UI Tags: Confidentiality in Office Open XML

Lawrence Kerr(✉)

University of Idaho, Moscow, ID, USA
lawrence.kerr@vandals.uidaho.edu

Abstract. Maintaining confidentiality of data is critical, particularly in need-to-know environments. Dissemination of classified data must be controlled according to user clearance, and rests on the proper tagging of data to ensure appropriate access. The eXtensible Markup Language (XML) provides opportunity for tagging through its extensibility, and as a standard format for data storage, processing, and transmission. Its widespread usage covers a broad range of applications, especially in productivity software such as the Microsoft Office suite. This paper describes the UI Tags Project which presents a strategy for imposing security tags within Office Open XML (OOXML) format documents used with productivity suites. Leveraging the underlying XML of these document types enforces mandatory and attribute-based access control policies. Project development goals include a comprehensive system based on a native XML database which allows users to upload new documents as well as read, edit, or delete existing documents, and controls for derivative classification.

Keywords: Mandatory access control · Attribute based access control · MAC · ABAC · XML · OOXML · Confidentiality · Security tagging

1 Introduction

XML, a markup language for describing a wide variety of data, has become a common means of storing and transmitting information. XML consists of a series of nested elements, each with an associated set of attributes. The elements, attributes, and structure of an XML document is typically defined in a schema that describes each element and attribute, along with data types and legal values for each. Many common document formats are based on XML [1, 2]. The flexibility of XML makes it suitable for many of these applications, as well as many others.

Extending these formats becomes a matter of augmenting underlying XML and schemas. Extension allows insertion of further information. One particular use of this extended information might be the inclusion of security information within a document. This security information is leveraged to determine which users have specific accesses to specific parts of a document, while continuing to allow users to utilize familiar tools for creation and editing of content.

This is the high-level goal of the UI Tags document management project. This project strives to create a means of adding paragraph level tagging to Microsoft Word.docx format documents to enforce mandatory and attribute based access

K. Haltinner et al. (Eds.): CSS 2015, CCIS 589, pp. 19–33, 2016.
DOI: 10.1007/978-3-319-28313-5_2

controls by manipulating the underlying XML directly in an automated fashion. This allows a user to create, edit, and delete document content under security constraints contained within the document itself. UI Tags leverages a native-XML database to facilitate storage and retrieval of tagged documents.

The remainder of this document is organized as follows. Section 2 discusses background research in mandatory access control (MAC), attribute based access control (ABAC), Office Open XML (OOXML), and XML change tracking. Section 3 provides an overview of UI Tags with a number of goals. Sections 4 and 5 describe development stages of UI Tags. Finally, a conclusion with future work is included.

2 Background

One high level goal of UI Tags is to provide a system that supports not only MAC tagging of documents, but also a wider set of ABAC tagging. Tagging builds on Office Open XML standards, with initial targets being the individual paragraphs within tagged documents, while change tracking looks at general approaches for detecting and incorporating changes within a general XML tree.

2.1 Mandatory Access Control

The typical model of an multilevel secure (MLS) environment follows much of the mandatory considerations of the Bell La Padula model [3]. Under this model, system entities are grouped as either objects or subjects. Objects are resources to be accessed. Each object is assigned a security level which conveys its relative sensitivity, represented by the object's security classification. Subjects are users or processes that require access to the objects. Each subject is given a security label as well, referred to here as the clearance level of the subject, or simply level. Both subjects and objects can be additionally associated with some number of compartments. Based on the relationship between the level of an object and the level of a subject, as well as their respective sets of compartments, the policy determines whether to grant or deny a particular access.

The fundamental comparison of levels and compartments is known as the "dominates relationship." Comparison is possible among classifications and clearances as each represents a totally ordered set. A typical set might include a number of possible levels such as:

$$U \sqsubset C \sqsubset S \sqsubset TS$$

Here U (unclassified) is the lowest level. All other levels are higher, or more sensitive, up to TS (top secret) which is the most sensitive. Compartments are not ordered as no individual comparison exists from one compartment to another. Taken together, the level and compartments form a partially ordered set. The dominates relationship uses this partially ordered set to determine if one entity dominates another. To dominate an object, a subject must have a clearance at least as high as the object, while also belong to a superset of the object's compartments. A stronger comparison, strict dominance, requires this same relationship with the additional constraint that the dominating subject

has either a higher clearance than the object's classification or at least one additional compartment that the object does not belong to.

