Five Examples of Web-services for Illustrating Requirements for Security Architecture

Ronald G. Addie, Sam Moffatt, Stijn Dekeyser
Department of Maths and Computing
University of Southern Queensland
Toowoomba, Queensland, Australia 4350
{ron.addie,dekeyser}@usq.edu.au,pasamio@gmail.com

Alan Colman
Swinburne University of Technology
Hawthorn, Victoria, Australia 3122
AColman@groupwise.swin.edu.au

***Abstract*—The tension caused by the need for expressive power when formulating security rules and the need to keep computational complexity low when undertaking the necessary access rule evaluations is a major challenge in the formulation of good security architecture. This paper provides five examples of security in web services. which illustrate this tension. These examples highlight the need for more expressiveness in the rules used to express policies in three cases, and in the other the fact that XACML appears to have nearly adequate expressiveness without undue complexity. Each example is expressed first informally, by describing a service which could concievably be provided in a web services architecture, then the example is also outlined using either XACML, first order logic or both.**

I. Introduction

In formulating security architectures for web services there appears to be a conflict between:

(i) the need for expressive power in expressing policies;
(ii) computational simplicity in access algorithms; and
(iii) a natural desire to use the same language for policies and access rules.

However, perhaps all these goals can be satisfied at once. Limiting the expressive power of the language for expressing policies because of the need for computational simplicity may be a mistake.

The reason why abstract rules which may be very difficult or even impossible to check may still be useful is that rules can be useful, or even vital, without ever having to be evaluated.

A good example of this is rules defining integrity constraints in a database. Suppose a database is designed to preserve the rule that for every item in table `carparts` there is a exactly one entry with the same `partno` in the table `currentstock`. This is a useful rule and it might be necessary for this rule to be true for many of the systems processes to function correctly – including the evaluation of rules used for access control and security in general. But checking that this rule is true would be a very lengthy task, and is not necessary, because the database uses checks carried out very efficiently at insert time which ensure that this rule is true at all times.

A second example involves the traditional procedure for authentication of users of a multi-user computer, e.g. running unix, is to check the password entered by a user with the encoded password in the *password file* (typically `/etc/passwd`). This procedure relies on the assumption that the password file contains exactly one password for every user, however this fact will not be checked as part of the authentication process.

This tension between the need for expressiveness in policies and the need for speed and simplicity in the actual check is further exacerbated by the diversity of applications which arise naturally in this field – from shared access to files via a network [1], on the one hand, and enforcement of complex rules for mediating collaboration between colleagues who share and manage the diverse roles of a conference organisation and its clients [2], on the other.

A tentative solution to this problem is to define two *different* languages – one for defining policies, and another for defining algorithms for access; this approach is implicitly adopted by some authors, however there are also clear advantages in simplicity of the overall architecture if the same language is used (perhaps with restrictions, depending on the use) for both purposes.

In addition to defining policies and access rules in [2], [3] an even higher level project of *policy validation* is envisioned. The policy validation concept investigated here is to check that appropriate coalitions of users *can access* the intended targets and inappropriate coalitions can't. The reason these issues are particularly pertinent in these papers is that the policies investigated include rules by which users may alter each others access rules. However, the introduction of policy validation as well as policy definition begs the question of why the validity rules are not also adopted as part of the policies themselves.

Let us now focus specific details of the language used for definition of rules. Should the quantifiers of first order logic (*there exists x:* $\exists x$, and *for all x:* $\forall x$) be used in a language for defining security validation, security policy, or access control? If these quantifiers are used, how are they to be interpreted? The problem with quantifiers is that evaluating expressions involving quantifiers is often undecidable, and even when decidable, determining this may be a very lengthy task.

The differing expressive power of theories with restricted use of quantifiers has been investigated in the mathematical literature [4, Chapter D.1]. It is clear that allowing more use of quantifiers increases expressive power, and barring the use of quantifiers prevents certain concepts from being expressed. Evaluating logical expressions which use quantifiers, even

when the domain over which they operate is finite, may have to be disallowed during the process of deciding whether to allow access. But, as explained earlier, the use of quantifiers in rules may be useful even though such rules are never evaluated.

First order logic also widely uses *schemes* for generating axioms in addition to simple axioms; One of the examples below uses an axiom scheme in order to define a service.

