APEX: Automated Policy Enforcement eXchange

Steven J. Simske, Helen Balinsky

HP Laboratories
HPL-2010-134

Keyword(s):

Policy, Text Analysis, Policy Server, Policy Editor, Document Systems, Document System Components, Security

Abstract:

The changing nature of document workflows, document privacy and document security merit a new approach to the enforcement of policy. We propose the use of automated means for enforcing policy, which provides advantages for compliance and auditing, adaptability to changes in policy, and compatibility with a cloud-based exchange. We describe the Automated Policy Enforcement eXchange (APEX) software system, which consists of: (1) a policy editor, (2) a policy server, (3) a local daemon on every PC/laptop to maintain local secure up-to-date storage and policy, and (4) local (policy-enforcing) wrappers to capture document-handling user actions such as document export, e-mail, print, edit and save. During the performance of relevant incremental change, or other user-elicited action, on a composite document, the document and its metadata are scanned for salient policy eliciting terms (PETs). The document is then partitioned based on relevant policies and the security policy for each part is determined. If the document contains no PETs, then the user-initiated actions are allowed; otherwise, alternative actions are suggested, including: (a) encryption, (b) redirecting to a secure printer and requiring authorization (e.g. PIN) for printing, and (c) disallowing printing until specific sensitive data is removed.

External Posting Date: October 6, 2010 [Fulltext] Approved for External Publication
Internal Posting Date: October 6, 2010 [Fulltext]
To be presented at ACM DocEng 2010. Manchester UK.

APEX: Automated Policy Enforcement eXchange

Steven J. Simske
Hewlett-Packard Labs
3404 E. Harmony Rd., MS 36
Fort Collins CO 80528 USA
Steven.Simske@hp.com

Helen Balinsky
Hewlett-Packard Labs
Long Down Avenue, Stoke Gifford
Bristol BS34 8QZ UK
Helen.Balinsky@hp.com

ABSTRACT

The changing nature of document workflows, document privacy and document security merit a new approach to the enforcement of policy. We propose the use of automated means for enforcing policy, which provides advantages for compliance and auditing, adaptability to changes in policy, and compatibility with a cloud-based exchange. We describe the Automated Policy Enforcement eXchange (APEX) software system, which consists of: (1) a policy editor, (2) a policy server, (3) a local daemon on every PC/laptop to maintain local secure up-to-date storage and policy, and (4) local (policy-enforcing) wrappers to capture document-handling user actions such as document export, e-mail, print, edit and save. During the performance of relevant incremental change, or other user-elicited action, on a composite document, the document and its metadata are scanned for salient policy eliciting terms (PETs). The document is then partitioned based on relevant policies and the security policy for each part is determined. If the document contains no PETs, then the user-initiated actions are allowed; otherwise, alternative actions are suggested, including: (a) encryption, (b) redirecting to a secure printer and requiring authorization (e.g. PIN) for printing, and (c) disallowing printing until specific sensitive data is removed.

Categories and Subject Descriptors
I.7.1 Document and Text Editing; I.7.4 Electronic Publishing

General Terms
Algorithms, Design, Security

Keywords
Policy, Text Analysis, Policy Server, Policy Editor, Document Systems, Document System Components, Security

1. INTRODUCTION

1.1 Document Security Problems

Sensitive information routinely escapes governments and companies in the form of digital or printed documents, for example [1,2]. Document fraud, intentional or unintentional, includes reading or removal of printed documents by other members of a company (even visitors), unauthorized emailing of the documents, and surreptitious, unauthorized alteration of documents. Private, confidential and otherwise sensitive documents should not be printed, routed, stored unencrypted, etc., outside of company, government, or other organization.

Today documents are becoming more complex, combining multiple parts and formats together: e.g., xml-files, images, video clips, Microsoft Word and PowerPoint, and PDF files. These composite documents are created and accessed by different workflow participants with various access rights, which requires the corresponding parts to be protected accordingly. Changing company rules, emerging security threats, new privacy requirements, and government legislation result in new policies for the lifecycle of a document. Thus, document workflows comprise a composite of different parts and different formats, each with potentially changing security and privacy levels.

