Integrated mandatory access control for digital data

Hsieh, George, Patrick, Gregory, Foster, Keith, Emamali, Gerald, Marvel, Lisa

George Hsieh, Gregory Patrick, Keith Foster, Gerald Emamali, Lisa Marvel, "Integrated mandatory access control for digital data," Proc. SPIE 6973, Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008, 697302 (17 March 2008); doi: 10.1117/12.777135

SPIE.

Event: SPIE Defense and Security Symposium, 2008, Orlando, Florida, United States

"Integrated mandatory access control for digital data

George Hsieh[*a], Gregory Patrick[a], Keith Foster[a], Gerald Emamali[a], Lisa Marvel[b]
[a]Norfolk State University, 700 Park Ave., Norfolk, VA, USA 23504;
[b]U.S. Army Research Laboratory, APG, MD, USA 21005

ABSTRACT

This paper presents an integrated mandatory access control (MAC) framework that incorporates MAC mechanisms at both operating system and application layers for digital data. The framework uses Security-Enhanced Linux (SELinux) as the foundation for MAC at the operating system layer. It uses XACML (eXtensible Access Control Markup Language) as the base mechanism for specifying and embedding information-layer MAC policies. This framework is designed to be general-purpose, flexible, and capable of providing fine-grained access control. This paper also describes a high-level architecture of a prototype being developed for the framework. One targeted application domain for this framework is information sharing and dissemination in a multi-level security environment.

Keywords: mandatory access control, XACML, SELinux, XML security, information security

1. INTRODUCTION

Security-Enhanced Linux (SELinux) [1-5] implements a very flexible and powerful Mandatory Access Control (MAC) mechanism that is enforced by the operating system. The mainstream security server provided with SELinux implements a combination of Role-Based Access Control (RBAC), a generalization of Type Enforcement (TE), and optionally Multi-Level Security (MLS) [6-7] which uses sensitivity (clearance) and category (need-to-know) labels to enforce security policy. The MAC capabilities can be used to provide protection for various digital objects, including system files, user files, etc.

However, the SELinux's MAC mechanism is not privy to the contents of those objects. It is necessary to extend mandatory access control to the contents of digital objects (e.g., documents), such that security policies regarding the content can be specified for and enforced by an integrated MAC environment that encompasses both operating system and application layers. Examples include the prevention of printing, copying, transmitting over a network, etc. of protected documents. Other examples include automatically redacting portions of documents for which the requesters do not have the prerequisite credentials, and tracking the usage of documents (who used them, what modifications were made, date and time of accesses, etc.)

It is also desirable to implement a "tagging" mechanism such that security policies can be embedded with the digital objects themselves as they are created, processed, and distributed. This embedded approach allows for all access control and tracking information to reside with the digital content as one entity, and hence alleviates the problems commonly associated with managing these different types of information in separate entities.

Furthermore, it is desirable to implement fine-grained access control such that different policies can be applied to and enforced for different portions of the digital data. With this capability, only a single version of the digital data needs to be maintained while different "views" of the digital data can be generated from the same source with each view meeting the requirement of a specific authorization policy. This capability can be used to facilitate the management and sharing of data with multiple levels of authorization, without the need to maintain multiple, separate versions of the data.

The overall objectives for the integrated MAC framework described here are:

1) Extend MAC to the application layer while leveraging SELinux for MAC at the operating system layer;
2) Implement a tagging mechanism to facilitate the embedding of MAC policy with the data; and
3) Implement fine-grained MAC to facilitate the management and sharing of information using one version of the data object.

[*] ghsieh@nsu.edu; phone 1 757 823-8313; fax 1 757 823-9229; nsu.edu

Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008, edited by Belur V. Dasarathy, Proc. of SPIE Vol. 6973, 697302, (2008) · 0277-786X/08/$18 · doi: 10.1117/12.777135

One targeted application domain for this framework is information sharing and dissemination in a multi-level security environment.

Central to our conceptual framework is the eXtensible Access Control Markup Language (XACML) which is a set of XML-based languages and profiles for access control applications. It has been standardized by the OASIS (Organization for the Advancement of Structured Information Standards). The first version of XACML specifications, 1.0, was published in February 2003, and the current version, 2.0, in February 2005 [8-10].

