Managing Trust in an Information-Labeling System

M. Blaze J. Feigenbaum P. Resnick
AT&T Laboratories
Murray Hill, NJ 07974
{mab,jf,presnick}@research.att.com

M. Strauss*
Iowa State University
Ames, IA 50010
mstrauss@cs.iastate.edu

November 4, 1996

Abstract

We address the problem of *trust management in information labeling*. The Platform for Internet Content Selection (PICS), proposed by Resnick and Miller [12], establishes a flexible way to label documents according to various aspects of their contents, thus permitting a large and diverse group of potential viewers to make (automated) informed judgments about whether or not to view them. For some viewers, the relevant aspects may be quantity or quality of material in certain topical areas, and, for others, they may be the presence or absence of potentially offensive language or images. Thus PICS users need a language in which to specify their *PICS profiles*, *i.e.*, the aspects according to which they want documents to be labeled, the acceptable values of those labels, and the parties whom they trust to do the labeling. Furthermore, PICS-compliant client software (*e.g.*, a web browser) needs a mechanism for checking whether a document meets the requirements set forth in a viewer's profile. A trust management solution for the PICS information-labeling system must provide both a language for specifying profiles and a mechanism for checking whether a document meets the requirements given in a profile.

This paper describes our design and implementation of a PICS profile language and our experience integrating the *PolicyMaker* trust management engine with a PICS-compliant browser to provide a checking mechanism. PolicyMaker was originally designed to address trust management problems in network services that process signed requests for action and use public-key cryptography [2]. Because information labeling is not inherently a cryptographically based service, and thus is outside the original scope of the PolicyMaker framework, our work on information labeling is evidence of PolicyMaker's power and adaptability.

1 Introduction

Rapid growth in the Internet has focused attention on a problem common to every medium that serves large and diverse audiences: Not all material is suitable for all audience members. Traditionally, broadcast media such as television and radio have been subject to more restrictions than print media, for exactly this reason. The PICS information-labeling system [12] provides a flexible approach to filtering information at the point of reception rather than at the point of distribution, thus holding out the possibility of avoiding government censorship in the process of controlling access to information on the Internet. The success of PICS (Platform for Internet Content Selection) as an approach to access control requires a mechanism for trust management.

*Work done as a consultant to the Network Services Research Center of AT&T Laboratories.

1.1 What is the Trust-Management Problem in PICS?

The PICS approach stipulates that documents will have labels that describe relevant aspects of their contents; the labels will either be embedded in the documents or available from a separate label server. For example, the RSAC (Recreational Software Advisory Council) scheme assigns four numbers to a document, in an attempt to indicate how much sex, nudity, violence, or potentially offensive language the document contains. Other schemes may label documents according to entirely different criteria, *e.g.*, whether they contain material in specific topical areas. PICS-compliant client software will examine the label(s) on a document and decide whether the document satisfies all of the requirements specified in the local "PICS profile" or access policy. For example, a profile might state "Viewing is allowed if the document is labeled two or less on the violence scale, and the label is certified as accurate by GoodHousekeeping." Crucial trust management questions thus include "how are access policies expressed?" and "whom does a given recipient trust to label documents?"

PICS is intended to be simple and popular – cryptographic security is built in as a rarely-used option. The relatively low emphasis on cryptography, however, does not diminish the trust management problem in PICS. Rather, it provides non-traditional challenges for a general trust management engine like PolicyMaker.

Currently many documents are unlabeled. A user would want to specify what action to take if a document is unlabeled by his favorite rater or in his favorite rating scheme. For example, the user may want to consult rater A only when rater B, whom he prefers, has not rated a particular document, or he may want to consult two raters and allow a document only when both raters give appropriate ratings. Situations in which most access control decisions are made in the absence of primary data are quite atypical in the security literature, and none of these strategies for dealing with unlabeled documents necessarily corresponds to a known, coherent security policy. Nonetheless, because they are popular, natural, and perhaps even appropriate for some classes of users, we must create a profile language that accommodates such strategies, and in any case they present an interesting trust management challenge.