Web-based documents such as Wikis, blogs and on-line forms have been around for some time now, and there is a consistent move to integrate dynamic document attributes into browsers and/or the cloud [3-5]. Because the confidentiality classification of these documents may change over time, the dynamic determination of policy may be beneficial.

The paper is organized as follows: we start with the problem statement (Sect 1.1), followed by existing solutions (Sect 1.2). In Section 2, we provide our proposed solution. In Section 3, details of the current implementation are discussed. Conclusions and future work are provided in Section 4.

1.2 Problem Statement

We considered the following security threats and threat responses in our design of an Automatic Policy Enforcement eXchange (APEX) system:

(1) Alleviating the threat of the "weakest" link security risk, wherein the individual least familiar with policy may inadvertently perform actions contradicting the required security policies - for example, printing at a multi-user printer, emailing outside the enterprise, etc.
(2) Alleviating the threat of the insidious insiders, using a local daemon to log all confidential, private, secure, etc. documents, if appropriate and/or allowed for the jurisdiction.
(3) Allowing policy enforcement actions to change independently of the interface software to monitor the documents during their lifecycle. Policy is enforced throughout the document lifecycle, so that documents are compliant by design.

1.3 Prior Art

Many current document systems and their security components are based on the application of static workflow policies. Access control is exclusively linked to the document metadata, which is either manually assigned by the document creator/owner (e.g. discretionary) or automatically determined by his/her role (e.g. mandatory, role based access control). Thus, access control and other security policies are determined when the document is created and not changed thereafter (e.g., HP Exstream); or, even if allowed to change thereafter (e.g. Adobe LiveCycle), a single policy is applied to the document. For these system, the document itself is considered the atomic element for the application of security policies. Currently, there is a rapid adoption of workflows comprised of distributed, multi-participant and composite documents – comprised of multiple traditional documents – by businesses and other large enterprises. These create are new concerns for document privacy, access control and other security topics. In an effort to staunch security attacks, a new model for enforcing the correct policies is proposed.

2. OUR SOLUTION

The sensitivity level of a document or one of its parts can dramatically change as the result of:

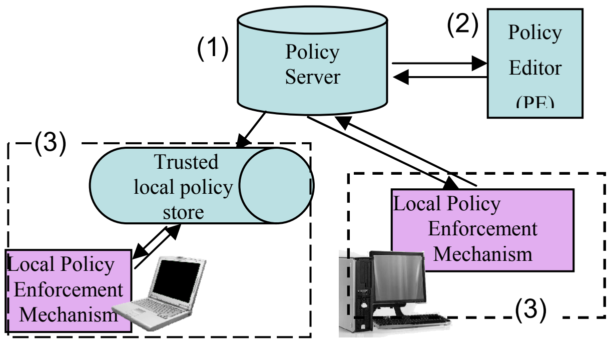
A. A part being modified by user or a process, e.g. a credit card number and customer data were added or removed;
B. A new government legislation, industry standard, company policy being introduced, e.g. the Freedom of Information Act [6], HIPAA [7], etc.

To adequately address this dynamicity, the sensitivity level of a document, and/or its parts needs to be determined when the document is "acted upon": e.g., is about to be saved into non-secure location, e-mailed, printed, etc. Any action by a user or by a process which can potentially expose sensitive information needs to be intercepted (captured). Not only document metadata, but also its actual content, must be evaluated for policy eliciting tags (PETs), which are Boolean expressions governing allowance of terms. Each executing unit (desktop, laptop, etc.) must have secure access to the up-to-date policy storage.

Steps:

1. Automatically intercept user/process actions that are potential data exposure risks.
2. Deep scan of document parts and metadata for PETs (including patterns of PETs) associated with the requested action to determine each part sensitivity level.
3. Policy-defined action for each part based on its current sensitivity level.