XACML describes both an access control policy language and a request/response language [11, 12]. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries (requests) about whether a particular access should be allowed and describes answers (responses) to those queries.

We chose XACML as the base mechanism for tagging/embedding digital data with access control policy information, because XACML provides an extensible, flexible, highly expressive, standards-based, and general-purpose access control policy language that can be used for controlling access to any type of resource [11, 13, 14], not just XML documents. In addition, there are other XML based security standards that can be leveraged to enhance the confidentiality, integrity, and authenticity of information. These include the XML Encryption Syntax and Processing [15] and XML Signature Syntax and Processing [16], both from the W3C (World-Wide Web Consortium). Integration of these technologies (access control, encryption, and signature) becomes significantly easier with XML being the common base language for them all.

We then extended the standard XACML languages and processing models to allow digital data be integrated with the access control policy in the same XACML-based document, which serves as a container for both types of information. Our approach involves transforming the digital data from its original format into a valid XACML document. The XACML document is composed of XACML policy language tags, as well as the content of the original data which is encoded into the Base-64 format. The original data can be further divided into multiple "parts", each of which is encapsulated by its own XACML policy language tags that specify the access control policies specific to this part. This multi-part organization and part-specific access control policies form the basis for the fine-grained access control.

1.1 Related Works

Our embedding/tagging concept is very similar to that of the Enterprise Rights Management (ERM). The Enterprise Strategy Group (ESG) in a published white paper [17] defines ERM as "A digital document-based security model that enforces access, usage, confidentiality, and storage policies." To highlight the benefits of ERM solutions, ESG describes a major problem with today's confidential data security solutions: data is virtually wide open when it resides somewhere between the security checkpoints such as desktop encryption, gateway filters, etc. To address this problem, "ESG believes that the only way is by integrating security into the confidential data files themselves. In other words, electronic documents should be extended so that they can carry persistent security policies that define and enforce document access, usage, confidentiality and storage rules regardless of device type or physical location." "Unlike traditional access controls, ERM governs the access and use of content at all times regardless of disposition or location: creation, replication, transmission, consumption, modification, expiration, etc." Liquid Machines, Inc. also discusses the benefits of ERM and their commercial solutions in an Information Brief [18].

The ERM architecture described in [17, 18] is built upon document-based policies capable of providing granular security policy enforcement. This is very similar to our tagging/embedding concept. However, our method is based on XACML which is an open standard; while ERM and a related Secure Information Sharing Architecture (SISA) [19] are built on top of Microsoft's Windows Rights Management Services (RMS) which is an information protection technology that works with RMS-enabled applications to help safeguard digital information from unauthorized use [20].

Our fine-grained XACML-based approach is also inspired by previous works on XML document security and XML-based access control languages, specifically the Fine-Grained Access Control System [21, 22] developed by E. Damiani et al., and the XML Access Control Language (XACL) [23] developed by M. Kudo and S. Hada at the IBM Tokyo Research Laboratory. Their research led to the development of the XACML standard by the OASIS organization [24].

The standard XACML languages and processing models do not support the document-based integrated data/policy model and fine-grained access control capability. Our research builds upon and extends the XACML specifications to add support for these two features.

1.2 Outline of the Paper

This paper is organized as follows. We present our Version 1 conceptual framework in Section 2. In Section 3, we discuss our initial prototype implementation focusing on the XACML processing. Section 4 describes ideas for future work to extend the framework and prototype. Finally, we conclude this paper with a brief summary in Section 5.

2. VERSION 1 CONCEPTUAL FRAMEWORK

In this section, we first present a brief introduction to XACML and an analysis comparing it with the security model/language for SELinux. We then describe the extensions we made to the standard XACML policy and request/response languages to support our embedding and fine-grained access control methods. We next summarize the modifications and extensions we made to the XACML processing models. We conclude this section with a discussion on how the XACML-based information-level access control method can be integrated with SELinux and application environments.

There are strong similarities between SELinux and XACML in terms of the security architectural model and the policy language. Both support an architectural model of separating the policy decision logic from the policy enforcement logic. Three key logical functions are defined by XACML. A *Policy Decision Point (PDP)* is an entity that evaluates applicable policy and renders an authorization decision. The PDP function in XACML is similar to the *Security Server* [1] which is a SELinux kernel component that encapsulates its security policy decision logic. An XACML *Policy Enforcement Point (PEP)* is an entity that performs access control by making decision requests to a PDP and enforcing the authorization decisions returned by the PDP. A PEP is comparable to a security *hook* function within the SELinux kernel that is used to mediate access to internal kernel objects. The third XACML logical function, *Policy Administration Point (PAP)*, is an entity that creates and manages access control policies.