This paper shows how to use the PolicyMaker system [2] to solve the trust management problem in PICS. Although PolicyMaker was originally designed to address trust management problems in network services that process signed requests for action and use public-key cryptography, it is applicable *mutatis mutandis* to the trust management problem for information-labeling. The question "is the key used to sign this request authorized to take this action?" corresponds to "do the labels on this document satisfy the viewing requirements of this viewer?" Similarly, the local policy that "I trust this certifying authority to authorize keys for this category of actions" is the analog of "I trust this rating authority to label documents in this category." This unforeseen use of the PolicyMaker framework is evidence of the framework's power and adaptability.

1.2 Why PolicyMaker?

Policymaker, as explained in Section 3 below, is a very expressive and general trust management system. It is possible to implement trust management for information-labeling schemes such as PICS with less expressive mechanisms. However, the generality of PolicyMaker has a number of important benefits, particularly because the Internet community's understanding of the manner in which PICS labels are used and the kinds of applications that will use them is still evolving.

First, because PolicyMaker is specified (and implemented) as a separate "service," and because its basic function does not depend explicitly on the exact requirements of any particular application, a single PolicyMaker engine can serve many different applications, whether PICS-based or not. This has important practical consequences. From an engineering perspective, it is very easy to integrate

PolicyMaker with the application, since its function is defined in terms of a narrow "query engine" interface. There is no need to re-code the interpreter for each application. More importantly, because the semantics of PolicyMaker queries are unambiguous, and defined by a standard reference implementation, it is much easier to assure that the answer returned for any given query depends only on the policies and credentials associated with the content, and not on the implicit decisions (or bugs) in the implementation itself. As PICS credentials become more complex, the issue of assuring correctness will become especially important, and modularity of function with a clean separation between the role of credentials and the role of the interpreter will make further development more manageable.

One of the most important features of PICS is the ability to *delegate* the authority to issue various labels or permission to view them. While the simplest case (unconditional delegation) is easily handled by almost any trust management mechanism, delegation can quickly become more complex, as will be seen in Section 4 below. All of the ways in which conditional delegation will ultimately be used are not yet known, and so it is premature at best to design or require the use of a restrictive, special-purpose language for describing delegation. PolicyMaker, because it is based on general programs, allows delegation to be described in terms of any computable function. Similarly, digital signatures and other cryptographic functions may become much more important to PICS users as PICS evolves and the contexts in which it is used proliferate. The use of PolicyMaker as a trust management engine ensures that accommodating increased use of cryptography will be straightforward and will not necessitate a redesign of the PICS profile language.

The next two sections of this paper provide overviews of the PICS information-labeling system and the PolicyMaker trust management system, respectively. In Section 4, we describe our PICS profile language and our experience integrating PolicyMaker with a PICS-compliant browser. In Section 5, we conclude that our experience supports the view that trust management is an important component of network services, including services that do not involve cryptography in an apparent and essential way.

2 Review of PICS

This section provides a brief review, excerpted and adapted from [12], of the PICS information-labeling system put forth by Resnick and Miller. Readers already familiar with [12] should skip to the next section.

2.1 The PICS Approach

PICS, the Platform for Internet Content Selection, establishes Internet conventions for label formats and distribution methods, while dictating neither a labeling vocabulary nor who should pay attention to which labels. It is analogous to specifying where on a package a label should appear, and in what font it should be printed, without specifying what it should say.

The main goal of PICS is to allow users to define for themselves what constitutes appropriate content. Parents may not wish to expose their children to sexual or violent images. Businesses may want to prevent their employees from visiting recreational sites during hours of peak network usage. Governments may want to restrict reception of materials that are legal in other countries but not in their own. Thus appropriateness depends on at least three things:

- The supervisor: Parenting styles differ, as do philosophies of management and government.

- The recipient: What is appropriate for one 15-year-old may not be for an eight-year-old, or even all 15-year-olds.
- The context: A game or chat room that is appropriate to access at home may be inappropriate at work or school.

The basic PICS idea is to interpose selection software between the recipient and the on-line documents. The software checks labels to determine whether to permit access to particular materials. It may permit access for some users but not others, or at some times but not others. Any PICS-compliant selection software can process any PICS-compliant label. A single site or document may have many labels, provided by different organizations. Consumers choose their selection software and their label sources (called *rating services*) independently. This separation allows both markets to flourish: Companies seeking to avoid "value judgments" can offer selection software without providing any labels; organizations that wish to make value judgments can, without writing software, create rating services that provide labels.