In this paper we present five examples which can be used to consider issues in relation to security for web services. The first example is a realistic general-purpose example in which rules similar to the database integrity rules arise naturally – in this example it appears that quantifiers are necessary in order to be able to express the obvious required security policy; the second example concerns a service for managing orders which requires an integrity check for the access rules to be valid - hence quantifiers appear to be necessary in the definition of rules for this service also; the third example concerns a *delegation service*. In this case there is no need for quantifiers in order to define the natural policies for the service, but a simple type of second order logic (which can be achieved by axiom schemes) is required. The fourth example is the access rules for the NTFS file system. It seems that specifying the access rules for NTFS can be achieved using XACML in its present form. The last example comes from the field of social networking, where the evolution in security rules in recent times has been rapid and problematic, confirming the thesis of this paper that more expressive and flexible tools are needed.

The remainder of this section provides a literature review and a definition of security architecture. The next five sections are devoted to the five examples and the last section contains some concluding remarks.

A. Existing Literature of Examples

The existing literature of examples of security for web services is diverse and extensive. We merely attempt, here, to present some of the important examples from the literature, and to identify some important themes of this literature.

In [1, p729] an example is given of authentication and *delegation*, leading, after several steps, to access to a requested resource being granted. *Delegation* of authority is a strong theme in the literature of examples. This paper makes use of propositional logic as the language within which rules for secure access should be expressed, which is also a common practice in much of the literature.

In [2], [5], [3], [6] three different examples are presented to illustrate two separate but closely related intiatives: (i) a scheme for generating XACML code for access rules, and (ii) a validation algorithm for access rules which seeks to demonstrate by a model-checking method that desired access is achieved, and undesired access is prevented.

The expressive power of XACML and RW (the more direct language in which rules are initally expressed, in [5], [3], [6]) are tested by seeking examples where: (i) access rules depend on the state of the system being accessed and also, potentially, on data in documents in the system; (ii) the ability to control access to the parameters of access control is also controlled by the system; (iii) delegation of rights to access any aspect of the system can be effected; and (iv) it is possible to express access rules which require collaboration between multiple parties.[5]

One example models the operation of a paper reviewing system, with editors, principle reviewers, secondary reviewers, and authors. Complex access restrictions and rules are required in this system. A second example is set in a health-care setting and *delegation* is a required feature of this system, i.e. the facility to allow one client to stand in for another, in connection with certain *roles*. A third example describes an Employee Incentive scheme in which the key resource under the control of the director, managers, and other staff, is the option for employees to be awarded bonuses.

B. Security Architecture

By *security architecture for web services*, we mean the protocols, algorithms, and declarations (rules) which are put in place in order to ensure that the identities of all the parties to the transaction(s) are authenticated and clear throughout, and in order that all the *requirements* of all parties are met at all times.

For example, if two parties, A and B, engage in a transaction, and party B requires party A to hold an account with credit at least $100.00, the security architecture should be able to ensure that this is the case as a pre-condition of the transaction.

This example already shows how security architecture overlaps unavoidably with *business rules*. Rather than resist this overlap, we feel that it is best to seek an approach which can deal with both in an integrated manner with the expectation that a unified approach will deal with both problems (and also access control – which might be regarded as a subset of security architecture) more effectively.

The issue of managing identity has been deliberately excluded from consideration in this paper. A treatment in which management of identity (including authentication and communication of information about participants which can be used to confirm identity) is undoubtedely a very important aspect of security architecture, however it is also useful to consider *other* issues, and those are the focus of this paper.

C. How to define a service

In order to ensure that a web services exchange is valid we must prove that at no time, during the course of the exchange allowed by the security architecture, will any party participate when the rules that it has adopted are not satisfied. We also expect that when all the rules adopted by a participant *are* valid that it *will* participate, and that includes responding to the requests of other parties within a reasonable time.

So, if we confine ourselves just to the security aspects of a web service, in order to satisfy ourselves that a service is well-defined, and therefore is a valid service, it is sufficient to show that the service will behave according to this simple principle: to continue responding to requests precisely when the all the rules that this service provider has defined are valid.

In each of the following examples, the task we set ourselves is precisely this: to show *how* we can define a service so that it behaves in this manner.

In particular, if an informal statement of the rules required for a service is clear, which is the case in all the examples below, our primary task is to answer this question: "Can the informal rules for this service be defined in the proposed security architecture?"

The two security architectures we consider, in the examples (neither of which can be regarded as complete in all details), are: (1) one based on XACML [7], [8], [9], in the sense that this is the language used by all parties to define their access control rules; and (2) one in which the language used for defining access control has the same expressive power as first order logic (the Predicate Calculus) [10], [11], [4].

II. Example 1. Academic Records

A. Informal description

The use of computers to store an academic record, and networks to disseminate such academic records, and access statements, from the student in question, to restrict access or open access to academic records is an excellent application of secure web services.