In a typical XACML usage scenario, a *subject* (e.g., user, application, or process) wants to take some *action* on a particular *resource* [11]. Conceptually, an XACML subject is comparable to a SELinux subject, an XACML action to a SELinux operation, and an XACML resource to a SELinux object [1]. The subject submits its query to the PEP that is responsible for protecting the requested resource (e.g., file system, or web server). The PEP forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other environmental information. The PEP then sends this XACML request to a PDP which examines the request, retrieves policies (written in the XACML policy language) that are applicable to this request, and determines whether access should be granted according to the XACML rules for evaluating policies. That answer (expressed in the XACML response language) is returned to the PEP, which can then allow or deny access by the subject.

The programmer's guide for Sun Microsystems' XACML implementation [11]

The base construct of all XACML policies is a Policy or a PolicySet. A *Policy* represents a single access control policy, expressed through a set of Rules. Each XACML policy document contains exactly one Policy or PolicySet root XML tag. A policy can have any number of *Rules* which contain the core logic of an XACML policy. The decision logic of most rules is expressed in a *Condition*, which is a Boolean function. If the condition evaluates to true, then the Rule's Effect (Permit or Deny) is returned. If the Condition evaluates to false, then the Condition doesn't apply (NotApplicable). XACML includes a number of built-in functions and a method of adding non-standard functions.

XACML policies operate upon attributes of Subject, Resource, Action and Environment in order to arrive at an authorization decision. For example, a user's name, group membership, a file the user wants to access, and the file read operation are all attribute values. *Environment* attributes are independent of a particular subject, resource or action, and they represent other information (e.g., the time of day during which access is permitted) that are relevant to an authorization decision. When a request is sent from a PEP to a PDP, that request is formulated almost exclusively of attributes, and they will be compared to attribute values in a policy to make the access decisions. The core XACML standard specifies a set of predefined attributes such as subject:subject-id, subject-category, resource:resource-id, action:action-id, environment:current-dateTime, etc. In addition, OASIS has specified a role based access control (RBAC) profile [10] for the use of XACML in expressing policies that use RBAC. Furthermore, users can define additional attributes or attribute values to meet their needs.

An XACML PDP is responsible for finding policies that apply to a given request. To do this, XACML provides a feature called a Target. A *Target* is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a Policy or Rule to apply to a given request. These elements use Boolean functions to compare values found in a request with those included in the Target. If all the conditions of a Target are met, then its associated Policy or Rule applies to the request. Once a Policy is found, and verified as applicable to a request, its Rules are evaluated.

XACML also provides a collection of *Combining Algorithms*, each of which presents a different way of combining multiple decisions, rendered by multiple rules contained in a Policy, into a single decision. An example of these is the Permit Overrides Algorithm, which says that no matter what, if any evaluation returns Permit, then the final result is also Permit. These Combining Algorithms are used to construct complex policies. XACML provides several standard algorithms, and custom algorithms can be developed to meet additional needs.

Figure 1 shows an example XACML policy document which specifies that a subject with the credential of belonging to the *developers* group will be granted permission to *read* the file identified by *http://ww.cs.nsu.edu/index.php*.

```
<Policy PolicyId="ExamplePolicy" RuleCombiningAlgId="…:permit-overrides">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="…:anyURI-equal">
          <AttributeValue>http://www.cs.nsu.edu/index.php</AttributeValue>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="ReadRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="…:string-equal">
            <AttributeValue>read</AttributeValue>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
    <Condition FunctionId="…:string-equal">
      <Apply> <SubjectAttributeDesignator AttributeId="group"/> </Apply>
      <AttributeValue>developers</AttributeValue>
    </Condition>
  </Rule>
</Policy>
```

Fig. 1. XACML policy language example.