Two guiding properties form the basis of access decisions. First, the simple security property states that only subjects which dominate an object are granted read access. Any subjects which do not dominate an object are denied access as this would represent a leak of information to lower levels. In a read only environment this single property suffices, but in dynamic environments where objects are not only read, but created, edited, and removed, a further rule is necessary. The *-property deals with this instance. It states that a subject is only granted write access to dominating objects. This ensures the subject only writes to objects which are at the subject's level or higher. A strong *-property takes this further limiting a user to write only to objects with the same level and compartment set.

One artifact of MAC environments is the potential for polyinstantiation. Polyinstantiation occurs when some object is necessarily described differently at different sensitivity levels [4]. A user may only see one instance, matching his or her level, or a user at a higher level might be able to see one or many lower level representations of the same object. Some have described this as a necessary side effect of maintaining multilevel data, even exploiting it to maintain cover stories for various entities [4], while others have sought to limit or eliminate the presence of polyinstantiation [5].

2.2 Attribute Based Access Control

While a number of different approaches have been proposed in the literature, there does not yet seem to be a clear consensus as to an exact definition or model of ABAC ([6, 7]). A common theme in existing work is the advantage of ABAC in context-aware or pervasive computing, where the identity of a service consumer is not necessarily needed or perhaps even known ([6, 8–11], etc.). ABAC is easily configured and presents the flexibility necessary to handle dynamic environments, though this flexibility comes at increased costs as changing or analyzing permissions can become a complex task as the number of attributes increases [7].

For an attribute-based messaging system, Bobba et al. construct policies from attribute name:value pairs [9]. Access control policies here consist of conditions that when satisfied, grant access to message recipients or recipient groups. A condition is a check on values associated with one or more attributes in disjunctive normal form. These conditions form the policy that in this case governs if a user with a particular set of attributes can send a message based on the attributes of the recipient.

Cirio et al. extend a role based access control (RBAC) model with ABAC [11]. They present the difficulties with a RBAC system such as the static nature of role assignments. ABAC supplements RBAC here, adding flexibility through use of attributes for role determination as opposed to user identity. Attributes are associated with both users and resources, allowing dynamic specification of privileges for resources and association of users with privileges.

Kuhn et al. [7] present a combined role-centric model RBAC-A, where attributes are used to supplement and further constrain an RBAC model. Roles and attributes are distinguished from one another based on whether they are static or dynamic. Attributes

that are static, or reasonably static, are used as the basis for roles. These include things such as office location, position, or nationality. They are not likely to change frequently if at all. More dynamic attributes such as time of day are leveraged in the ABAC portion of the combined model. Using static roles and dynamic attributes together can significantly cut down on the number of possible roles and rules. An example system with 4 static attributes and 6 dynamic results in at most 2^4 roles and 2^6 rules, whereas a strictly RBAC or ABAC approach results in as many as 2^{10} roles or rules, respectively.

Jin et al. [6] state the necessity of more clearly and mathematically defining ABAC, while providing a model for ABAC that is capable of expressing other, more traditional models such as mandatory, discretionary, or role based access control. Under this model, each attribute is a function with a specific range that returns a value or set of values for some entity. Entities include users, subjects acting on behalf of users, and objects representing the resources available in the system. Each entity is associated with a finite set of attribute functions which return properties of the associated entity. Policies are constructed using constraints on the values of these attribute functions.

Once the basic entities and attribute sets are defined, four configuration points are defined: (1) authorization policies (2) subject attribute constraints, (3) object attribute creation time constraints, and (4) object attribute modification constraints [6]. Authorization policies return true or false as access is granted or denied. Using this framework, the authors are able to create ABAC policies that adhere to DAC, MAC, and RBAC policies.

2.3 Office Open XML

Office Open XML, or simply Open XML, is a standard that seeks to provide a stable document format while providing all features offered by pre-existing productivity applications [12]. The standard originally appeared as Ecma-376 in 2006, and has subsequently progressed to a fourth edition [13] as well as an International Organization for Standardization (ISO) standard [2]. These standards provide schemas for the markup used in various document types including word processing (WordprocessingML), spreadsheet (SpreadsheetML), and presentation (PresentationML), in addition to a number of features shared between file types including the organization of and relationship between various document components.

Each Open XML file consists of a number of different parts, all collected in a single package. The contents and layout of the package are defined in the Open Packaging Conventions (OPC) section of Ecma-376 [13]. An Open XML file is a package that contains a number of individual XML files referred to as parts that specify document content, markup, features, and relationships between different parts. It relies on ZIP technology to combine the various parts into a single object.