2.1 Apex Architecture



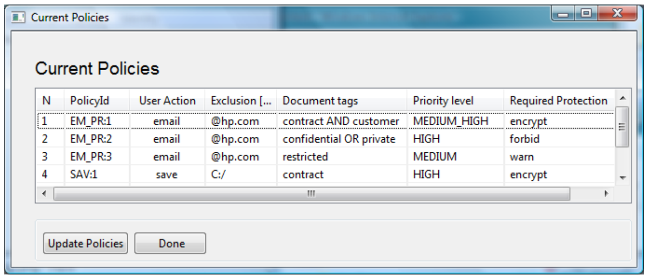
(1)
Policy Server
(2)
Policy Editor (PE)
(3)
Trusted local policy store
Local Policy Enforcement Mechanism
Local Policy Enforcement Mechanism
(3)

Figure 1: APEX architecture high level overview

The APEX architecture is based on the following currently implemented elements (Figure 1):

(1) Central Authoritative Policy Server - the APEX Policy Server (PS) to maintain persistent access to current policies.
(2) A Policy Editor (PE) to manage the policies on the PS in a single location (Figure 2).
(3) Local Policy Enforcement Point on every PC/laptop comprising a Local Daemon (LD) to maintain local secure up-to-date storage, and local wrapper applications to enforce policies.

There are two distinct deployment modes for APEX: as a part of a document handling application and as a "corporate desktop" advisory or mandatory safeguarding agent.



Current Policies

N	PolicyId	User Action	Exclusion [...	Document tags	Priority level	Required Protection
1	EM_PR:1	email	@hp.com	contract AND customer	MEDIUM_HIGH	encrypt
2	EM_PR:2	email	@hp.com	confidential OR private	HIGH	forbid
3	EM_PR:3	email	@hp.com	restricted	MEDIUM	warn
4	SAV:1	save	C:/	contract	HIGH	encrypt

Figure 2: Simple Policy Editor, where each policy is represented by one SQL-entry with 3 main components: document handling operation, policy eliciting terms, and a protection mechanism.

3. APEX IMPLEMENTATION

3.1 Local Policy Enforcement Point

APEX Document Scanning (Parsing)

The APEX prototype searches the salient fields in any combination of Microsoft PowerPoint, Word, Excel, and Adobe PDF documents (and is potentially extensible to any other common formats) to determine if PETs are present. This has advantages over previously-reported XML-based methods [8] which include the fact that the composite document parts are in their native form. For example, PETs may include "Private", "Confidential" and "Secure", or any employer-defined tags provided with their corresponding levels of security, e.g.: Confidentiel (French), high security; Privé (French), medium security, etc. As regular expressions, PETs have wildcards, error tolerance, and AND, OR, NOT operations, etc. We implemented deep content-based search for other potentially sensitive data, e.g. "credit card number", "social security number", "customer data", etc. This scan can be assigned to individual parts of any partitioning of the document. Figure 3, left, shows one such partitioning: by footer, header and body of the document.

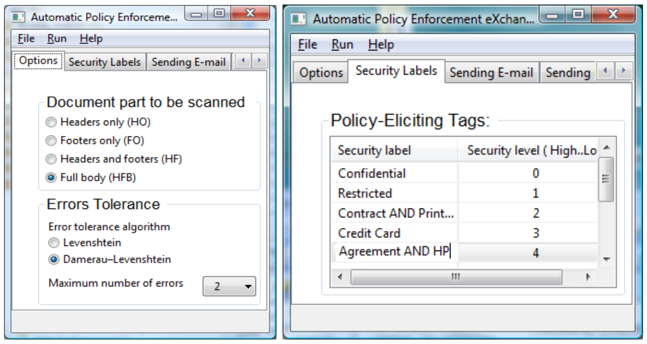
Statistical language processing (SLP) techniques are used to provide "fuzzy" matching for these terms in case of misspelling or language variants (e.g. plural forms of a PET defined in the singular, etc.). We implemented both the Levenshtein and the Damerau–Levenshtein distances [9] with variable tolerance to error (Figure 3, left). The system is currently integrated into the APEX prototype, and can be just as readily integrated with the specific application software monitored (as a "corporate desktop") or run as a Local Daemon, or LD.

