Security Policies to Mitigate Insider Threat in the Document Control Domain[1]

Suranjan Pramanik, Vidyaraman Sankaranarayanan and Shambhu Upadhyaya
Department of Computer Science and Engineering
201 Bell Hall
University at Buffalo, Buffalo NY 14260
Phone: (716)645-3180; Fax: (716)645-3464
Email: {pramanik, vs28, shambhu}@cse.buffalo.edu

Abstract

With rapid advances in online technologies, organizations are migrating from paper based resources to digital documents to achieve high responsiveness and ease of management. These digital documents are the most important asset of an organization *and are hence the chief target of insider abuse. Security policies provide the first step to prevent abuse by defining proper and improper usage of resources. Coarse grained security policies that operate on the "principle of least privilege" [1] alone are not enough to address the insider threat, since the typical insider possesses a wide range of privileges to start with. In this paper, we propose a security policy that is tailored to prevent insider abuse. We define the concept of subject, object, actions, rights, context and information flow as applicable to the document control domain. Access is allowed based on the principles of "least privilege and minimum requirements", subject to certain constraints. Unlike existing techniques, the proposed policy engine considers, among other factors, the context of a document request and the information flow between such requests to identify potential malicious insiders. Enforcing these fine-grained access control policies gives us a better platform to prevent the insider abuse. Finally, for demonstration purposes, we present a framework that can be used to specify and enforce these policies on Microsoft Word documents, one of the popular document formats.*

***Index terms*-- Insider Threat, Digital Documents, Access Control, Information Flow**

1. Introduction

With the evolution of computerized data entry methodologies, organizations have attempted to move towards the paperless office model. Digital documents are at the core of these paperless offices. They have become a means to express and store any piece of information in an organization, ranging from open office memos to top-secret design documents. Protecting these documents is of paramount importance to the corporate network, for the compromise of these documents could be potentially devastating to the survival of the organization. The protection of these documents falls under the broad category of Digital Rights Management.

This paper refers to the generic architecture of a corporate network with users owning digital documents as the "Document Control Domain" – referred to as DCD. The users in the DCD have a need to collaborate and share the documents securely and with restrictions on the usage of the document contents. Existing solutions like Microsoft Information Rights Management for Microsoft™ Office 2003 [2], Authentica™ PageRecall [3] for Adobe Acrobat, etc., allow for the protection of the documents (in doc or pdf format). This broadly translates into encryption of the documents for secure electronic transmission and setting basic policies on the documents before they are sent out to other parties. These policies typically dictate if the receiving party can read or edit or print the document. They are coarse-grained and setting them is usually based on the discretion of the owner of the document. These policies reflect the "principle of least privilege implicitly" [1], i.e., by trusting that the human operator will grant the appropriate permissions. These existing solutions provide an effective line of defense against the classic hacker who tries to penetrate the organization from outside. They are however not at all useful when it comes to combating the malicious insider threat [4], [5]. For the purposes of this paper, we take the working definition of the malicious insider as an authorized user in a corporate network, (usually an employee), who has a conflict of interest with the organization. As an employee, he holds authorization for various activities relating to his job function. Hence

[1] Research supported in part by Advanced Research and Development Activity (ARDA), contract no. NBCHC030062.

his capability to cause damage to the organization ranks on a higher scale than the classic hacker.

Security policies in the DCD can be thought of as the first line of defense against insider abuse. An access control framework in the DCD is a kind of security policy that defines "authorized users" and "authorized use" of resources. A simple policy in the document control domain can be specified as (S, D, A), where S is the set of subjects (users), D is the set of documents and A is an access matrix. The entry A[s, d] gives the privileges that $s \in S$ has on $d \in D$. The drawback of such basic security policies are that they do not consider the context in which the access is being requested. The context includes among others, the sequence of requests already issued in the past, the documents that are currently open, etc. With reference to the malicious insider threat, such issues assume high importance in the DCD and are the primary means through which the access control framework is subverted. For example, if a user is allowed to copy a document for which he is not the owner, then he may copy the document contents and set different rights on the new document for which he would become the owner.

This paper extends standard policy models and applies them to the document control domain. The policies are designed with a conscious attempt to thwart the potential damage that a malicious insider could cause, or at least flag a warning if such damage is suspected. The goal is to specify and enforce security policies on digital documents. We go beyond the normal discretionary access control and role based access control, by considering the context at which the request is issued. This gives us the flexibility of performing context sensitive decisions, rather than standalone decisions. We also perform the runtime monitoring of possible information flow between documents and thus restrict access to a document if the sequence of requests leads to an insider threat (see Section 4.1). We have also developed a policy specification tool that gives the security analyst or the document creator a simple graphical interface for specifying policies on Microsoft Word documents. The final contribution is a secure viewer, which is a MS Word add-in that can enforce the policies at runtime.