Contents of the package are organized in a hierarchy, with each part having a unique name that includes both the location in the hierarchy and the content name. The name represents a pack URI used for addressing specific parts within the package [2, 13]. Common parts of interest are *document.xml*, *[Content_Types].xml, app.xml,* and *core.xml.*

For Open XML word processing documents, the *document.xml* part contains the main body of text for the document. The metadata associated with an Open XML document is stored in either *core.xml* or *app.xml,* depending on the nature of the metadata. Open XML standards ([2, 13]) define a metadata schema for the core properties common to all file types in *core.xml*, but *app.xml* is reserved for extended properties - application specific items. The schema for the core properties defines fifteen pieces of information that can be used, including such items as creator, creation date, last modifier, last date modified, subject, title, and keywords. None of the metadata elements is required, and if no data is present, the part as a whole can be omitted. Repetition of elements is not allowed – for example, a document with two creator elements results in an error. Any deviation from what the schema allows in the core properties also results in an error.

The extended properties are application dependent, and allow the incorporation of information beyond the core properties. The same rules apply here – adherence to the schema, non-repeating elements, etc. The schema governing extended properties defines nearly twice as many types as the core properties, including such items as application and version; character, word, paragraph, and page counts; template used; and total editing time.

2.4 XML Change Tracking

In order to support document altering operations, the system must detect changes to both the document content and the underlying XML structure. Changes can then be collected in a delta script, essentially a list of all changes necessary to derive a new version of a document from an original. Traditionally, a "diff" between two files consists of a line-by-line comparison of different files. This works well for many file types, but misses structural information when dealing with XML tree based data [14, 15]. Thus, a number of algorithms deal specifically with the tree structure of XML in detecting and characterizing changes.

Many of these algorithms tend towards the hashing of elements or subtrees of an XML document for the purpose of fast comparisons between documents. In many instances, only leaf nodes of the XML tree are hashed, with interior nodes aggregating the hashes of their descendants, with the hope of providing a means of pruning subtrees from the possible search space.

Khan et al. present an algorithm for change detection that generates signatures for each node in the XML tree [14]. For a leaf node, the signature is computed as a hash of the contents of the node. All other nodes will have a signature that combines the signatures of all the child nodes (in this case using XOR). Comparisons between trees are then only needed if the root nodes differ. Cobena et al. match as large a proportion of the original tree in the edited tree by leveraging unique node IDs [16]. Lindholm proposes a three-way merge algorithm for XML documents [17]. A three-way merge involves an original document, and two replicas of the original that are edited independently. The merge occurs in two stages: (1) each replica is compared to the original, changes are detected, and two deltas are obtained; (2) the deltas are applied to the original, generating a new document that incorporates changes found in each replica. The changes that appear in the delta script are based on changing content of a node, or some structural change (structure here is based on the parent-child relationship in the XML

tree). Conflicts are identified as ambiguous conditions where different changes are detected in the same node in both replicas. For example, a node in the original document might contain the text "Hello," where in the replicas it may contain "Hi" and "Bye", respectively. To resolve such conflicts, Lindholm includes the options to either defer resolving conflicts until post-processing, or to allow for some level of "speculative merging" – taking a best guess.

The example provided is a date field in a document's metadata. If two edited replicas of an original have different dates associated with them, which date should be used for the merge? This question is left unanswered.

Rönnau, Pauli, and Borghoff use hashes of individual elements, as opposed to subtrees, to construct a fingerprint [18]. This fingerprint represents the context of an element and consists of the hash of the element and hashes of other elements within a specified radius. The hope here is that an edit operation on a specific element is possible in the presence of other changes – matching some of the fingerprint or context allows determining where an operation takes place. This allows for delta scripts that do not rely on absolute addressing of changes, as well as commutative deltas, where the ordering of operations does not change the end result.

In further work seeking an efficient change detection strategy, Rönnau, Philipp and Borghoff [19] frame the change detection problem in terms of the longest common subsequence (LCS), that is the sequence of leaf nodes of maximum length at a specific depth that appears in both original and edited documents. This algorithm, called DocTreeDiff, consists of three steps. First, hash values of all leaf nodes and their respective depths are computed, from which a LCS is determined. The second step involves inspecting the ancestors of leaf nodes found to be matching. Differences along the path to the root element from the leaf indicate structure preserving changes as opposed to content changes which only occur at the leaves. Finally, the third step investigates nodes for which no match exists in the opposing document – each of these represents an insert or delete operation.