Figure 2 shows an example XACML request to read the file identified by http://www.cs.nsu.edu/index.php by a user ghsieh@nsu.edu who also belongs to the developers group.

```
<Request>
  <Subject>
    <Attribute AttributeId="…:subject:subject-id" …>
```

```
      <AttributeValue>ghsieh@nsu.edu</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group" …>
      <AttributeValue>developers</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="…:resource:resource-id" …>
      <AttributeValue>http://www.cs.nsu.edu/index.php</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="…:action:action-id" …>
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Fig. 2. XACML request language example.

Figure 3 shows an example XACML response indicating that the access request (shown in Figure 2) is granted permission based on the access control policy (shown in Figure 1).

```
<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="…:status:ok"/>
    </Status>
  </Result>
</Response>
```

Fig. 3. XACML response language example.

2.1 Extensions of XACML languages

To support our embedding approach, we extended the XACML policy language to allow a <ResourceContent> element within a <Resource> element. The standard XACML has already defined a <ResourceContent> element as a valid construct for the request language, and it is typically used by a PEP to transmit the resource content "in-line" in the request message to a PDP. However, the <ResourceContent> element is not part of the standard XACML policy language. Hence, our embedding method requires a modification of the schema definition for the XACML policy language such that a <ResourceContent> element can be a valid descendant within a <Resource> element.

In addition, we designed a set of structural and feature conventions for the format of the XACML document such that it can be used as a container for the digital data and embedded security policies capable of supporting fine-grained access control. Figure 4 shows an example XACML document incorporating our policy language extension and conventions.

```
<Policy RuleCombiningAlgId="…:permit-overrides">
  <Target>
     …
     …
  </Target>

  <Rule RuleId="1", Effect="Permit">
    <!-- Access Control Rule & Content for First Part -->
    <Target>
        …
      <Resource>
        <ResourceContent> … Base-64 Encoded Data Part One … </ResourceContent>
      </Resource>
        …
    </Target>
    <Condition> … Access Control Rule One … </Condition>
  </Rule>
```

```
  <Rule RuleId="2", Effect="Permit">
    <!-- Access Control Rule & Content for Second Part -->
    <Target>
        …
      <Resource>
        <ResourceContent> … Base-64 Encoded Data Part Two … </ResourceContent>
      </Resource>
        …
    </Target>
    <Condition> … Access Control Rule Two … </Condition>
  </Rule>