The rest of the paper is organized as follows. Section 2 presents the related work in this area. In section 3, we enumerate the possible insider threats in the DCD. In section 4, we give a detailed description of the security policies and an example scenario where the policies can be applied. In section 5, we describe the implementation of our scheme on Microsoft Word documents. Finally, section 6 gives the conclusion and some future work.

2. Related Work

Originator-based access control [6] and propagated access control lists [7] are some of the access control mechanisms that try to protect dissemination of information by attaching an access control list with the data. It prevents dissemination of content by propagating the access control list of the object to other subjects and objects to which the content can flow. Though these techniques provide schemes for protecting information, they can become too restrictive as the information passes through several subjects.

Enforceable security policies [8] describe the notion of security automata that can be used to specify the types of security policies that can be enforced by monitoring the system execution. Such class of information policy is called EM for Execution Monitoring. It uses information that can be obtained just by observing the runtime events. More enforceable security policies [9] improve on the previous work by defining a superset of EM policies that includes the ability to insert and remove events from the input stream. The sequence of execution is important in these cases but in our case we care only about the set of actions that have occurred.

Multi-level security (MLS) policies such as Bell-LaPadula [10] provide rigid constraints which need to be relaxed during an emergency. For example, if a user is on leave, then any other user, who takes over his job, needs to be in the same security level. This highlights the need for adaptive policies. Mandatory access control policies such as Chinese-Wall model [11] prevent conflict of interest in a commercial setting. Objects are grouped into Conflict of Interest (COI) and Common Database (CD) groups. Each COI can have multiple CDs. The COI sets are mutually exclusive. A user can access resources based on his rights, but if an element in a particular CD is accessed, access to other CDs in the same COI is denied.

Role-based access control (RBAC) is used to specify policies for tasks, called roles, in the organization rather than individual users [12]. The users get assigned to roles in order to gain access to resources. The assignment of users to roles follows some constraints such as the "principle of separation of duty", role hierarchy, etc. RBAC provides an advantage over other access control models, since they reduce the inconsistencies that arise when policies have to be specified for subjects individually. Though some of the constraints can be leveraged into our framework, the role-based access control does not address any information flow restrictions. Temporal Role Based Access Control (TRBAC), an extension of RBAC, also specifies time constraints on when a role can be enabled or disabled [18]. For example, a constraint stating that a document cannot be opened beyond normal office hours can be specified in TRBAC. We go beyond temporal analysis and consider other factors such as machine's IP address.

HRU model [19], Typed Access Matrix model [20] and Take-Grant Protection model [21] are some models that study the safety property of a system. They address the transfer of right problem, that is whether a sequence of commands exist that add a right not originally present in the access control model. Our work is not targeted to the safety problem, but to prevent illegal information flows and abuse of privilege. In [22], the take-grant model has been extended to model methods, which can be used to share or steal information. Our information flow model is different from [22] and more specific to the DCD since we do not consider the human channel for information flow. This is because, an insider will have to cut and paste information from one document to another, rather than rewrite it in his own words, to maintain the authenticity.

In [13] advanced security policies are presented that define legitimate user privileges on resources, can adapt to system threat levels and detect abuse of privilege. Though this scheme can be made to work on a variety of resources, they are mainly targeted towards network resources, such as bandwidth, whereas our work is document centric. Various XML based specifications such as XACML [14], XACL [15] are available to specify adaptive security policies. We have used XACML to specify our policies in a standardized form.

3. Insider Threat Scenarios in the DCD

In this section, we describe a sample DCD and focus on the potential insider attacks possible in the DCD. Based on these, we will formulate the design considerations of the security policies necessary to overcome these threats. The DCD can be visualized as a corporate network with *n* users. Each user belongs to a group with a specific function, usually dictated by the nature of the organization. For instance a software company might have the groups: {CEO, Board Member, Administrator, Software Developer, Technical Writer, and Secretary}. During the course of his work, a user produces and consumes a variety of documents related to his work function. The DCD aims at protecting these documents from unwarranted usage and compromise. The CEO might work on a merger document whose compromise to the outside world could prove catastrophic to the organization. The existing solutions mentioned in Section 1 protect the document against the classic outsider. However, a malicious insider in the DCD starts off with several privileges. The CEO's secretary, for instance, could be leaking information to the outside world. It is quite possible for the secretary to forward the merger document she received for corrections to a rival company. Hence if there are no constraints on the privileges in the form of access control, then a malicious insider is capable of inflicting serious damage to the documents.

The possible insider threats are enumerated as follows:

a) An insider can read, copy, and print any document he has access to unless fine-grained access control is in place.
b) An insider can become the owner of the document by copying it to a new file and thus set new access control on the copied document.
c) An insider can forward a document to another user either inside or outside the organization.
d) A user can work late or early hours when the intrusion/misuse detection systems are not running.
e) He can copy the contents of a document into another document that is opened simultaneously.
f) An insider can remember the contents of a document, which he opened before, and then create a low priority document with the same contents.
g) An insider can take a dump of the document from the memory (such as the video buffer), and then print the document.
h) A malicious insider can tamper with the existing rights on the documents.