3 UI Tags Overview

The UI Tags project seeks to develop an end-to-end solution which enforces tagging of specific elements within a document, while enforcing MAC and ABAC policies involving read, insert, update, and delete operations on document content. Challenges include management of the document metadata, merging document changes across security levels, and the capacity for queries over a collection of documents to produce new derivative documents.

The UI Tags Project (or simply UI Tags) encompasses a number of objectives, many of which have been addressed in prior studies including work in document-level and hardware-level tagging. Kerr [20] provided early work enforcing tagging in XML documents, using custom schemas to define tags to be used as attributes of various elements. The secure document portion of UI Tags (henceforth referred to as simply UI Tags in this proposal) extends this work to focus on word processing documents based on Office Open XML ([1, 2]), a standard format used by recent editions of Microsoft Word.

The objectives of the secure document portion of UI Tags include:

1. *Enforcement of element level tagging within Office Open XML documents.* The elements tagged depend on the type of Office Open XML document (e.g., word processing, presentation, or spreadsheet) containing the tagged elements. Current focus is on MS Word documents, with paragraph level tagging.
2. *Utilize an XML database for storage and retrieval of tagged documents.* As work moves to utilize OOXML data, a native XML database seems appropriate, as it allows for the individual processing and storage of individual OOXML parts that make up the document.
3. *Richer set of tags.* While mechanisms are in place to support tagging of documents with ABAC tags beyond the necessary MAC tags, these ABAC tags are not yet fully incorporated into the access control decisions.
4. *Tools for original classification.* Original classification is the initial process of classifying information, which is typically limited to specific entities [21]. Having a set of tools that integrates with Microsoft Word (or any other application the original classifier might employ when creating documents) will help greatly with the creation of multilevel documents, specifically limiting the burden of tag creation and application on the user, while ensuring tagging occurs in an appropriate manner.
5. *Derivative classification.* In a Master's thesis at the University of Idaho, Amack [22] extended continuing work on UI Tags adding controls for derivative classification. Derivative classification can occur when information is derived or combined from other sources. Information derived from a classified source needs to bear a like classification, while combining particular low sensitivity items might result in a combination that requires a higher classification.
6. *Read, insert, update and delete operations.* A further objective is the support of a variety of operations on security tagged Office Open XML documents, while adhering to a defined MAC and ABAC security policy. Possible operations include read, insert (a new element), update (the content of an existing element), and delete (removing an element from the document). Read operations are currently supported through eXist-db.
7. *Document change detection and merge.* In support of document-changing operations, the system must be able to merge changes back into the original document, while conforming to the constraints of the security policy.
8. *Document metadata.* One rather interesting observation of Lindholm [17] that required more work was that the document metadata tends to change inconsistently. In our MAC environment, document metadata presents a possible leak of higher-level information to lower levels. The document metadata typically contains information about the document – time created, number of words, creator or last editor – which could be used to infer more sensitive information. This is also critical in the presence of document-changing operations.
9. *Queries and reporting over document content.* The proposed system provides the capacity to dynamically create documents based on user-supplied queries over a collection of documents, while adhering to the security policy and providing provenance for the retrieved data. The documents returned by these queries will essentially be derived documents, and thus subject to derivative classification constraints.

Also desirable here is the ability to specify the provenance information returned from the query.

UI Tags is primarily envisioned as a tool for government environments where an established MAC policy exists, though it could be implemented in any environment where need-to-know determines access. In a government setting, MAC aspects of UI Tags enforce the existing MAC control requirements (users with specific clearances and need-to-know), while ABAC lends a more dynamic access control mechanism (access based on date or time, nationality of user, etc.). Outside government, medical facilities could also benefit from such a system. Here need-to-know relates to care for a patient. Admission or billing staff needs access to patient demographics, but likely does not require the level of detail into a patient's care that a physician requires.

4 UI Tags Phase 1

The initial phase of UI Tags began with mandatory access control for XML documents. Document here are based on a custom XML schema. The schema defines not only the structure of the elements within the document, but also a series of XML attributes representing the security tags. These XML attributes apply to specific XML elements of interest throughout the document. In this case, the user of the system is the subject, and the objects are specific elements within an XML document. Each labeled element bears three attributes. The first two attributes represent the classification and compartments associated with the containing element. To mitigate some editing issues arising in this multilevel environment, a preserve attribute is also used to indicate a need to partially delete an element. This early effort mitigates a number of issues specific to XML while allowing for four basic database operations. The work culminated in a prototype client/server system that allowed a user to perform any of the four basic operations on an XML database while adhering to a MAC security policy.

