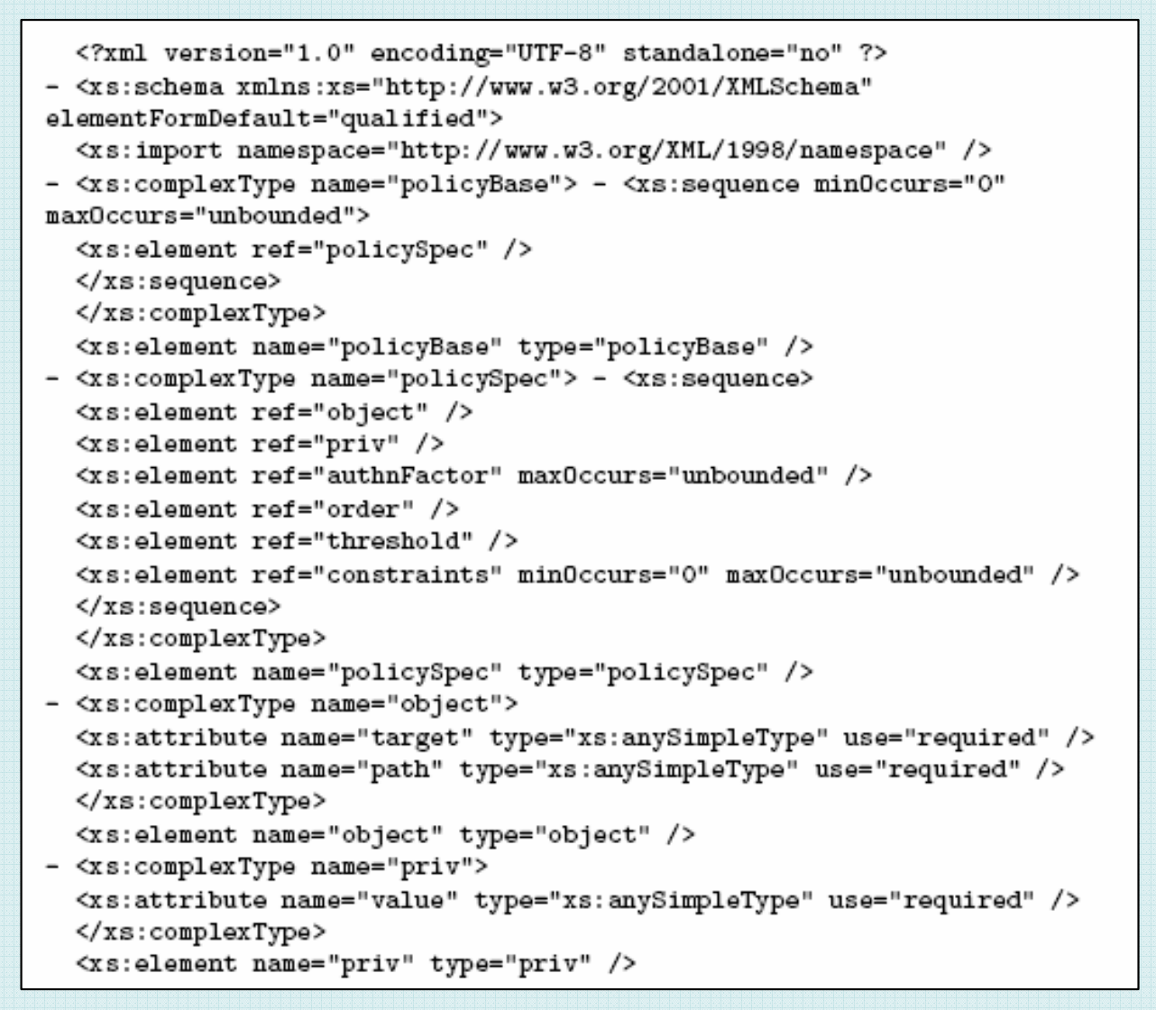
The idea is to:

1. A priori - associate policies with resources. (*Authoring* box in diagram below)
2. Remember user's authentication history. (*Authentication Event Log*)
3. When user requests a resource (performed by *Authentication Enforcement Pt*)
   a) Grab user's authentication history
   b) Grab policy associated with resources
   c) Compare them
      i. Success = allow for access control to take place
      ii. Failure = try to re-authenticate the user to satisfy policy.