In this paper we focus on threats (a) – (e). We do not address the situation where a user can remember the contents of a document he has read previously and reproduce it at a later time (threat (f)). Also because we are targeting the insider threat at the application level, we cannot prevent a malicious user from taking a dump of the document from the main memory, where it is stored in plaintext format. This threat is different from copying the documents from the hard disk, because in the hard disk the documents are stored in an encrypted format. Finally, since in our framework the policies are stored separately from the document, as long as the policies are stored in a safe repository, threat (h) does not arise.

4. Policies Design Considerations

To design a policy specification to prevent the insider threat in a DCD, we need to consider both the context and information flow between requests. We take an approach where multiple policies are specified on the same resource. The policies differ in the context when they become applicable. For example, a policy might allow access to a document in the normal office hours but not during after-office hours. The current context is contained in the request for access (or is alternatively maintained on the policy server). The policy decision engine matches the request with the current policies in place and generates the new policy. We refer to such newly generated policies as *contextual* policies. Besides the current conditions the policies should also contain the obligations [14] or the provisional authorizations [16] that the subject should satisfy before access can be granted. The obligations are returned to the viewer at the client side as a part of response to the request and the viewer is expected to enforce them. An obligation might specify that a high priority document can be opened if and only if no other documents are currently open. Another obligation might specify that the user can print a document if and only if he has performed a biometric authentication.

The security policies can be bypassed if a user is able to copy the document content and become the owner of the document. In this way the user will be able to specify his own authorization rules on the document rather than using the rules specified by the actual owner of the document. Besides the originator control we also want to monitor the information flow occurring between different documents. This is because a sequence of authorized actions can cause information flow between documents leading to an information leak.

Consider an organization with a set of subjects S and a set of documents D that it wants to protect. Each subject $s \in S$ has some attributes that can be represented as a tuple $<s_1, s_2, \ldots, s_n>$. The attributes can be the name of the subject, the role the subject has in the organization, his classification in the organization, his credentials, his age and so on. Each document $d \in D$ also has attributes $<d_1, d_2, \ldots, d_m>$, representing features such as name of the document, the category of the document (e.g., top-secret, secret, unclassified), the type of document (e.g., system oriented, management oriented) and so on. The documents will have a set of actions A that can be performed on them. Without loss of generality we can assume that the set A is same for all documents (e.g., open, close, cut, copy, paste, print, send-to, etc.). In the application domain, this translates to the set of actions being the same for all similar document formats (PDF, DOC). We denote the set of policies as P. Each access control policy $p \in P$ is specified as a tuple $<d, \{Rule_1, Rule_2, \ldots\}>$. Here $d \in D$ specifies the target of the policy and rules specify the actions allowed/denied to subjects on this target. The rules also contain the conditions under which the rules apply. For example, a rule will allow a document to be accessed during normal office hours, but won't allow such access after office hours. Table 1 summarizes all the entities in our security policy.

Table 1: Description of entities used in security policy.

Entity	**Description**
S	Set of subjects in the organization
$<s_1, s_2, \ldots, s_n>$	Attributes of a subject
D	Set of documents in the organization
$<d_1, d_2, \ldots, d_m>$	Attributes of a document
A	Set of actions on a document
P	Set of policies
$p = <d, \{Rule_1, Rule_2, \ldots\}>$	A policy specified on document d
$Rule = <S, A, permit/deny, C, O>$	A rule giving authorization to a subject
C	Conditions under which the authorization is allowed or denied
O	Obligation that should be satisfied for the authorization to be allowed
$Req = <S, D, A, Situation{:}Var \rightarrow Values>$	A request.
$Situation{:}Var \rightarrow Values$	Mapping the context variables to their values
$Res = <permit/deny, Obligation{:}Var \rightarrow Values>$	A response
$Obligation{:}Var \rightarrow Values$	Mapping the obligation variables to their values

The critical parts of the policy are the conditions and the obligations. They are both static and dynamic in nature. The static conditions are those that are based on the known threats and remain the same whereas the dynamic conditions are evaluated based on the system conditions. While static conditions are evaluated based on known threats, this does not imply that the threats themselves are static. Known threats do change and when the threat knowledge base increases, the static conditions on the policies will be revised. The conditions are specified in propositional logic on the subject attributes, object attributes and other system attributes. The description of static conditions and obligations are widely available in access control literature [14] and are not discussed further. In the rest of the section we focus on the dynamic conditions based on the possible information flows between documents.

For all document actions performed, a request is generated which contains the user, document, action requested and other client side information. If some conditions are not specified in the request, the policy decision point should be capable of determining them from a third party. Among other variable, value pairs contained in the request, the most prominent of them is the set of currently open documents, *OD*.