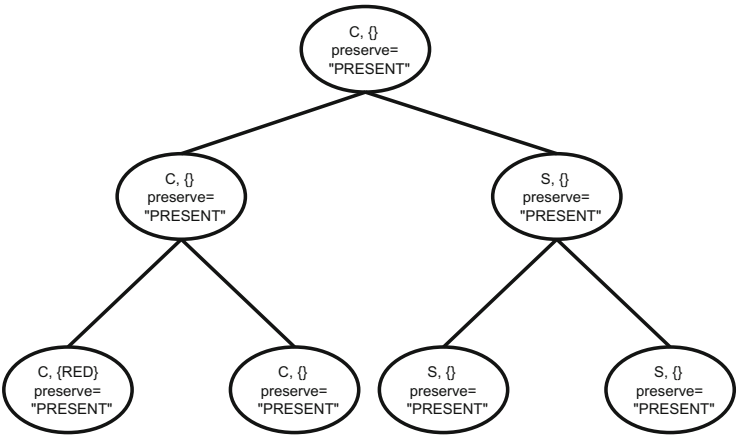

Fig. 1. Example tagged XML tree

Kerr [20] defines four basic operations for security tagged XML documents: (1) read, (2) insert, (3) update, and (4) delete. Each of these operates at the element level within the constraints imposed by both the simple security property and the *- property. For each operation, the path from the root element of the XML tree to the target of the operation is critical – if the path is not fully dominated by the user, there exists the possibility that the element exists as a child of a more sensitive element, essentially orphaning the element in the eyes of the user.

Read access is similar to a Bell La Padula style read, wherein the user must dominate the object. One added attribute, the present attribute, is used here as some editing operations my cause undesirable situations as discussed below. If this flag is present with a value of "REMOVED," any user with a level matching the level of the element flagged as "REMOVED" will not be granted access to the element as it is considered deleted as far as a user of that level is concerned. Consider a user with a confidential clearance, with membership in no compartments. If this user were to request read access to the example XML tree in Fig. 1, only three elements would be allowed: the root element, the left child of the root, and the right child of that child – only those elements bearing at least a C classification with no compartments and a preserve attribute of PRESENT. All data of the higher S classification or any elements with compartment membership are removed from the user's read view.

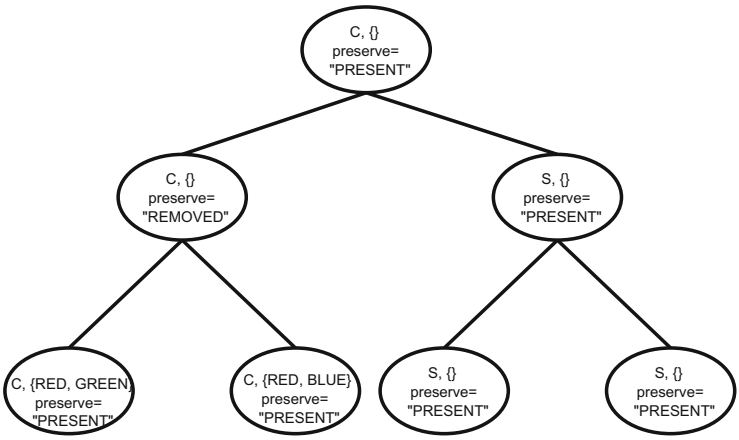

Fig. 2. XML tree with deletion

An insertion operation, where a user adds a new element, has similar concerns. The user's classification must dominate that of the path from the root to the *parent* of where the insertion is to occur. Adopting a strong *-property, the inserted element can only have a sensitivity that matches that of the inserting user's classification. If this requirement is relaxed, two problems arise: (1) a downward flow of information is possible if a high level user inserts a new element at a low level, or (2) a conflict occurs when a low level user attempts to insert an element that may already exist at a higher level. The adoption of a stronger *-property as described above, where insertions are only allowed at the user's classification, prevents this issue. Recalling a C user with no compartments,

insertions will only be allowed at any of the three readable elements, provided the insertion bears only the classification and compartments of the inserting user. Any insertion not bearing a C classification with no compartments represents a violation of the strong *-property.