  <!-- Deny all other cases -->
  <Rule RuleId="final", Effect="Deny" />
</Policy>
```

Fig. 4. Example XACML incorporating our policy language extension and conventions.

This example XML document uses an XACML <Policy> element to encapsulate the digital data and the access control policies. The data is divided into two separate parts, each of which has its own access control policy. Each part of the data and its associated access control policy are encapsulated in a separate <Rule> element. The first <Rule> element with RuleID="1" is defined for the first part of the data, while the second <Rule> element with RuleID="2" is defined for the second part of the data. Each of these two rules includes a <Condition> element to specify the conditions, when all met, a request is to be permitted for this part of the data. As described above, the <Condition> elements contain the core decision logic for their encompassing <Rule> elements.

Each of these two <Rule> elements also contains a <Target> element. In our approach, the role of this <Target> element is simplified, because a PDP no longer needs it to determine whether this Policy or Rule is applicable to the request. This is because the Policy or Rule embedded with the digital data in this XACML document is, by definition, applicable to this request for accessing the data. The primary use of the <Target> element is to contain a <ResourceContent> element, nested within a <Resource> element, to encapsulate the original data encoded into the Base-64 format. Access to the data contained within the <ResourceContent> element will be regulated by the policy expressed in the encompassing <Rule> element.

The example XACML document also shows a third <Rule> element with RuleID="final" and Effect="Deny". This rule is intended to be an explicit "deny all", similar to access control rules used by routers or firewalls. The document also uses "permit-overrides" as the rule combining algorithm for the root <Policy> element, such that a policy decision of Permit is rendered if there is at least one Permit decision returned by any of the rules (i.e., access to at least one part of the original data is permitted).

To support our embedding and fine-grained access control methods, we further extended the XACML response language which defines a root <Response> element containing one or more Results. Each <Result> contains a <Decision> (Permit, Deny, NotApplicable, or Indeterminate), some <Status> information (for example, why evaluation failed), and optionally one or more Obligations (things that the PEP is obligated to do before granting or denying access). Our extension was designed to allow the "authorized content" be transported "in-line" within the <Response> from the PDP to the PEP. This was accomplished by extending the schema definition for the <Response> element to add the <ResourceContent> element (with zero or more occurrences) as a valid descendant type. Each <ResourceContent> element within <Response> will encapsulate one part of the "authorized content". Upon receiving this response message, the PEP can convert the authorized content from the Base-64 format, and concatenate the multiple authorized parts into one data stream that is accessible by the requester.

In summary, our extensions to the XACML languages involve only minor schema changes to allow the already-defined <ResourceContent> element be used in the policy language and the response language (in addition to the request language). There are no other changes to the syntax or semantics of the languages.

2.2 Modifications and extensions of XACML processing models

In our model, a PDP needs to locate and select the applicable policy in a different manner. A conventional XACML setup is that the resources to be protected and the access control policies are maintained as separate entities. A commonly used method is for a PEP to include a Uniform Resource Identifier (URI) in the request message such that the

receiving PDP can identify the resource (which the PDP may not need to access). The PDP needs access to a policy store which may contain a large number of policies (and maybe only one of which is applicable to a particular request).

Our embedding method, on the other hand, uses a single XACML document to contain both the resource content and its associated access control policies. Hence, the PDP in our model needs to access the security policies embedded within the XACML document itself. For the PDP to locate and access this XACML document, a PEP can include a URI for this document as a Resource attribute in the XACML request message. This is the same as the URI mode supported by the standard XACML.

Our PDP also needs to accumulate those <ResourceContent> elements each of which was evaluated to "permit" in a rule section. Furthermore, it needs to evaluate all the rules in the XACML document such that it can find all parts of the data that are authorized for this request. A conventional PDP, on the other hand, may stop evaluating any remaining rules once it determines that an overall decision of "permit" or "deny" is reached. After all the rules are evaluated and a "permit" decision is reached, our PDP needs to include all those accumulated <ResourceContent> elements in the response message to the PEP.

The PEP, in our model, needs to retrieve all <ResourceContent> elements, if any, from the <Response> message, convert the data to its original format from the Base-64 encoded format, and concatenate all parts of data into one data stream.

In summary, the standard XACML languages, profiles and processing models can be adopted, with minor modifications and without losing original capabilities, to support our fine-grained embedding method. As a result, our approach can fully leverage the benefits of XACML, and inherit its strengths such as being standards-based, generic, and extensible.