The response contains the decision taken by the policy engine and the obligations that need to be further enforced. The static obligations are those that are already specified in the policy statement. The dynamic obligations are generated based on the current request and the rights of other users. The decision algorithm for generating these dynamic obligations is shown in Figure 2.

Next, we define the building blocks used to monitor information flow:

Definition 4.1: *Information flow* – We say there is an information flow between document d_i and d_j, represented as $d_i \rightarrow d_j$, if a user has both d_i and d_j opened at the same time and the user has write permissions on d_j.

Definition 4.2: *Information flow graph (IFG)* – An information flow graph is represented as $IFG=(V, E)$, where V is the set of documents currently open in the user's session and E is the set of edges, such that $(d_i, d_j) \in E$ if and only if $d_i, d_j \in V$ and $d_i \rightarrow d_j$.

Definition 4.3: *Privilege Graph* – A privilege graph is a directed graph and represented as $PG=(V, E)$, where V is the set of all documents a user has access to and E is the set of edges, such that $(d_i, d_j) \in E$ if and only if d_i, $d_j \in V$ and user has write permission on d_j.

The concept of a privilege graph here is different from the *privilege graph* defined in [17], because it is not a representation of vulnerabilities in the computing system but just a representation of rights a user has on the documents in the organization. In our approach, a node represents a document and an edge represents a write permission owned by the user. Whereas in [17], a node represents a user having some privileges in the organization and an arc represents a vulnerability by which a user can obtain greater privileges than what he is specifically granted.

The information flow graph IFG is constructed at runtime whereas the privilege graph PG is constructed statically (and updated only when a document is created or deleted). In both the graphs, IFG and PG, if a node has write permissions then it has incoming arcs from all the other existing nodes in the graph. Initially we start with a static pool of users and documents in the organization. Later in the section we relax this assumption and allow for additions/deletions of users and documents. Because of this assumption the size of the graphs remains the same.

```
Algorithm: Generate-Privilege-Graph
Input
    D: set of documents in the organization
    P: set of specified policies
Output
    PG: privilege graph of a user
Steps:

    $PG = (V, E)$, $V=\phi$ and $G=\phi$
    for all $d \in D$
      for all $p \in P$, s.t. $p=<d, *>$
        if $p$ allows read permission for the user
            add $v_d$ to $V$ /* $v_d$ : node for doc $d$ */
        if $p$ allows write permission for the user
            add a tag to $v_d$
        for all $v \in V$
            if $v_d$ has a tag add edge $e=(v, v_d)$ to $E$
            if ($v$ has a tag) and ($v \neq v_d$) add edge
                $e=(v_d, v)$ to $E$
```

Figure 1: Algorithm for computing privilege graphs.

Once the system is deployed, its first task is to build the privilege graph of all users in the organization. It is a one time approach and needs to be modified only when a new document gets created or an existing one is deleted. The *PGs* are generated based on the policies specified on the documents. Algorithm *Generate-Privilege-Graph*, in Figure 1, is used to generate the graphs for all the users in the organizations.

Information flow graph gets created and deleted with every request for opening and closing a document. Whenever a request is received the *Generate-Information-Flow-Graph* is called which is the same as *Generate-Privilege-Graph*, except it uses the set *OD*, the set of documents currently open in the user's workspace (*OD* is contained in the request), rather than the complete set of documents.

When a request is received and allowed by the policies, the information flow graph and privilege graphs are used to decide whether the right should be granted. The algorithm in Figure 2 outlines the steps used in making the decision.

Decision Algorithm:

a) Consider the IFG of the current request and the PG of another user.
b) If allowing the request in IFG leads to a violation in the PG then either the request is denied or some other obligations are added.
c) Steps (a) and (b) are repeated for all the remaining users other than the one who made the request.

Figure 2: Decision Algorithm for generating dynamic obligations.

The primary objective of the decision algorithm is to prevent illegal information flow from one document to the other. Based on the definition of information flow (see Definition 4.1), in order to prevent illegal information flows we have to prohibit writing to a document when another document is open. The new restrictions are given in the form of obligations. For example, if document *d3* is being opened when another document *d1* is already open, the obligation might state that the write permission on *d1*, previously allowed, will be disabled as long as *d3* is open. The restriction will be enforced by the client side viewer by disabling the edit options. When *d3* is closed edit permissions on *d1* will be enabled again. This dynamic algorithm is run each time a request comes from a user for a document. We hypothesize that the number of documents any user works on in a given time is not high and hence application of the dynamic algorithm will not cause any noticeable delay in an intranet.