Each of the other types of access involves changing the XML data, either by value of some element or attribute, or its structure as in the cases of adding or removing elements. Deletion of an element presents a number of issues. An element must only be considered for deletion if it is accessible by the user in a read capacity – a user would not realistically need to delete an element that the user is unaware of. So as with a read access, the clearance of the user must dominate the path from the root to the element to be deleted. A confidential user with no compartments would then only be allowed to remove elements tagged with a like classification.

A further consideration involves the content associated with the element. The sensitivity of descendant elements to the deleted element must be considered to ensure only appropriate data is removed. If the clearance of the user performing the deletion dominates the sensitivity of all child trees rooted at the element being deleted, there is no potential for information flow down to lower levels, as any child elements would already be unknown to lower level users. The deletion of an unknown element does not impact a low level user.

If the element contains some child elements strictly dominating the user's clearance, however, the deletion cannot simply remove the element. As the deleting user does not know of the existence of the higher level data, no judgment can be made by the user if the higher level child elements are appropriate for removal. To resolve this situation, a preserve attribute is used to mark the element to be deleted as removed. The setting of this flag must propagate to all descendant elements matching the classification of the deleted element. Once this has been set, the elements "deleted" by the user are no longer visible to a user of that clearance, but remain visible to strictly dominating users. This scenario is shown in Fig. 2, where a confidential user with no compartments has deleted a child element or the root element. As this element has children which the user was not cleared to view (and are thus unknown to the user), the present attribute is updated to REMOVED, effectively removing the element rom the user's view while retaining the more sensitive child elements for higher cleared users.

With an update action, there are two possible scenarios for the update: either a value associated with an element is being updated, or that of an attribute. In the first case, the value of an element is changed (the value here being some child element or some inner data, but not the tag itself). This operation is allowed for subjects with a clearance that is the same as the classification of the element being updated – that is, the subject's level both dominates and is dominated by the level of the object. The path in this case must allow the user to have read access to the element (as an update makes no sense for something the subject cannot read), but does not necessarily determine the ability of the subject to update the accessible element. This allows the value to be changed while avoiding any downward flow of information, and prevents any "blind writes" into higher level data that the subject should not have access to.

The second update scenario, where a subject attempts to update an attribute follows the same logic as an element update, with a few conditions:

1. The classification of the containing element determines the updateability of the attribute.
2. The classification label attribute cannot be changed by a subject as this would represent a potential downward flow of information.
3. Similarly, the compartment attribute cannot be changed.
4. The preserve attribute cannot be added or manipulated directly by users

A further concern here, as with delete, is the possibility of child elements of the edited node that are of a higher classification than the clearance of the user. These children are not visible to the user and therefore subject to loss if we allow the edit to proceed. To overcome this obstacle, we employ a series of steps:

1. Ensure both the simple security property and *-property hold for the node with content being edited.
2. Examine the *MaxDescendant* values for affected child elements. Those with a value that strictly dominates that of the edited element must be retained.
3. Set the preserve attribute to "REMOVED" of those elements with higher max descendants that would be lost by the update.
4. Perform the update to the element, making certain that the "REMOVED" or dominating descendants are retained.

Following these steps, users can edit content of a node in the XML tree without the worry that the edits of a low level might displace any child elements of a higher sensitivity.

5 UI Tags Phase 2

Since the early work of Kerr [20], UI Tags has been extended in a number of ways by coordinated efforts in three different areas: (1) an early cooperative effort among Amack, Bhaskar, and Kerr worked towards a foundational prototype system leveraging a native XML database in which individual word processing documents are stored in compliance with a MAC policy as their individual XML parts; (2) Amack's work on a derivative classification module that operates on documents as they are processed, adjusting tagged paragraphs as necessary based on their content as matched to a set of rules [22]; (3) Bhaskar's extension of the prototype, adding elements of attribute-based access control [23].

UI Tags Phase 2 work began by extending the Phase 1 tagging approach to OOXML based word processing documents. Tagging occurs in the *document.xml* part at the paragraph level, largely as the concept of paragraph exists both in the text content of the document and as a common element in the XML. Most content in the main body of text within the document is contained within paragraph elements. Other more granular elements could be used, though these do not necessarily relate to a logical unit of content in the eyes of the user (text runs, for example, are used to distinguish separate editing events or other application specific artifacts).

In addition to attaching compartments and labels to XML paragraph elements, a set of tags appearing at the beginning of each paragraph's text is also inserted. This allows for persistence of tag information when the document is opened and subsequently saved,

as any custom XML markup is removed by MS Word, per a court decision involving a patent on custom XML in Word documents [24].