2.3 Integration with SELinux and applications

Various software modules, API's, libraries and tools are needed to integrate this access control and embedding method with different operating system and application environments. Capabilities are needed to support the creation of XACML-based documents embedded with access control policies and resource content, to retrieve only the authorized content from this type of document given an access request, and to enforce the access control policy.

These modules include a potentially general-purpose PDP component, which examines an access request, accesses the document referenced by the URI, evaluates the embedded policies (written in the XACML policy language) in conjunction with the request context, and determines if any part of the content is permitted for access according to the XACML rules. The decision on Permit or Deny as well as a data stream containing only the authorized content is then returned by the PDP in a response to the request.

On the other hand, different software modules and API's are required for the XACML document creation function to support integration with different document authoring tools (e.g., Open Office, Microsoft Word, Adobe Acrobat), such that integrated data/policy documents can be created without requiring users to code in the XACML language. Similarly, different modules are required for different types of PEP's which represent the entities (e.g., file system, web server) that need to access the protected data and enforce access control.

Integration of this XACML-based access control and embedding method with SELinux involves several dimensions. First, SELinux can be used to secure the XACML processing software and pipeline. Second, SELinux can be used to secure the integrated policy/data document by enforcing file-level MAC. It is very important to secure the XACML documents which contain both access control policies and data. With appropriate SELinux security policy configuration, access to these documents can be restricted to only those authorized software modules (e.g., a PDP, a web server which serves as a data distribution system and a PEP). It is also very important to secure the different "views" derived from the same XACML document, such that each view (a data stream) is accessible only by authorized users without leakage. Again, SELinux security policies can be used to control access at the operating system and the stream level.

Another dimension of integration is to implement a consistent security model and policy vocabulary at both operating system and application levels. This will allow tighter integration and provide better usability. The commonly used security models such as MLS, MCS and RBAC are supported (or can be easily supported with extensions/profiles) by both SELinux and XACML. The decision of which model or combination of models to use can be made to meet the needs of the targeted application.

3. A PROTOTYPE IMPLEMENTATION

We are developing an initial prototype of this framework for proof-of-concept purpose. It focuses on the XACML processing with a sample implementation of the PDP, PEP, and PAP functions. Each sample function is implemented as a separate Java program. All these programs are developed by leveraging the source code from Sun Microsystems' open source XACML software implementation [11]. We also use open source XML and Java development tools, such as Apache Xerces-J and Xalan-J as well as Eclipse Java IDE.

Figure 5 shows the architecture of the initial prototype. The prototype PDP is built on Sun Microsystems' XACML code base, with new/modified code to accommodate our extensions and modifications, such as policy selection and invocation, authorized content extraction during policy evaluation, and including authorized content in the response message. The prototype PEP is a simple implementation to show how a request message can be constructed from user input, how this message can be transmitted to the prototype PDP, and how the response message can be received and the authorized content extracted from the response message. The communication between the PDP and the PEP is via a socket interface, and the extended XACML request/response languages are used as the message-level protocols.

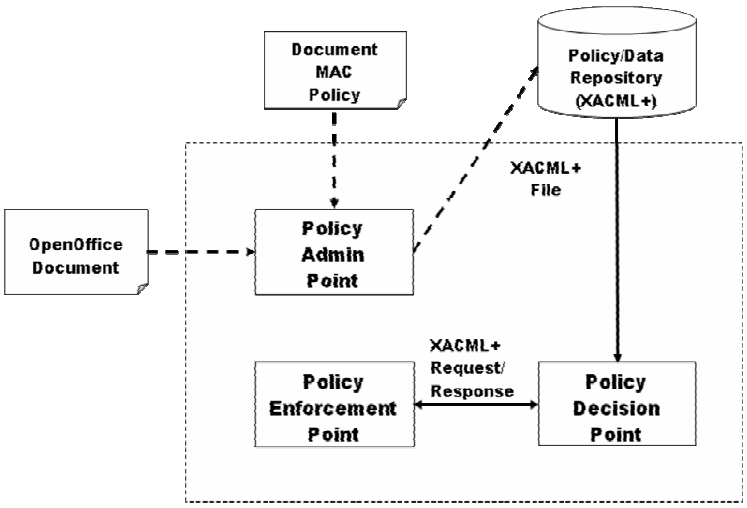

Fig. 5. Architecture of the initial prototype implementation.

The prototype PAP is intended to illustrate how it could be integrated with a document authoring tool such as OpenOffice.org office suite [25]. The prototype PAP takes as input a document produced by the OpenOffice.org office suite, which is already in XML format conforming to the OASIS Open Document Format standard [26]. The author of this OpenOffice document can embed user-defined tags for specifying access control policies for different parts of the document, using a template which represents an organization's document MAC policy. The PAP transforms the OpenOffice document into an XACML (with our extensions) document, and stores it on the local disk. This XACML document is then accessed by the PDP when it receives the document's URI in a request message from the PEP.

The implementation of the initial prototype is nearly complete and is undergoing final integration and system testing. Our experiences show that the code modifications required for the PDP to accommodate our framework are very minimal and localized, given the solid foundation of Sun Microsystems' XACML code base. The code changes to the core XACML processing functions in PEP are also very minimal and localized. On the other hand, the XACML policy creation function is more complex and challenging, having to work with XACML policy language's particular (and somewhat cumbersome) constructs, and to create policy documents that are generic, accurate and efficient.