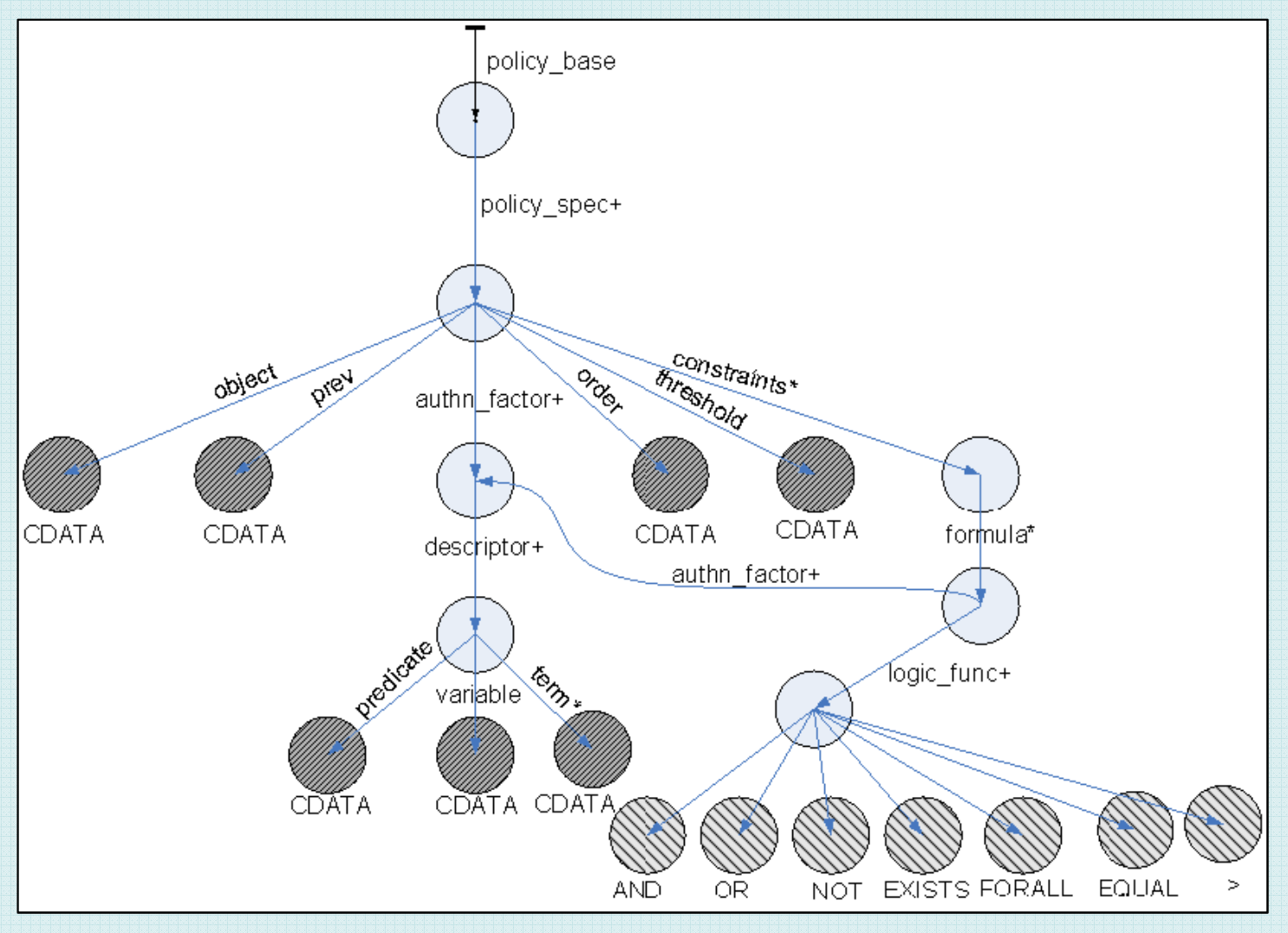
The Language

- We developed an articulated framework for authentication based on an expressive *authentication policy language*. By using such language, one can specify how many authentication factors are required.
- *The goal of our language is to specify policies driving authentication decisions*; as such policies expressed in our language may also take into account previous authentication decisions, taken for example by other sites in a distributed system, together with other information in order to reach an authentication decision.



```
  <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" />
- <xs:complexType name="policyBase"> - <xs:sequence minOccurs="0"
maxOccurs="unbounded">
  <xs:element ref="policySpec" />
  </xs:sequence>
  </xs:complexType>
  <xs:element name="policyBase" type="policyBase" />
- <xs:complexType name="policySpec"> - <xs:sequence>
  <xs:element ref="object" />
  <xs:element ref="priv" />
  <xs:element ref="authnFactor" maxOccurs="unbounded" />
  <xs:element ref="order" />
  <xs:element ref="threshold" />
  <xs:element ref="constraints" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  </xs:complexType>