Using a database for backend storage of XML content led to the selection of native XML database eXist-db [25]. With eXist-db, individual XML parts of an OOXML document can be stored, queried, and retrieved using XPath [26] and XQuery [27] expressions.

On a read request, each individual part of the document is retrieved from the database. Once the *document.xml* part is obtained, a check of each paragraph is conducted, ensuring the requesting user's clearance dominates the classification of the paragraph. If a paragraph is not dominated, it is removed from *document.xml*, resulting in a document containing only those paragraphs a user is allowed to view. In the event there are no paragraphs viewable by the user, no document is returned.

This work provides limited support for edit operations. The prototype is able to determine if change has occurred in a document using a fingerprint scheme similar to Rönnau, Pauli, and Borghoff [18], though this only works on documents that are whole, not in cases where the user is allowed a view of only a subset of the document. In cases where the whole document is available, the edited version simply takes the place of the original – no delta script is generated or applied.

In addition to this issue with change tracking and subsequent merging of different versions, one other concern of interest was identified in the course of this work. First, as mentioned by Lindholm [17], metadata management becomes an issue when merging changes between versions of documents. This is further compounded by the security policy. Document metadata may itself represent a leakage should higher level information be exposed. For example, a document containing paragraphs with a range of classifications is edited by a high level user. If the metadata reflects this change to all users, lower level users are able to view this and may infer the existence of nature of the higher level insertion simply by having the identity of the higher level user.

Building on the UI Tags foundational prototype, Amack [22] developed a means of building and applying derivative classification rules. These rules define specific strings and the classification that paragraphs containing these strings must bear. The need for derivative classification arises when new documents are created based on content in previously classified documents, with the new content requiring classification in a like manner.

Derivative classification is governed by a classification guide that consists of a number of classification rules established by an original classifier [21]. Each rule contains a string that is associated with a specific classification. Should a paragraph contain the indicated string, it is expected that paragraph will be assigned the indicated classification. Amack established an XML format for classification rules in UI Tags [22]. Rule creation is facilitated for the original classifier by a rule builder. As rules are created, they are appended to a rule file stored in eXist-db.

Derivative classification can occur when documents are uploaded to the server as well as when read requests are submitted. In either case, each paragraph is inspected for an occurrence of a classification string in the collection of rules. If a match is discovered, a check for rule expiration is made. Non-expired matching rules result in a check for dominance. Here only rules that result in a new classification that dominates the existing classification are allowed. If the rule indicates a new classification that is non-dominating

or non-comparable, the change is marked for review and an error condition explaining the non-dominating result is logged. An original classifier must then review and resolve these issues.

Bhaskar [23] introduced some elements of ABAC to UI Tags. Attributes in this context represent further indicators that supplement the MAC labeling which could include designations such as dissemination and export controls.

Facilitating these ABAC tags presents somewhat of a problem. While the document is being stored and processed, these attributes can be stored in the XML, but for persistence (as they will be removed as custom XML) a new solution is necessary as the addition of further text prepending each paragraph becomes somewhat unwieldy. To resolve this issue, the use of endnotes is used here in conjunction with XML attribute tags. Each paragraph still has the security label prefix on each paragraph, accompanied by a reference to an endnote. In the endnote, where space and readability are not issues, full details on the classification, compartments, and additional attributes are found, allowing for persistence in a way that is familiar to the user. While this provides a means of ABAC tagging, it does not use the tags in access control decisions.

6 Future Work

As UI Tags evolves, a number of areas for further work have emerged. While many of the previously enumerated objectives have been satisfied with prior work, several critical issues still remain. Among these are extending the MAC model to incorporate ABAC features, particularly in how these attributes affect specifically MAC artifacts – the simple security property, the *-property and polyinstantiation. In coordination with extending this model, implementing the read, insert, update and delete operations with support for ABAC constraints is also necessary.

Along with ABAC operation support, metadata management and change tracking are critical. Metadata needs to be managed in such a way as to remain coherent and consistent, but not leak any information to lower levels. While change tracking is supported in a limited way, further work is necessary to adequately track changes under security constraints.

Finally, dynamic document creation work must be completed, whereby a user is able to submit a query, and a document containing relevant results is returned, consisting of information pulled from various documents in the database. These dynamic documents are subject to MAC and ABAC security constraints, as well as derivative classification, and must show sources for all information retrieved.