An informal academic record is shown in Figure 1 and a more formal academic record is shown in Figure 2. A *completely* formal example, which has been checked using an implementation of a secure web server is presented in Section 6 of [12] .

Here is a *scenario* for the use of an electronic academic record:

- The student (WLA), approaches a potential employer (NBC), claiming to have a Bachelor of Certification (BoC) from UoC. This claim is supported by a reference to the academic record. See Figures 1 and 2.
- If NBC expresses sufficient interest, WLA passes a signed document which states that NBC should be allowed to access their academic record at UoC and informs the University of the details of this document that should be passed to NBC, including the validity period. See Fig. 3.
- NBC accesses the academic record at UoC, providing their identity and the access document, thereby taking advantage of the allowed access set up by WLA at the previous step.

The access software reports, in the course of retrieving the document, that it is genuine, that it is the academic record of this same WLA, that the University is genuine, and that WLA has a Bachelor of Certification issued at such-and-such a date.

This example is interesting because: (i) there are more than two parties involved in the key transactions in a manner where each has a distinct role; and (ii) the services provided are intrinsically concerned with security. This supports the contention that business rules and security rules may intimately connected.

```
The following document is an academic
record for Wendy Louise Angstrom.
Wendy Louise Angstrom has completed
the following units, for partial completion
of the Bachelor of Certification and
received the indicated grades:
acadrecxacml
Unit              Grade
----              -----
66311             A
66101             B
...
```

Fig. 1. An Academic Record - informal

```
<document type="academic record"
   scope="this whole figure">
<statement type="definition">
<object>the student</object><isDefinedAs>...
</statement>
<statement>
....
</statement>
</document>
```

Fig. 2. An Academic Record – formal

B. XACML

There are several transactions to consider here: the request from WLA to UoC to make the academic record available; the message from WLA to NBC which says that it will be available, and the request for the record from NBC to UoC. We confine ourselves to the latter – extension to the other cases is not difficult.

Because the permission to access the academic record is transitory, we assume that a policy set is created as a consequence of the request by WLA to UoC. This policy set expresses the fact that NBC can access the transcript of WLA between the dates nominated in the request from WLA to UoC. The XACML is not shown here because of space limitations, but is available on request.

Although the rules appropriate for this example can be expressed in an XML form which in a broad sense can be interpreted as XACML certain aspects of this example go beyond the scope of the XACML standard as it now exists. In the next subsection we will see that features of first order logic which are not currently allowed for in XACML arise naturally in this example. Another issue is that XACML rule-sets must be applied to the problem in hand via one of the standard *algorithms*, whereas the statements encoded as XML in this example rely on being interpreted according to the rules of logic. None of the standard XACML algorithms for applying rules consistently respect the logical meaning of all rules.

For example, the algorithm which allows access only if there are no rules which deny access and at least one rule which allows access will not respect rules which allow access whenever there are rules which deny access. This means that the applicability of rules in XACML is conditional on the context in which they are interpreted; a literal intrepretation

```
WLA permits NBC to access
citp://UoC.edu.au/acadrec/q123
```

Fig. 3. citp;//q123.graduate.UoC.edu.au/authacadrec/NBC

of rules may be misleading, which is a disadvantage. All the XACML algorithms are deterministic and at most $O(n)$ in complexity, where n is the number of rules. More sophisticated algorithms might not achieve this degree of efficiency, and since in many applications access control must have low order complexity, developers of XACML have been reluctant to endorse more sophisticated rules. The architecture for access control described in [12] makes use of a more sophisticated algorithm which still has low complexity.

C. First-order Logic

Suppose it is desired to offer scholarships to all students who complete a year of study in which they achieve a grade of better than or equal to 6 (where the top grade is 7), in all their courses for any year. Adopting the predicate `grade(S,C,G,Y)` to mean "student S achieved grade G in course C in year Y, the rule fora student to be awarded a scholarship is

$$\big(\forall C \forall G \texttt{grade}(S,C,G,\texttt{thisyear}) \Longrightarrow G \geq 6\big). \qquad (1)$$

In this instance, checking that a condition defined by a quantifier holds is both necessary and feasible.

In many situations, it is natural to create a list of students satisfying condition such as (1), and to use membership of this list as a proxy for checking the condition. Again, even though the condition is important, we do not need to check it explicitly.

III. Example 2. Order Optimization

Suppose a web server provides the service of *checking orders* against a list of alternative providers. Alternative providers can also be added to the list by a process involving submission, and confirmation or denial.