In order to compute the new set of obligations, two new graphs $K_{|D|}$ and $\neg PG$ are computed. $K_{|D|}$ is the complete graph on the set of documents with all reads and writes enabled and $\neg PG$ is the complement of PG (difference between $K_{|D|}$ and PG). $K_{|D|}$ remains the same as long as no document is created or deleted. Similarly, $\neg PG$ remains the same as long as PG doesn't change. Examples of PG and $\neg PG$ are given in Figure 2. Step (b) of the decision algorithm first computes U, the union of IFG_i and PG_j (union is taken over the set of edges). The subscripts *i* and *j* represents user *i* and *j*, where $i \neq j$. It then finds the common edges E, between U and $\neg PG_i$. E contains the illegal information flows, which get created if the new document is opened. For each $(v_i, v_j) \in E$, the obligation "no edit on v_i" is added, if the obligation is not already present. When the document is closed, two sets of obligations are computed. The first set is the one without the document being closed and the second set is the one with the document being closed. All the obligations present in the first set but absent in the second is now allowed. In section 4.1, we give an example scenario of the framework in action.

Till now we had assumed that no new documents get created or old documents get deleted from the document pool. Here we relax those restrictions. Whenever a new document gets created, a request is sent to the policy server with the list of currently open documents, *OD*. The rights that can be granted on the new document will be a subset of all the allowed rights on the currently opened documents. Such a restriction prevents the current user from copying a document and setting liberal rights on the copy (threat (b) in section 3). Also when a new document is created the privilege graphs of all the users are recomputed. When a document is deleted, the static access rights are checked and if allowed then the document is deleted and the privilege graphs of all the users are updated.

4.1. Example Scenario

In this section, we consider a set of two users and three documents to illustrate our framework. The users are identified as *s1* and *s2* and the documents are identified as *d1*, *d2* and *d3*. Table 2 gives the access control matrix.

Table 2: Access control matrix

	s1	s2
d1	R,W	R
d2	R,W	φ
d3	φ	R,W

The following attack scenario is considered:

1. *s1* opens *d2* to read the contents.
2. *s1* opens *d1* to modify its contents.
3. *s1* doesn't have write permission on *d3*.
4. *s1* writes onto *d1*.
5. *s2* opens *d3*.
6. *s2* opens *d1*.
7. *s2* can now copy the contents of *d1* to *d3*.

Effect: *s1* is able to leak the contents of *d2* into *d3* with *s2*'s help.

The privilege graphs of *s1* and *s2* are shown in Figure 3. When *s1* opens *d1* the IFG_{s1} only contains *d1* and nothing is done in this case because there is no information flow possible (see definition 4.1). When *s1* opens *d2* the IFG of *s1* is the same as PG_{s1}. The next step is to find $U = IFG_{s1} \bigcup PG_{s2}$ (see Figure 3). Finally, the intersection of U and $\neg PG_{s1}$ is computed (represented as I in Figure 4). All the edges in I represent the possible ways of information leak. As (d_1, d_3) is an edge in I, we add an obligation that prevents

IEEE COMPUTER SOCIETY

write access on *d1* when any other document is open. Read permission on *d1* is still allowed and both read and write on *d2* are allowed. Now, since *s1* cannot write to *d1* he cannot leak the information of *d2* into *d3*. This framework does not prevent information flow through the human channel. Also, we are attempting to prevent information flow *given* certain policy specifications.

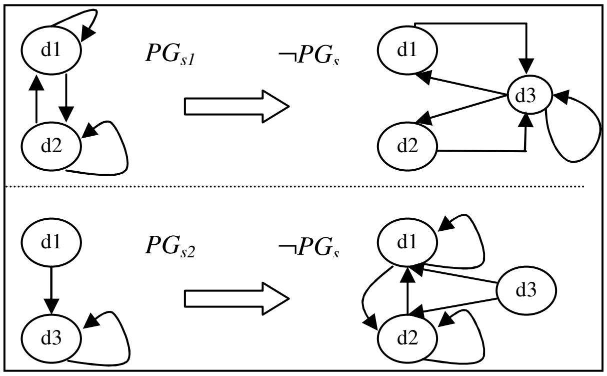


Figure 3: Privilege graphs and their complements.

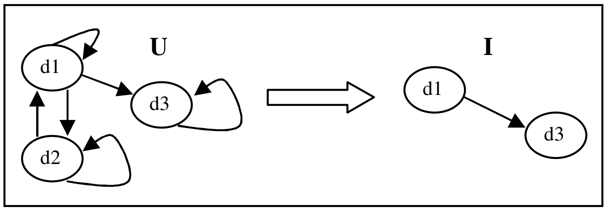


Figure 4: Detecting malicious information flows.

5. Implementation

In this section, we describe our framework to set and enforce custom policies to prevent insider abuse. The DCD scenario as described in Section 1 is placed in perspective in this section.

5.1. Initial Setup

All users in the DCD work on a set of machines <M_1,M_2,….,M_n>. These machines run Windows XP Professional. The targeted editing software is Microsoft™ Word 2003 – henceforth referred to as Word™ in this paper. We have developed an Add-in to Word™ that has certain features added on to it. One of those features is to enforce our custom policies. The initial setup of the Secure Viewer creates an account for the user at the policy server. Specifically, the steps followed during the initial setup are:

1. Install the Secure Viewer Add-in Component (Word™ 2003 must be pre-installed on the machine with the .NET framework version 1.1)
2. The Secure Viewer is configured to load when Word™ starts up.
3. Start up Word™ the first time after installing the Secure Viewer. It'll contact the Policy Server which will create an account for the user.