  <xs:element name="policySpec" type="policySpec" />
- <xs:complexType name="object">
  <xs:attribute name="target" type="xs:anySimpleType" use="required" />
  <xs:attribute name="path" type="xs:anySimpleType" use="required" />
  </xs:complexType>
  <xs:element name="object" type="object" />
- <xs:complexType name="priv">
  <xs:attribute name="value" type="xs:anySimpleType" use="required" />
  </xs:complexType>
  <xs:element name="priv" type="priv" />
```

A snapshot of the XML template corresponding to the formal authentication policies



Authentication Policy Template Graph Representation

The Implementation

We implemented our solution using FreeBSD 6.1-RC.

Policy Storage and Binding:

- Have a predefined repository of policies each with unique ID.
- Resource stores ID of policy and several parameters for policy interpretation in an extended attribute of the vnode. Thus policies are directly associated with the resources to which they pertain.

Authentication Log:

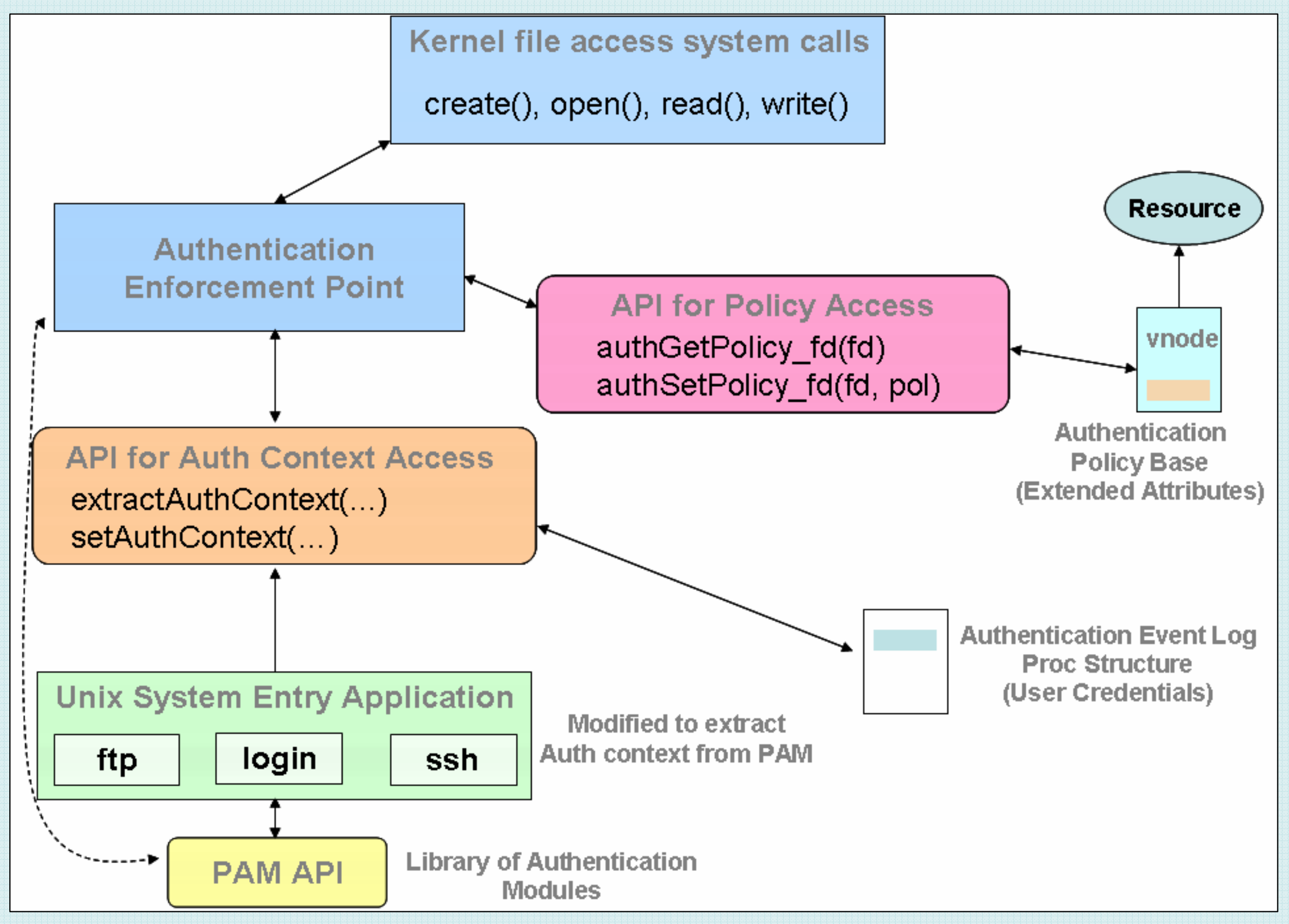
- Maintain log of a user's authentication in the proc structure of the process.
  - o Added a structure to the ucred struct (the ucred maintains other information such as the user id, group id, etc...)