A. Informal description

We assume that there is, at any time, a list of alternative suppliers and for each supplier it is possible to inquire if this supplier provides an item, and then, if the item is provided, to inquire as to the price and delivery cost of that item or a list of items.

The service provided is to check that there are no changes which can be made to the order which will reduce the total cost. For simplicity, we limit the types of changes to ones which change the supplier and we ignore delivery charges. It is therefore sufficient to consider changes for one item at a time.

Informally, the defining rule for this server is that for any item, and any supplier in the list of allowed suppliers, if we change the supplier for this item to the new supplier, the total cost will be greater than in the order as it stands. This rule can readily be stated in first order logic, as we see in the next subsection, but the statement requires quantifiers, which means that it can't be expressed in XACML as it currently exists.

B. First-order Logic

Let `O` denotes the order, $\texttt{cost}(O)$ denotes the cost of order O, and $\texttt{chsup}(O,I,S)$ denotes the order obtained from order O by replacing the supplier of item I by supplier S. Then the defining rule for this service can be stated in the following form:

$$\forall S \forall I \big(I \in \texttt{O.itemlist} \Longrightarrow \texttt{cost}(O) < \texttt{cost}(\texttt{chsup}(O,I,S))\big).$$

IV. Example 3. Delegation

Suppose a web service, `DS`, provides a higher-level service of *delegation* which is intended to work with *any* other web service, `WS`. Informally, a client, `S` (secretary, for example), of `WS` sends a message (or a request) to `WS` and `DS` which states that another client, `M` (member, say), should assume all the authorities and responsibilities of `S` at `WS` for a period of time `date1` to `date2`. These requests should have no effect unless an assenting message (or request) is sent by `M` accepting these authorities and responsibilities. If such a request is made, however, during the relevant period, the web service `WS` should operate as if the client `M` is able to substitute for `S` in relation to their authority and responsibilities.

A. Formal Statement

The delegation service should refuse to act on this request unless it is accompanied by proof that the requestor is xxx, and xxx is the treasurer. If this is the case, the action taken by DS should be to put in place the following policy, which refers to the site AF:

```
If P(x) is any statement with one free
variable then
(P(xxx) ∧ date > StartDate ∧ date < FinishDate) ⟹
P(yyy)
```

Observe that this statement refers to *any statement* P. This appears to make use of second order logic, since it is a statement which refers to other statements. However, by convention, statements of this sort are allowed in first order logic by allowing a *scheme* of axioms where, e.g. P, varies over all possible expressions of the indicated kind, i.e. statements with one free variable x. Such a scheme of axioms may be (and usually is) infinite in extent, although it is usually assumed that the axioms can be enumerated by means of an algorithm.

For example the principle of mathematical induction, for the natural numbers, must be expressed in this way [10, p21]. In fact, as a consequence of this necessity of using a scheme of axioms to describe the natural numbers, many if not most axiomatic systems used in mathematics make use of infinite schemes of axioms. Infinite schemes of axioms are not ruled out in applications because it is never necessary to use more than a finite number of the axioms in an individual proof, and because the axioms that are needed can be generated, or checked, by means of the algorithm which enumerates them.

Let us suppose that the site DS stores this *delegation* assertion at the address http://Delegator.com.au/wssecurity/treasurerdelegation10-4-2010-20-4-2010.xml. In order to put this delegation assertion into effect, the site DS must now

send its response to the request from AF. This response should include the following:

```
The delegation you requested
is now in place at
http://Delegator.com.au/wssecurity/treasurer-
delegation10-4-2010-20-4-2010.xml
```

Let T denote the collection of rules at http://Delegator.com.au/wssecurity/treasurer-delegation10-4-2010-20-4-2010.xml. This might include rules specified by means of a scheme, as well as specific rules. As part of its response to the request from DS, the AF site should make sure that the following policy assertion is in place:

$$\text{If } \phi \text{ is any statement such that } T \vdash \phi, \text{ then } \vdash \phi. \quad (2)$$

The symbol $\vdash$ denotes "is valid". When used in conjunction with a theory, e.g. T, as in $T \vdash \phi$, the interpretation is that ϕ is either an axiom in the theory T or follows from them. When T is omitted, the interpretation is relative to the current context. So (2) says that the axioms set up at DS for AF to use for delegation are adopted as axioms at AF.

V. Example 4. NTFS

Describing NTFS access control is sufficiently complex that it immediately begs the question: shouldn't this system be described by means of a language? Since a language for security controls exists, namely XACML, let us consider how it might apply to NTFS.