PICS labels describe content along one or more dimensions. It is the selection software, not the labels themselves, that determine whether access will be permitted or prohibited. Each rating service can choose its own labeling vocabulary. Information publishers can "self-label," just as manufacturers of children's toys currently label products with text such as, "Fun for ages 5 and up." When publishers are unwilling to participate, or cannot be trusted to participate honestly, independent organizations can provide third-party labels. For example, the Simon Wiesenthal Center, which is concerned about Nazi propaganda and other hate speech, could label materials that are historically inaccurate or promote hate.

2.2 Elements of PICS Specifications

There are two PICS specification documents [8, 10]. The most important components are:

- A syntax for describing a rating service, which allows computer programs to present the service and its labels to users.
- A syntax for labels, which allows computer programs to process them. A label describes either a single document or a group of documents (*e.g.*, a site.) A label may be digitally signed and may include a cryptographic hash of the associated document.
- An embedding of labels (actually, lists of labels) into the RFC-822 transmission format [3] and the HTML document format [5].
- An extension of the HTTP protocol [6], which allows clients to request that labels be transmitted with a document.
- A query-syntax for an on-line database of labels (called a *label bureau*).

We now give an example of a PICS rating service. The initial section includes general information about the service. The second section describes each of the dimensions, or categories, and the scales used for each. In this case, there are three dimensions, indicating the number of sax and violins recorded or depicted, and the presence of bass materials.

```
((PICS-version 1.1)
 (rating-system "http://www.musac.org/Ratings/Description/")
 (rating-service "http://www.musac.org/ratingsv01.html")
```

```
 (name "Musac")

 (category
  (transmit-as "v")
  (name "Violins"))
 (category
  (transmit-as "s")
  (name "Sax"))
 (category
  (transmit-as "b")
  (name "Bass")
  (label
   (name "None")
    (description "Does not appeal to bass or prurient interest")
     (value 0))
  (label
   (name "Some")
    (description "May appeal to bass or prurient interest")
     (value 1))))
```

Next, we give a sample PICS label (actually a label list containing just one label). The URL on the first line, which identifies the labeling service, makes it possible to redistribute labels yet still identify their original sources. The label can also include information about itself, such as the date on which it was created, the date it will expire, the fact that the label is associated with a certain document (in this case, "http://www.gcf.org/stuff.html"), and the label author. The last line shows the attributes that describe the document: a "bass" value of 1; "sax" 2; and "violins" 0. The service description allows client software to make the connection between the transmit-name "b" and the name "bass" and to interpret the value 1 as an indication of bass content.

```
(PICS-1.1 "http://www.musac.org/ratingsv01.html"
       labels on "1995.11.05T08:15-0500"
              until "1996.12.31T23:59-0000"
              for "http://www.gcf.org/stuff.html"
              by "John Doe"
              ratings (b 1 s 2 v 0))
```

Anything that can be named by a URL can be labeled, including documents that are accessed via FTP, gopher, or Netnews, as well as HTTP. PICS also proposes a URL naming system for IRC, so that chat rooms with stable topics can be labeled.

Finally, the PICS specifications give three ways to distribute labels. The first is to embed labels in HTML documents, using the META element in the document header. The second requires a client to ask an HTTP server to send labels along with the documents it requests. The third is to use a label bureau, whose sole purpose is to dispense labels and who may handle labels created by one or more services. Separation of label distribution from content distribution facilitates third-party labeling even when the content providers do not want the labels distributed. For example, the Simon Wiesenthal Center can label anti-semitic propaganda without the cooperation of neo-Nazi content providers.

3 Review of PolicyMaker

This section provides a brief review, excerpted and adapted from [2] and [9], of the trust management approach of Blaze, Feigenbaum, and Lacy. Readers already familiar with [2] should skip to the next section.