Multiple policies may be applied to a document part simultaneously, obviating the need to create a new policy update aggregating the two. As potentially multiple policies are applied in response to any specific operation, they are classified by the operation type to simplify the policy enforcement interactions best overall system performance. Policy assignment is dynamic in APEX as a consequence of the deep document scan for policy eliciting terms. Document metadata and internal sections (header, footer, body, etc.) are probed for salient fields as described above.

The composite document is logically partitioned based on relevant policies and the security level of each part is determined.

Parts with like security are combined logically to reduce security algorithmic overhead. If a document contains no PETs, then the user required actions are allowed. If, however, PETs which require a change in enforced security approach are identified, then alternative actions are enforced as prescribed by the identified policy. For example, the mandatory encryption can be applied or the job can be redirected to a secure printer, requiring the user to provide a PIN for job retrieval.

Documents are automatically scanned when any action that can potentially reveal the information is detected.


Automatic Policy Enforceme...
File Run Help
Options Security Labels Sending E-mail

Document part to be scanned
- ○ Headers only (HO)
- ○ Footers only (FO)
- ○ Headers and footers (HF)
- ● Full body (HFB)

Errors Tolerance

Error tolerance algorithm
- ○ Levenshtein
- ● Damerau–Levenshtein

Maximum number of errors: 2

Automatic Policy Enforcement eXchan...
File Run Help
Options Security Labels Sending E-mail Sending

Policy-Eliciting Tags:

Security label	Security level (High..Lo
Confidential	0
Restricted	1
Contract AND Print...	2
Credit Card	3
Agreement AND HP	4

Figure 3 APEX prototype: [left] Selection options for the search for policy-eliciting terms(PETs) including selection of error tolerance algorithm and the number of errors tolerable; [right] security levels associated with specific PETs, "Confidential" here has the highest security (level 0)

Action Capturing Wrappers:

Local wrappers capture document-handling user actions such as export, e-mail, and print, in addition to read/write permission with document part sensitivity levels and access controls.

User actions are captured using system call interception with code injection. On Linux, this is accomplished using Linux kernel hooks and/or the Linux login shell (e.g. systrace). More generally, Aspect Oriented Programming (AOP) can be deployed as the process around the system call interpretations. AspectJ, for example, can be used to capture and act upon all requests for save, print, import and send.

Local Daemon (LD)

The LD is a secure automatic service (CORBA, RMI, etc) run in background of every unit. Owned by the root/admin, the LD is persistent: if accidentally or intentionally killed, a new daemon automatically appears. The properties of the LD are:

1. Accepts the https PUT request for new policy definitions only from the PS, or automatically polls it.
2. Updates the secure local storage accordingly.
3. Generates a transaction ID and sent it as the confirmation to the PUT request.

The transaction ID is {policy_update, timestamp} signed by the LD signature key. Only this signature and the timestamp are sent back.

The LD provides real-time logging, so that inadvertent or malicious early session terminations are still auditable (stored and suitable for data mining). Role management can be tied to existing access control management/identity systems (PIN, Smart Card, static biometrics, username and password, etc.).

Communication Channel PS and LD

Secure communication between the PS and the LD comprises: (1) mutual authentication based on known certificates; (2) an encrypted communication channel (SSL/TLS, https); and (3) preliminary registration/subscription, which can be automated, during or after LD installation.

The communication channel between the local enforcement point (LD) and the PS is very secure, as compromising it results in every corporate unit (PC/Laptop/iPhone/…) being exposed to policy spoofing. Every unit handling sensitive data, must run an LD and subscribe to the PS by exchanging certificates.