References

1. Ecma, T.C.: Office Open XML (2006)
2. ISO/IEC 29500-1:2012 - Information technology – Document description and processing languages – Office Open XML File Formats – Part 1: Fundamentals and Markup Language Reference (2012). http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=61750. Accessed 30 October 2014

3. Bell, D.E., La Padula, L.J.: Secure computer system: Unified exposition and Multics interpretation (1976)
4. Lunt, T.F.: Polyinstantiation: an inevitable part of a multilevel world. In: 1991 Proceedings of Computer Security Foundations Workshop IV, pp. 236 –238 (1991)
5. Wiseman, S.: Lies, Damned Lies and Databases (1991)
6. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering DAC, MAC and RBAC. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 41–55. Springer, Heidelberg (2012)
7. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. Computer **43**(6), 79–81 (2010)
8. Wang, L., Wijesekera, D., Jajodia, S.: A logic-based framework for attribute based access control. In: Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering, pp. 45–55 (2004)
9. Bobba, R., Fatemieh, O., Khan, F., Gunter, C.A., Khurana, H.: Using attribute-based access control to enable attribute-based messaging. In: 2006 22nd Annual Computer Security Applications Conference, ACSAC 2006, pp. 403–413 (2006)
10. Frikken, K., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. **55**(10), 1259–1270 (2006)
11. Cirio, L., Cruz, I.F., Tamassia, R.: A role and attribute based access control system using semantic web technologies. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2007, Part II. LNCS, vol. 4806, pp. 1256–1266. Springer, Heidelberg (2007)
12. Ecma Technical Committee 45, "Office Open Xml Overview." Ecma International (2006)
13. Standard ECMA-376 (2012). http://www.ecma-international.org/publications/standards/Ecma-376.htm. Accessed 30 June 2013
14. Khan, L., Wang, L., Rao, Y.: Change detection of XML documents using signatures. In: Proceedings of Workshop on Real World RDF and Semantic Web Applications (2002)
15. Peters, L.: Change detection in XML trees: a survey. In: 3rd Twente Student Conference on IT (2005)
16. Cobena, G., Abiteboul, S., Marian, A.: Detecting changes in XML documents. In: 2002 Proceedings 18th International Conference on Data Engineering, pp. 41–52 (2002)
17. Lindholm, T.: A three-way merge for XML documents. In: Proceedings of the 2004 ACM Symposium on Document Engineering, pp. 1–10 (2004)
18. Rönnau, S., Pauli, C., Borghoff, U.M.: Merging changes in XML documents using reliable context fingerprints. In: Proceedings of the Eighth ACM Symposium on Document Engineering, pp. 52–61 (2008)
19. Rönnau, S., Philipp, G., Borghoff, U.M.: Efficient change control of XML documents. In: Proceedings of the 9th ACM Symposium on Document Engineering, pp. 3–12 (2009)
20. Kerr, L.: Polyinstantiation in multilevel secure XML databases. MS Thesis, Department of Computer Science, University of Idaho, Moscow, Idaho (2012)
21. Executive Order 13526- Classified National Security Information | The White House (2009). http://www.whitehouse.gov/the-press-office/executive-order-classified-national-security-information. Accessed 22 October 2014
22. Amack, A.S.: Automating derivative classification in multi-level secure documents. MS Thesis, Department of Computer Science, University of Idaho, Moscow, Idaho (2014)
23. Bhaskar, D.V.: Software Design Specification for Storing Multilevel Secure XML for Easy Retrieval. University of Idaho, Moscow (2014)
24. Microsoft Corp. v. i4i Ltd. Partnership - Supreme Court (2010). http://www.supremecourt.gov/opinions/10pdf/10-290.pdf. Accessed 25 November 2014

25. eXistdb - The Open Source Native XML Database. http://exist-db.org/exist/apps/homepage/index.html. Accessed 22 October 2014
26. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M.F., Kay, M., Robie, J., Siméon, J.: XML path language (xpath). In: World Wide Web Consort. W3C (2003)
27. XQuery 1.0: An XML Query Language (Second Edition) (2011). http://www.w3.org/TR/xquery/. Accessed 25 November 2014

http://www.springer.com/978-3-319-28312-8

Cyber Security
Second International Symposium, CSS 2015, Coeur d'Alene, ID, USA, April 7-8, 2015, Revised Selected Papers
Haltinner, K.; Sarathchandra, D.; Alves-Foss, J.; Chang, K.; Conte de Leon, D.; Song, J. (Eds.)
2016, IX, 143 p. 60 illus., 5 illus. in color., Softcover
ISBN: 978-3-319-28312-8