4. FUTURE WORK

We plan to continue extending this framework to enhance its functionality, usability, robustness, and security. Incorporation of XML security (encryption and digital signature) syntax and methods, as well as effective cryptography

and key management methods, into the framework is of the highest priority. Integration with SELinux is also of high interest. Integration with enterprise-scale authentication and authorization infrastructures, such as Security Assertion Markup Language (SAML) [27] and identity management systems, will enhance the functionality and usefulness of this framework. Incorporation of PEP functionality into an open source web publishing framework (e.g., Apache Cocoon) will further illustrate the concept of this integrated approach for information sharing and dissemination in a multi-level security environment.

5. SUMMARY

In this paper we present an integrated mandatory access control framework that leverages SELinux to provide MAC at the operating system layer. In addition, the system uses an XACML-based access control and embedding method to support fine-grained access control at the application layer. This method is flexible, extensible, powerful, and generic so it can accommodate a wide spectrum of access control policy needs and new requirements. It is also based on open standards, so it can leverage many software and tools available through both commercial and open-source communities. The XACML based embedding method can handle digital data of all formats, because it is independent of the format of the original data. It can also be used to ease the burden of managing digital data and its related access control policies, by maintaining both types of information (even for multiple authorization levels) in a single XACML document.

ACKNOWLEDGEMENT

This research was sponsored in part by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-05-2-0036. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

The authors also wish to acknowledge the U.S. Department of Education, Title III Program, for its funding support of this research effort.

REFERENCES

1. P.A. Loscocco and S. D. Smalley, "Meeting Critical Security Objectives with Security-Enhanced Linux," *Proc. of the 2001 Ottowa Linux Symposium*, July 2001.
2. C. Wright et al., "Linux Security Module Framework," *Proc. of the 2002 Ottawa Linux Symposium*, June 2002.
3. C. Wright et al., "Linux Security Modules: General Security Support for the Linux Kernel," *Proc. of the 11th USENIX Security Symposium*, August 2002.
4. S. Smalley, C. Vance, and W. Salamon, *Implementing SELinux as a Linux Security Module*, NAI Labs Report #01-043, Feb 2006.
5. Security-Enhanced Linux, National Security Agency, http://www.nsa.gov/selinux/
6. C. Hanson, "SELinux and MLS: Putting the Pieces Together," *Proc. of the Second Annual Security Enhanced Linux Symposium*, pp. 13-17, Feb/Mar 2006.
7. R. Coker, "MCS - adding MLS features to the targeted policy," *Proc. of the Second Annual Security Enhanced Linux Symposium*, pp. 71-76, Feb/Mar 2006.
8. OASIS eXtensible Access Control Markup Language (XACML) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
9. *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS Standard, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
10. *Core and hierarchical role based access control (RBAC) profile of XACML v2.0*, OASIS Standard, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf
11. Sun's XACML Implementation Programmer's Guide for Version 1.2, July 11 2004, http://sunxacml.sourceforge.net/guide.html

12. J. Rihak, *Access Control Markup Languages for XML Documents*, Semester Thesis, Department of Computer Science, Swiss Federal Institute of Technology Zurich, August 2004.
13. A. Anderson, "eXtensible Access Control Markup Language (XACML)," GSA Identity Workshop, Feb 2007.
14. A. Matheus, "How to declare access control policies for XML structured information objects using OASIS' eXtensible Access Control Markup Langauge (XACML)," *Proc. of the 38th Hawaii International Conference on System Sciences,* 2005.
15. *XML Encryption Syntax and Processing*, W3C Recommendation, 10 Dec 2002, http://www.w3.org/TR/xmlenc-core/
16. *XML-Signature Syntax and Processing*, W3C Recommendation, 12 Feb 2002, http://www.w3.org/TR/xmldsig-core/
17. J. Oltsik, *Enterprise Rights Management: A Superior Approach to Confidential Data Security*, white paper, Enterprise Strategy Group, May 2006.
18. *Beyond Full Disk Encryption*, Information Brief, Liquid Machines, April 2007.
19. Secure Information Sharing Architecture Alliance, http://www.sisaalliance.com/
20. *Technical Overview of Windows Rights Management Services for Windows Server 2003*, Microsoft Corporation, April 2005.
21. E. Damiani et al., "Securing XML Documents," *Proc. of the 7th Int'l Conf. on Extending Database Technology* (EDBT 2000), Springer Verlag, Berlin, 2000, pp. 121-135.
22. E. Damiani et al., "A Fine-Grained Access Control System for XML Documents," *ACM Transactions on Information and System Security*, 5(2), 169-202 (2002).
23. M. Kudo and S. Hada, "XML Document Security Based on Provisional Authorization," *Proc. of 7th ACM Conf. Computer and Communication Security* (CCS 2000), ACM Press, New York, 2000, pp. 87-96.
24. E. Damiani et al., "Controlling Access to XML Documents," *IEEE Internet Computing*, 5(6), 18-28 (2001).
25. OpenOffice.org, http://www.openoffice.org/
26. Open Document Format for Office Applications (OpenDocument), v1.0, OASIS Standard, May 2005, http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf
27. OASIS Security Services (SAML) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security