3.2 Policy Server (PS)

The PS is an SSL/TLS supporting server that provides two secure services:

(1) Get the last policy update (PU) reference number, where policy updates are numbered sequentially; and

(2) Get a PU by its given unique number.

In the polling mode, each client periodically queries for the latest policy update PU_X, compares it with its own latest policy update number PU_C, and downloads, sequentially, all intervening policy updates *pu*, whose IDs are $PU_C < pu \leq PU_X$ and $PU_C > PU_X$ is an invalid state and system fault.

In the active policy distribution mode, the PS deploys messaging to publish or subscribe to a service (e.g. in our system using JavaMS) to inform its subscribers that the new policy is placed on-line. This message is securely communicated and the transaction ID is communicated back. Off-line systems communicate and retrieve the latest policy when they are updated.

For each downloaded policy update, the client generates the transaction ID by signing the downloaded policy update together with the client name/ID and a secure timestamp. Only this signature is communicated to the PS. The server verifies the timestamp and the unit name/ID, then validates the signature for the communicated policy update using the (known) client signature verification key. If everything verifies, the server marks that the client received the policy and stores this information for any subsequent audit. All transactions are archived by the PS: they provide the non-repudiation proof for timely policy delivery.

A central high reliability server or distributed cluster (e.g. a JBOSS cluster) can be used to ensure reliability and persistence of policy distribution across multiple machines [10]. At least one of the policy servers should be responsive to allow a policy decision to be made. As an example, even over a virtual private network (VPN) connection, at least one gateway connection should respond for a connection to be established.

Policy: Update and Policy Distribution Tracking

A policy comprises a unique ID, a policy condition, a "*required_action*" and a response, as shown by simple example in Figure 2. The unique policy ID, p, is usually sequential in time (since there is no need to obfuscate the ID). The policy also encodes a policy condition, which can include new PETs, new security requirements (e.g. encryption standards, certificates, or hashing algorithms), new data retention rules, new auditing rules, etc. The policy also contains a *required_action* specification which determines, for example, if the policy needs to be applied retroactively to existing documents (possibly even ones associated with completed workflows, which may need to be updated for compliance on encryption, archiving, etc.). The *required_action* may also specify the terms of the response. The response is the acknowledgement of policy receipt. It may also include user roles to which the given policy is applied. The policies can be different for different roles.

A Policy Update (PU) for adding a new policy contains this new policy P_X (with new policy id) in its body. The PU is one of 3 types: adding a new policy, updating an existing policy, or deprecating a policy. The type of PU is optional: as it can be automatically inferred by the system. It contains the following security-related elements:

1. [mandatory] The corresponding sequential ID number assigned to each PU; for example, its primary key in the PU database.
2. [mandatory] PS signature of this PU. This will reassure each client that this is a valid/legitimate update.
3. [optional] The type of update: "new policy", "update" or "cancellation". This optional field is a convenient way to simplify software logic on the client side.
4. [optional/mandatory] Timestamp T_1

As with software updates, the order of application for policy updates is extremely important, as policy update k could be an update or cancellation of an existing policy that by itself was introduced by some previous update PU: $j < k$.

When a new policy update is received, a client verifies its signature using the known PS signature verification key, the policy update ID, and the current timestamp. If all tests on these data pass, then the new policy is placed into the local client's policy store. The client then generates a transaction ID to confirm to the PS that the policy update has been accepted. The client automatically signs the received PU together with the current timestamp. This timestamp T_2 and the signature S are communicated to the PS. The PS checks that the received timestamp T_2 is not earlier than T_1 and is within acceptable limits: $0 \leq T_2 - T_1 \leq \in$. Knowing the communicated PU, T_2, and the client's verification key (known from client subscription to the service), the PS validates the received signature. If it validates, the PS accepts that a given PU was successfully delivered. This transaction ID is stored for further audit or to respond to a situation in which the client denies being notified of policy. This security non-repudiation is very important for both the PS and a client when a policy breach is investigated.