A. The NTFS Access Model

The NTFS Access Model is based on the use of access control lists (ACLs). Each file and directory (container) has an ACL which is composed of a list of access control entries (ACEs). Access to a file is controlled by its ACL and the ACL of its container, and possibly its container's container, and so on. An ACE defines who the entry applies to (user or group), a list of actions and if they are permitted or denied and an ACE of a container type may optionally nominate if the ACE will be inherited to child items.

With one exception, the rules are evaluated by traversing the ACL of the target resource until a match (either a deny or allow) for a given action is found. If a match cannot be found in the ACL for the target resource, the inherited access controls in the ACL's of the target resources' ancestors are traversed until either a match is found or all ancestors and their lists have been exhausted without match. If no match was found then the permission is denied. The exception is that in the case of files, some access attributes of the file's container take precedence over those of the file itself.

ACE's have two flags to control inheritance: object inherit and container inherit. If object inherit is set it will apply to all child files. If container inherit is set it will apply to all child folders. If neither is set then the ACE is not inherited. Both object inherit and container inherit can be set and permissions will then apply to both child files and folders.

It is important to note that with NTFS rules are checked starting from the target resource through the ancestor list to the root of the file system. Thus (except in the case of files vis-a-vis their containers) the rules of the children override any rules of the parents. Access control list entries are also evaluated in the order that they are found and evaluation stops after the first match is found. This means that if the first match is an allow then any later matching rules, including a deny, will be ignored. Finally while Windows defines a "canonical" order for access control entries – deny rules first then allow rules (so deny rules take precedence), this canonical order is not mandatory.

The exception to this upward order of evaluation of ACLs, is that when attempting to open a file, its container's "List File/Read Data" permission will be consulted before consulting the ACL of the file. This behaviour is unique to files. Folders do not exhibit this behaviour, e.g. you can open a folder so long as you have permission to do so over that folder, even if the parent folder does not permit the action.

A common myth about NTFS is that deny rules inherently overrule allow permissions. By default Windows will structure the ACE's in the ACL for an item with deny entries ahead of allow entries. As such this makes the statement true as the deny entries are tested before the allow entries. However third party applications can construct an ACL that isn't in this order, for example by positioning allow entries ahead of deny entries. Since entries are evaluated in the order they appear in the list, an ACE that allows access ordered before one that denies access will result in the request being granted access. Furthermore an ACL that grants access will overrule a deny ACE that is inherited from an ancestor – and this is true without recourse to any third-party software for managing ACLs.

One final rule is that the owner of an object can always change the discretionary access control list. So while another user might set the access control entries to prevent access to a file by its owner, the owner will always have effective rights to read and change the permissions of the object.

NTFS defines four categories of action:

- Generic Rights — These are similar to the UNIX model, generic read, generic write, generic execute and all permissions. All items (files and directories) have these actions. It should be noted that generic *all* permission is a specific bit that matches all permissions of generic read, write and execute.
- Standard Rights — These are NTFS specific ACL's such as synchronise, write owner, write DAC, read control and delete. All items (files and directories) have these actions. "Write DAC" controls altering the discretionary access control list, or the permissions of the item. There is a permission for being able to access the ACL for the item (Access System Security) and maximum allowed is a bit used in access control checks to return the maximum allowable permissions for an item (it isn't stored on the file system).
- File Rights — Execute; Read, Write and Append data to the file; read and write the attributes and extended attributes for a file.
- Directory Rights — List, add file, add subdirectory, read

and write extended attributes, traverse, delete child and read and write attributes.

While there is space in the data structure for extra actions (as noted by the Figure 4), the access model does not allow an arbitrary action to be defined within the existing security system.

Finally, privileges are special rights assigned to users or groups that enable overriding other aspects of the file system. NTFS has "Back up files and directories" privilege, and "Take ownership of files or other objects" privilege. The back up privilege permits applies to all files and folders however it has to be invoked by a particular application.

B. Transforming NTFS Rules into XACML

To properly emulate NTFS, at least two policies need to be defined. The first of these is the policy that defines overall rules of the system. The first rule in this system policy is that the owner of a file can always change the DACL of the file. This means that the owner can never be locked out of their own file. Other core rules are the privileges that permit access. Privileges in this context are evaluated as their own action type and are located ahead of any other rule. This policy is referenced by every other policy set first prior to defining their own rules to ensure that these system rules have priority.