The final version of the Secure Viewer is expected to have support for importing certificates from reliable sources if present (Active Directory, LDAP Server, etc.) or generating one for the user on registration.

Our access control architecture is shown in Figure 5.

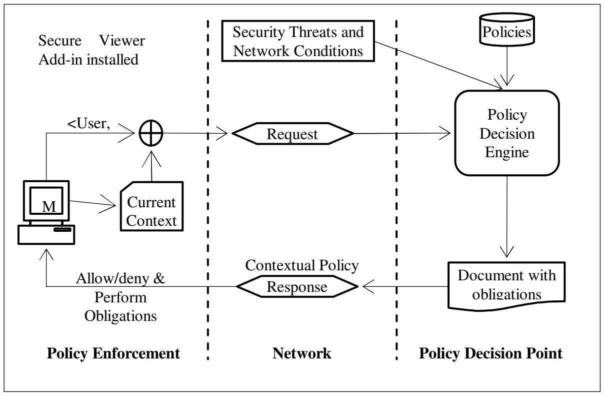


Figure 5: The Policy Enforcement Architecture

5.2. Policy Enforcement Point (PEP)

The policy enforcement point (PEP) is where the user actually views the document on a machine. In our architecture, we allow for the user to be an administrator on the machine he is working on. We find that this is typically the situation with most organizations that use Windows XP. Users are granted local administrator rights along with a firewall on the hub where the machines are connected. The other configuration most commonly found was a base Linux distribution with virtualization software such as Vmware™ installed. Windows™ XP was installed on Vmware™ and users were granted local administrator privileges on the virtual machine. Hence the Secure Viewer Add-in is designed to render the document and enforce the policies on the documents irrespective of the rights the user has on the machine. This also implies that the user could download any document onto his laptop and view it at home. The policies on the document would still be enforced. In fact he would be able to read a document offline only if:

1. The Secure Viewer Add-in is installed on the local machine AND
2. The policy on the document allows for offline viewing.

5.2.1. Secure Viewer

The Secure Viewer is a COM Add-in to Microsoft™ Word 2003. Written in C#, it's based on the .NET framework version 1.1.
Its function is to:

1. Monitor all actions performed on Microsoft Word – this is done through setting hooks to Word™ actions through delegates in C#.
2. Read in an encrypted document, decrypt it and display it to the user on the fly – this is done through custom encryption and decryption methodologies using standard algorithms. By custom methodologies, we mean to say that the Office Information Rights Management (IRM) [2] backbone is currently not used, though it is planned in the final version. The service provider for the crypto system is provided by the office API's. Also, the document is encrypted whenever it is on the disk. Hence lack of file level control is irrelevant since the document cannot be viewed without the Secure Viewer.
3. Read in the contextual policy sent in by the policy server and enforce it on the user's existing context.
4. Set user-defined policies on newly created documents and encrypt them – these user-defined policies are tuned to prevent the insider threat.
5. Other anomaly detection schemes for addressing the malicious insider threat are embedded which are not directly related to policies.

All these functions are performed unobtrusively in the background. The only addition to Microsoft Word on the interface part is a Menu-Bar button shown in Figure 6. As the name suggests, it's created to set policies on the document.

5.2.2. Policy Assignment Tool

The policy assignment tool is a stand alone application that assigns policies on Word documents. It is also integrated with the Secure Viewer so that the custom policies may be set on documents when they are created by the owner and uploaded to the file server. The primary utility of the policy assignment tool as a stand-alone application is during migration scenarios. An organization that already has an unprotected document infrastructure can use this tool to set policies on multiple documents in order to set the document policies and classifications. Permissions are set on documents along with obligations (that are organization-specific).