A Policy Update replay attack cannot be accomplished. Since all policy updates have a unique ID, only PU's with ID's different (typically incremental) than the current PU ID may be applied.

Policy Distribution and Off-Line Work

The deployment availability of a unit that is off-line or otherwise without active connection to the PS depends on the deployment scenario and the sensitivity of handling data. An important factor is the tolerable time delay, τ, between an update issuance and its mandatory applicability. For a period of time $t \leq \tau$, since the last policy update, the unit is allowed to function as usual and/or be fully functional, or to have some limited functionality: e.g., to store the user request pending final approval.

4. CONCLUSIONS AND FUTURE WORK

We have herein introduced a system for the dynamic application of policies to composite documents throughout their lifecycles. This system can be integrated with a separately-described distributed document access control system [11] through the use of full composite document security. Security overhead for encryption, decryption, etc., is minimized through the use of "virtual" policy "parts" which can include any combination of parts of individual files, multiple files, or even other composite documents.

APEX uses a policy-specified, timely scan of its contents to ensure that documents are not changed, saved, emailed or otherwise altered in opposition to the required policy. This provides a real-time policy adherence approach that would augment, for example, role-template based policy adherence [12]. APEX is focused, therefore, on "keeping honest people honest"; that is, in preventing users who may not be familiar with all aspects of company policy from inadvertently "leaking" documents that are more appropriately restricted, Combined with an architecturally-compatible access control approach [11], and the LD-ensured logging, APEX can also prevent the attacks of intentionally dishonest users.

The work presented here focuses on the real-time analysis of the contents in the visible parts, or "data portions", of the document: header, footer, and body. However, real-time analysis of the metadata, or more sophisticated (e.g. natural language processing based) analysis of the data, in the document, is readily supported by APEX. Future work will focus on improving the complexity of policies supported (Figure 2) by the PE.

The applicability of APEX to dynamic, composite documents is by design. The APEX system is also currently being evaluated for digital rights management (DRM) and multi-media composite document applications - along with the associated necessary extensions in the PE. Further work on optimizing the PETs is also underway.

5. REFERENCES

[1] BBC News: UK's families put on fraud alert http://news.bbc.co.uk/2/hi/7103566.stm.

[2] BBC News: Q&A: Child benefit records lost http://news.bbc.co.uk/2/hi/7103828.stm.

[3] http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html, last accessed on 16 March 2010.

[4] http://spectrum.ieee.org/computing/software/microsoft-shows-off-experimental, last accessed on 16 March 2010.

[5] http://www.microsoft.com/windowsazure/, last accessed on1 June 2010.

[6] Freedom of Information Act, United Kingdom, http://www.direct.gov.uk/en/Governmentcitizensandrights/Yourrightsandresponsibilities/DG_4003239.

[7] HIPAA, http://www.hipaa.org/ last accessed 1 June 2010.

[8] E. Damiani, S. de Capitani di Vimercati, S. Paraboschi and P. Samarati, "A Fine-Grained Access Control System for XML Documents", ACM TISSEC, vol. 5, issue 2, pp. 169-202, 2002.

[9] G.V. Bard, "Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance," AISW, Ballarat, Australia, CRPIT, vol. 68, Australian Computer Society, Inc., 2007, 8pp.

[10] Data Protection Guide, last accessed 24 March 2010, http://www.ico.gov.uk/for_organisations/data_protection_guide.aspx

[11] H. Balinsky and S.J. Simske, "Differential Access for Publicly-Posted Composite Documents with Multiple Workflow Participants," accepted, DocEng 2010, 10 pp.

[12] L. Giuri , P. Iglio, Role templates for content-based access control, Proc. ACM Workshop Role-Based Access Control, p.153-159, 1997.