From here further policies need to be defined for folders and for files. As inheritance for access controls on folders can apply to either just the folder itself (no inheritance), just to child *files*, just to child *folders*, or to *both* child files *and* folders. To handle this, four policies need to be defined so that each of the possible inheritance options a folder can have are put into effect. Files need their own rules which should be evaluated prior to that of the parent container – except when the parent has a deny for the read data/list files action (this requires another policy for folders).

The default manner in which rules are evaluated in the XACML model of NTFS implemented here is the "first-applicable" algorithm. This means that the first matching rule, be it deny or allow, is returned. This is broadly similar to the approach adopted in in NTFS so it is difficult to conceive of adopting any other algorithm.

Using this algorithm, at a first attempt, a reasonable strategy appears to be to use the following policies in the order indicated:

- System Rules
- The folder's "read data/list files" rule
- File rules
- Folder rules that are inheritable to files
- Folder rules that are inheritable to folders
- Folder rules that are inheritable to files and folders
- Folder rules that are non-inheritable

However, there is a problem with this model in that folder rules that are inheritable to files and folders might deny a user access however, if there is a rule that *allows* the user access in the folder rules that are inheritable to files the user will be granted access. This is not an accurate representation of what would normally occur because normally, in NTFS ACL's, deny rules appear before access rules, and so in this example access should be denied. To resolve this the three inheritable rules can be split into "deny" rules and "permit" or "allow" rules, and all the deny rules can be invoked ahead of all the deny rules. This leads to the following policies, in order of execution:

- System rules
- The parent folder's "read data/list files" rule
- File rules
- Folder deny rules that are inheritable to files
- Folder deny rules that are inheritable to folders
- Folder deny rules that are inheritable to files and folders
- Folder permit rules that are inheritable to files
- Folder permit rules that are inheritable to folders
- Folder permit rules that are inheritable to files and folders
- Folder rules that are non-inheritable

This permits reasonable emulation of the way the NTFS model works (except for the possibility of ACL's not in canonical order).

Sample System Rules Policy

In [13] there are various examples of applying XACML to NTFS access control rules. Pages 49 through to 57 present extensive examples of how to represent various NTFS access controls utilising XACML. Page 50 introduces the first policy which covers a folder that has inheritable permissions where the owner can read and write the files. The second policy is introduced on page 52 that is specific to an individual file with a permission to explicitly exclude a user identified as "john". Finally a third policy starting on page 55 prevents user "fred" from writing to a folder in such a way that the permission is not inherited.

Through reviewing the various permissions it appears to be extremely complicated to achieve the rules exhibited by NTFS. Many pages are spent in the required amount of XACML to express an equivalent concept to the NTFS rules. Some of this can be attributed to the nature of XML however much is due to the complexity of the NTFS permissions model.

C. Limitations

The XACML samples referenced above are intended to indicate that the NTFS access rules can be successfully modelled using XACML. However, there are some assumptions and constraints which limit the scope of this model:

- It is assumed that all NTFS ACEs are in "canonical" order – which is not mandatory;
- the concept of ownership has not been fully modelled in the XACML examples referenced above.

The behaviour of the NTFS model is to evaluate the ACE's stored in an object ACL in order and return the result of the first match. By default, the Windows GUI positions deny rules ahead of allow rules however this is not a technical limitation. It would be possible to create an allow ACE ahead of any deny ACE which then could be followed by allow ACE's. This situation is not handled but would be possible, however it would just complicate the transformation further.

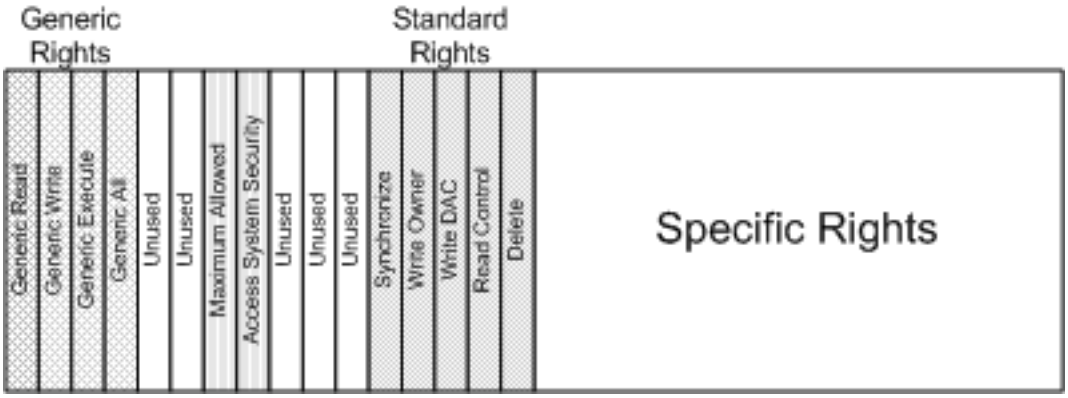

Fig. 4. NTFS Access Mask Layout

XACML doesn't necessarily support the concept of an "owner", however this could be emulated. For example, a special "owner()" function could be used to match subjects that own their resources.