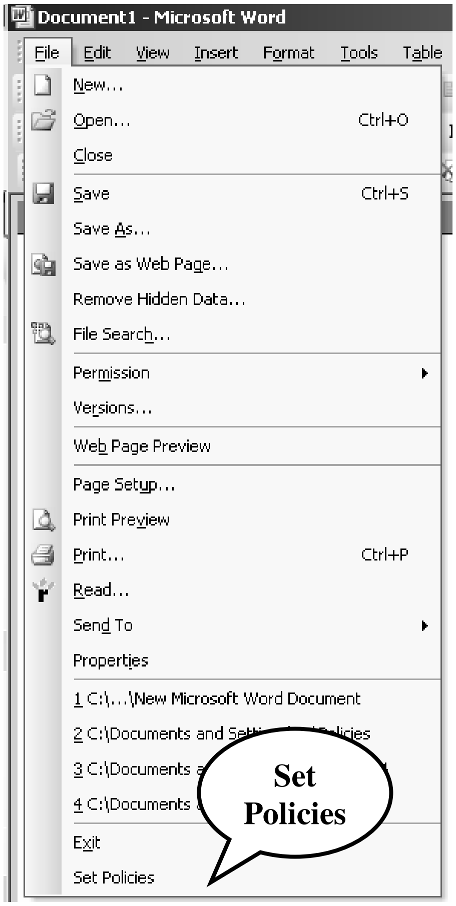
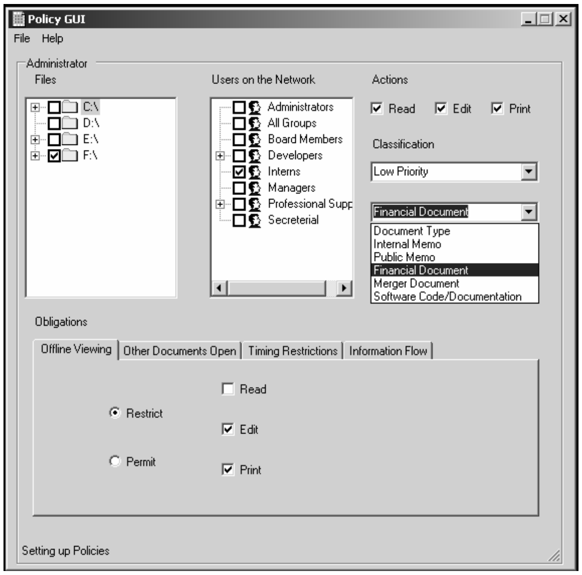


Figure 6: Secure Viewer Interface to Word™

Figures 7a and 7b show screenshots of the policy tool.



Files
□ C:\
□ D:\
□ E:\
☑ F:\

Users on the Network
□ Administrators
□ All Groups
□ Board Members
□ Developers
☑ Interns
□ Managers
□ Professional Supp
□ Secreterial

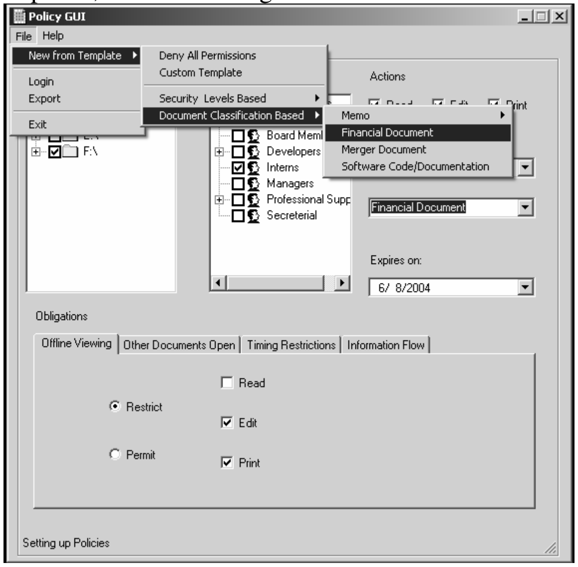
Actions
☑ Read ☑ Edit ☑ Print

Classification
Low Priority
Financial Document
Document Type
Internal Memo
Public Memo
Financial Document
Merger Document
Software Code/Documentation

● Restrict
○ Permit
□ Read
☑ Edit
☑ Print

Figure 7a: Policy Tool

The administrator can choose entire directories or individual files and the groups of users for whom the permissions are applicable. The document type and its security levels are set along with the obligations shown in the lower half of figure 7.a. An optional expiry date can also be set. Since the policies are expressed in XACML for interoperability, it is also possible for the organization to apply existing templates to the documents, or alternatively, apply their own custom templates, as shown in Figure 7.b.



Financial Document

Expires on: 6/ 8/2004

● Restrict
○ Permit

□ Read
☑ Edit
☑ Print

Figure 7b: Policy Tool

5.2.3. Mechanisms

All documents in our architecture are stored at the policy server to begin with. They are encrypted with document specific keys at the storage point. These encrypted documents cannot be viewed with the normal out-of-the-box installation of Microsoft™ Word. They require the Secure Viewer Add-in component to be installed on the machines. This is the first step taken to ensure that the documents cannot be read off-hand. To open an existing document, the user is presented with a custom interface that actually shows the documents residing on the remote file/policy server. Once a document is selected, the request from the user is created. This request encapsulates the user authentication token and the document identifier. The Secure Viewer component also packages the current context of the machine. This includes the documents that are currently open, the authentication type of the user, etc. This request is sent encrypted over the network to the Policy Server.

5.3. Network

All communications over the network go through the Secure Socket Layer. The user's certificate that was created during the setup procedure is used for generating session specific keys for communication. The certificate generation process during setup is also secured when passing through the network. Although this plays an important role in the overall security of the DCD, this is not directly related to policy creation or enforcement and the description is hence omitted.