Policy Enforcement:

- When system calls are invoked to access a particular resource, the *Authentication Enforcement Point* is activated. This is done prior to any authorization check to be sure who the user is before applying any access control mechanism. The *Enforcement Point* verifies that the user's authentication history satisfies the criteria set forth by the policy associated with the requested resource.

Pluggable Authentication Modules (PAM):

- PAM organizes multiple authentication mechanisms into a single API.
- Used by system entry applications to authenticate the user.
- Since PAM is designed to be completely transparent, we had to modify it in order to accommodate our solution:
  - o Added code to a PAM module to export authentication details such as: type of mechanism, time of authentication, number of authentication tries, threshold[1], TTL, storage location (local or remote), and storage mode (encrypted, plain text, etc...).


Kernel file access system calls
create(), open(), read(), write()
Resource
Authentication Enforcement Point
API for Policy Access
authGetPolicy_fd(fd)
authSetPolicy_fd(fd, pol)
vnode
Authentication Policy Base (Extended Attributes)
API for Auth Context Access
extractAuthContext(…)
setAuthContext(…)
Authentication Event Log Proc Structure (User Credentials)
Unix System Entry Application
ftp
login
ssh
Modified to extract Auth context from PAM
PAM API
Library of Authentication Modules

[1]This parameter is somewhat context dependent. For example, in the case of password authentication, this is the minimum password length allowed by the authentication module. In the case of a fingerprint, this is the percent match, as specified by the module settings, necessary for a match.

Benchmarking

We considered three simple policies:

1. First with one a single factor
2. The second with two factors and zero constraints
3. The third with two factors and one constrain binding the two factors.

Each policy is composed of two factor assertions, and refers to a password authentication mechanism. The results show that our implementation does not introduce significant latency.

Time to open a file 10,000 times (milliseconds)

Policy	Baseline No policy	Successful Authentication	Unsuccessful Authentication
1	112	185	85
2	120	169	97
3	118	201	103

Intel(R) Xeon(TM) 2.80GHz CPU with 1 GB of RAM

Future Work

Policies:

- Develop different policies for different actions on resource. That is, one policy will apply if the user is reading a file and a different policy if a user is writing to the file.
- Develop different policies for different users, groups, and roles.

Implementation:

- Develop and test more complex policies.
- Develop re-authentication capability and export policies into a separate library.
- Possible extension to distributed/federated system.

Acknowledgements

We would like to thank Keith Watson for his advice on kernel development and modification.

PURDUE UNIVERSITY

CERIAS

Discovery Park e-Enterprise Center