VI. Facebook

Facebook is the most popular social networking site available today with more than 400 million active users. Facebook's platform has evolved from a closed social network at a university level to increasingly publicly data exposed to the entire internet. Facebook's privacy model has under gone significant changes [14], [15] and at present very little is in fact protected. Both wall posts and photos default, at this time, to enable *everyone* to view them, although users can change the defaults to permit the user to limit their access.

Facebook permits users to be granted or explicitly prohibited access to either a wall post or a photo. Named lists of users can also be created which can also be used for access control purposes. Photos can also be stored within albums which also have their own security controls.

Facebook's access control allows each item to have its own access control. While this is effective, it does not scale with the number of items managed by a user. Take for example two Facebook albums of photographs. One set is from a family event whilst the other is from a friend's party. In this situation you would rather not have your friends perhaps see your family photos whilst you may not be interested in your family seeing what antics you get up to with your friends.

An example like this, presented by Paul Adams in The Real Life Social Network [16], demonstrates just a situation. In this example, Debbie, the social networker, is a member of various groups: her family, friends from LA, friends from San Diego and a group of ten year old kids to whom she teaches swimming. Debbie's LA friends work in a gay bar and share pictures which she comments on. Debbie realises that because of her commenting on them, the ten year old kids can now also see these photos. Adams notes that Debbie was upset for not realising that the system allows this to happen.

This adds an extra dimension to the problem. To handle the original situation one could simply create two lists of people. One list for family members and another list for friends. Then with both of these groups with each item ensure that the appropriate security level is assigned. However in Debbie's particular situation the problem is a lot more complicated: Facebook doesn't have access control for comments. Her commenting and liking those photos immediately shared it with all of her friends — the LA friends, San Diego friends, her family and the ten year old's.

A similar situation exists with tagging in Facebook. People can be tagged in videos, images, notes and other updates. People can untag themselves however they have no easy ability to prevent someone from tagging them. Until such time as the person logs in and untags themselves, the picture can appear in their friends stream if they can see it. People could tag undesirable pictures of others and this will appear to their friends even if the person tagging the photo isn't a direct friend (e.g. a friend of a friend).

Facebook offers reasonable levels of access control for content that you provide. However the "social graph" provides the ability for people to be linked either through their own actions (commenting on other peoples content) or through the actions of others (being tagged in photos provided by others) which offer no means of protection and is immediately exposed to all friends regardless of the actual context.

It is this concept of "context" that Adams introduces with different groups which we naturally socialise within. Adams suggests that we need to have support from the social networking systems to provide this functionality. Cluster analysis, based on the interconnectedness of the friends in the network could, provide a automatic grouping, although friend networks do not necessarily form consistent clusters.

A. A new security-related feature

Consider a "new feature" which might be added to facebook, as follows: photos can be *tagged*, in facebook, by the owner of the photo. This associates another facebook user with the photo, and usually implies that the person is visible in the photo. We would like the *tagged* person to have certain rights in relation to this photo by virtue of being tagged. All sorts of possibilities can be conceived (this is the problem, and perhaps also the opportunity presented by this example). For simplicity let us consider just one new feature in relation to *rights*: the tagged person can *show* the photo to other users. This means that the tagged user is indicating that a certain

additional person, other than the owner, can access the photo.

B. A tentative solution

When the tagged user indicates that a certain additional user should have access to a photo, the system should add this additional person to a list of users with access to the photo. This would probably be achieved by a simple action with the mouse which updates the access list and sends a message to the person being given access informing them of the availability of the photo. The *system* would then need to check this list when access to the photo is being granted. The list of users with such rights would probably need to be kept separately from other lists so that it can be managed appropriately.

It would be possible to hard-code the necessary access procedures into the software for the system, however doing so would compromise the maintainability of the system and potentially lead to difficult to control bugs. A better approach would be to implement a generic rule-based system in which rules such as these would be encoded in XACML or another formal language.

The rules that we have already described, informally, for this system, are as follows:

- photo owners can tag photos with the names of users;
- tagged users have the right to provide other users with access to photos, which actually means adding these users to a list of users allowed access by taggers;
- users in the latter list can view such photos.