5.4. Policy Decision Point (PDP)

The PDP is the place where the decision on the incoming request is made and the document is sent out with the relevant contextual policy. The PDP is a logical representation of the policy server and the file server in the organization. The decision algorithm in Section 4 is applied at this point. The incoming request is disassembled. At the first stage, a trivial policy check is made if the user has the requisite permissions for the actions requested on the document. Once this check has passed, the IFG is generated from the Context of the incoming request. The IFG and the PG are compared with some additional input from the existing Security Threats. The existing threats are an input that comes from the Intrusion detection system (not shown) in our architecture. The output of this comparison leads to the decision (to send or not to send the document) along with the obligations that the user must follow in order to access the documents. This is referred to as the contextual policy. The contextual policy is packaged as the response and is sent to the PEP where the Secure Viewer receives it and enforces the new contextual policy.

6. Conclusion and Future Work

In this paper we have presented a security policy that can be used to prevent insider abuse on digital documents. The security policy takes into account the system context when it is applied and the possible information flows between documents. We have also described a framework being developed for demonstrating the application of such security policies. The framework can be used by organizations to protect their critical documents. Initial versions of certain parts of the framework are available for download at http://www.cse.buffalo.edu/DRM/prototype.

In the future, several insider attack scenarios will be created based on the known insider threats. These scenarios will be used to test the framework and help in specifying optimal security policies. The framework can also be deployed in a real test environment and used to capture the malicious insider behavior in an organization. Finally, the policy server will be made to work in tandem with an anomaly detector to capture novel attacks.

Though the security policies have been implemented for MS Word, it can be incorporated into other document types also. MS Word was chosen for its widespread use. In the future we will develop a secure viewer for other document formats also.

7. References

[1] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Communications of the ACM* 17(7), July 1974.

[2] http://www.microsoft.com/office/editions/prodinfo/technologies/irm.mspx

[3] http://www.authentica.com/products/document_protection.asp

[4] E. E. Schultz. A Framework for Understanding and Predicting Insider Attacks. *Computers and Security* 21(6), pages 526-531, 2002.

[5] C. Bateman et al. A proposal for a Thread on the Insider Threat Problem Proposal. System Dynamics Modeling for Information Security: *A Group Modeling Workshop*, January, 2004.

[6] R. Graubert. On the Need for a Third Form of Access Control. In *Proceedings of the 12th National Computer Security Conference*, pages 296-304, October, 1989.

[7] D. Wichers et al. PACLS: An Access Control List Approach to Anti-Viral Security. In *Proceedings of the 13th National Computer Security Conference*, pages 340-349, October, 1990.

[8] F. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security* 3(1), February, 2000.

[9] L. Bauer, J. Ligatti, and D. Walker. *More enforceable security policies*. In *Proceedings of the Workshop on Foundations of Computer Security (FCS'02)*, Copenhagen, Denmark, July 2002.

[10] D. E. Bell and L. J. La Padula. Secure computer systems: Mathematical foundations and model. *Technical Report M74-244*, The MITRE Corporation, May 1973.

[11] D. F. C. Brewer and M. J. Nash. The Chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206-214, Oakland, CA, May 1989.

[12] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, Role Based Access Control, *Artech House*, 2003.

[13] T. Ryutov and C. Neuman. The Specification and Enforcement of Advanced Security Policies. *Policy* 2002.

[14] S. Godik, T. Moses, et al. eXtensible Access Control Markup Language (XACML) Version 1.0. OASIS *standard,* February, 2003.

[15] M. Kudo and S. Hada. XML Document Security based on Provisional Authorization. In *7th ACM Conference on Computer and Communication Security (CCS 2000)*, November, 2000.

[16] S. Jajodia, M. Kudo and V. S. Subramanian. Provisional Authorizations, In Gosh, A., editor, *E-Commerce Security and Privacy*, pages 133-159, 2001, Kluwer Academic Press, Boston.

[17] M. Dacier and Y. Deswarte. The Privilege Graph: An Extension to the Typed Access Matrix Model. In *European Symposium in Computer Security*, 1994.

[18] E. Bertino, P. A. Bonatti and E. Ferrari. TRBAC: A temporal role-based access control. *ACM Transactions on Information and System Security*, 4(3), pages 191-223, August, 2001.

[19] M. Harrison, W. Ruzzo and J. Ullman. Protection in operating systems. *CACM*, pages 461-471, 19(8), 1976.

[20] R. Sandhu. The typed access matrix model. In *IEEE Symposium on Security and Privacy*, 1992.

[21] A. Jones, R. Lipton, and L. Snyder. A linear time algorithm for deciding security. In *17th Annual Symposium on the Foundations of Computer Science,* pages 33-41, October, 1976.

[22] M. Bishop. Theft of information in the take-grant protection model. *Journal of Computer Security*, 3(4), pages 283-309, 1994/1995.