C. Implications

The most significant implication of the rules described above is that a user in the tagger-generated user access list can access a photo. Formulation of a rule of this nature in a language such as XACML is not difficult to envisage. *Enforcing* such a rule is also not difficult to envisage.

What is more difficult to envisage, but nevertheless is clearly required, is that rules such as this should be formulated, formalised, validated or tested, implemented, modified, used, and eventually decommissioned, on a regular basis. Facebook gives us an example where the complexity of the access rules is sufficient that a regular process of rule development appears to be necessary merely to support normal operation of the system.

This suggests that the expressive features in the language for expressing rules need to be as full as possible – ideally *complete*, in the sense that *any* reasonable rule can be specified within this language.

VII. Discussion anc Concluding Remarks

The access control architecture implied in the use of XACML is unable to accomodate examples of Sections II–IV and Section VI .

Accomodating these examples in XACML appears to require fundamental changes, such as introducing a new algorithm for applying rules to access control, adding quantifiers to the vocabulary of XACML, and including *schemes* of rules as well as simple rules. These changes would enable XACML to be as expressive as first order logic which is a well-established object of study in mathematical logic, with a highly developed theory, including a completeness theorem [10], [4], which suggests that once these extensions of XACML are incorporated, no more extensions will be necessary.

In this paper we have seen five examples which illustrate why each of these extensions is needed.

Although the extensions to XACML introduce additional complexity, it has been shown that these extensions can be introduced in a way that the potential additional complexity is avoided. For example, we could add an auxiliary access control algorithm which denies access whenever application of the primary access control algorithm is unable to complete its determination within a specified time.

Speed of evaluation of access rules is of paramount importance in many situations, however we have seen that this does not preclude appropriate use of rules which *potentially* introduce undesirable complexity.

Three of the new examples presented here confirm the usefulness of additional expressive power in the definition of security policies, confirming the hypothesis with which the paper began, that additional expressive power may be needed for XACML to achieve its intended objectives.

References

[1] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin, "A calculus for access control in distributed systems," *ACM Transactions on Programming Languages and Systems*, vol. 15, no. 4, pp. 706–734, 1993.

[2] Dimitar P. Guelev, Mark D. Ryan, and Pierre-Yves Schobbens, "Model-checking Access Control Policies," in *Seventh Information Security Conference (ISC'04). Lecture Notes in Computer Science*. 2004, Springer-Verlag.

[3] Mark D. Ryan Nan Zhang and Dimitar Guelev, "Evaluating Access Control Policies Through Model Checking," in *Eighth Information Security Conference (ISC'05). Lecture Notes in Computer Science*. 2005, Springer-Verlag.

[4] Jon Barwise, Ed., *Handbook of Mathematical Logic*, North Holland, 1977.

[5] Nan Zhang, Mark Ryan, and Dimitar P. Guelev, "Synthesising Verified Access Control Systems in XACML," in *LNCS, 2nd ACM Workshop on Formal Methods in Security Engineering*. 2004, Springer-Verlag.

[6] Nan Zhang, Mark Ryan, and Dimitar P. Guelev, "Synthesising Verified Access Control Systems through Model Checking," *Journal of Computer Security*, vol. 16, no. 1, pp. 1–61, 2007.

[7] Simon Godik and Tim Moses, "eXtensible 2 Access Control Markup 3 Language (XACML) Version 1.0," Tech. Rep., OASIS, 2003.

[8] Intel, "The XACML Enabled Gateway - The Entrance to a New SOA Ecosystem," Tech. Rep., Intel Corporation, 2009.

[9] OASIS (Organization for the Advancement of Structured Information Standards), "OASIS eXtensible Access Control Markup Language (XACML) TC," 2010.

[10] S. C. Kleene, *Introduction to Metamathematics*, D. van Nostrand, 1952.

[11] Elliot Mendelson, *Introduction to Mathematical Logic*, Van Nostrand, Princeton, New Jersey, 1964.

[12] R. G. Addie, "Certified Documents and their Interchange," Tech. Rep. SC-MC-2002, University of Southern Queensland, Department of Mathematics and Computing, 2010.

[13] Samuel Alexander Moffatt, "Access control in semantic information systems," Masters dissertation, University of Southern Queensland, 2010.

[14] Kurt Opsahl, "Facebook's Eroding Privacy Policy: A Timeline," April 2010.

[15] Matt McKeon, "The Evolution of Privacy on Facebook," April 2010.

[16] Paul Adams, "The Real Life Social Network," July 2010.