3.1 Basic Elements of PolicyMaker

The goal of trust management is to evaluate whether a particular *query* or proposed action is consistent with local policy. In the PolicyMaker system, the action under consideration is represented by an *action string*. The trust assumptions are divided into a *policy,* which is under local control, and *credentials,* which generally represent statements by others. The PolicyMaker interpreter accepts as input the local policy and received credentials (collectively called *assertions*) and an action string. Depending on the credentials and form of the query, it can either return a simple yes/no answer or additional credentials that would make the proposed action acceptable. The PolicyMaker interpreter can either be built into applications (through a linked library) or run as a separate "daemon" service.

An essential feature of PolicyMaker is the separation of generic trust management mechanism (provided by PolicyMaker) from application-specific policy (which is defined by each application). Action strings are application-specific strings, bound to a list of one or more *requester-ids*. Policies and credentials are programs, written in a "safe" (I/O and resource-limited) language, that examine the binding between an action string and one or more requester-ids. The PolicyMaker mechanism for checking that the action string and credentials conform to local policy does not depend on the application-specific semantics of the action-strings, credentials, and policies; indeed these semantics are not even known to the PolicyMaker interpreter.

A credential is a statement giving a criterion for what action strings its *authorizer-id* accepts. Credentials take as input an action string bound to a list of requester-ids and authorizer-ids with which it is associated. If the credential *trusts* the IDs to perform the action encoded in the action string, it can *accept* the action by adding its own authorizer-id to the list. As a simple example, an *unconditional delegation credential* states that its authorizer-id will accept all action strings that are accepted by another specific authorizer-id.

PolicyMaker credentials thus differ in two important ways from application-specific notions of "credential" such as PGP signed keys [13] or PICS labels. First, application-specific "credentials" may be digitally signed, in which case they are usually called "certificates," whereas PolicyMaker credentials are simple statements of the form "A allows B." These statements may be concerned with authorization of cryptographic keys to request actions, *i.e.*, they may be the usual type of statements signed by the issuers of "certificates," or they may be concerned with something entirely noncryptographic. Second, credentials are written in the PolicyMaker syntax, while applications (including PICS) usually require their own syntax. Consequently, the use of the PolicyMaker interpreter as a trust management mechanism requires an application to *translate* its own "credentials" into PolicyMaker credentials. This translation consists of verifying signatures, in the case of certificates, to make sure that they really correspond to authorizer-ids, and reformatting in PolicyMaker credential syntax.

PolicyMaker policies are analogous to credentials in that both are assertions of trust. In fact, policies are implemented as a special case of credentials, in which the authorizer-id is the keyword "POLICY." Action strings accepted by POLICY are considered to be locally acceptable; thus, credentials generated from translated, application-specific "credentials" can never be given this authorizer-id. Policies are locally generated and trusted and form the "trust root" under which all queries are evaluated.

Assertions (*i.e.*, both credentials and policies) can also modify the action strings that they accept. *Annotations* allow credentials to add additional information to the action strings they accept, rather than being limited to the boolean choice of accepting the action string or not. As such, annotations can be thought of as a mechanism for communication between assertions, as well as communication between the application and the credentials. Such a mechanism is required, because the ultimate question to be answered is whether the *entire* set of assertions that is fed to the trust management engine accepts the requested action, not whether a single assertion accepts it. One assertion may need to communicate to a second that the requested action meets one necessary condition, and the second assertion may use this information in the process of checking whether the request meets a second necessary condition.

3.2 Signature Schemes and Filter Languages

PolicyMaker does not itself verify signatures on signed assertions or queries or even process the original signed messages. Instead, signatures are verified by the calling application or, more typically, by some external program such as PGP or PEM that the calling application chooses for this task. The external program guarantees that the signature was valid for the identified public key. The public key passed to the PolicyMaker interpreter identifies the program and the key (*e.g.*, "`PGP:0x01234567abcdefa0b1c2d3e4f5a6b7`"). By not interpreting signatures itself or insisting on a particular signature scheme or format, PolicyMaker makes it very easy to implement a certification authority that exploits existing infrastructure. For example, it is possible to have trust structures that consist of a mixture of X.509 certificates [7], interpreted by a program that converts them into PolicyMaker credentials, and certificates consisting of simple text messages signed with a program such as PGP.

In the PolicyMaker assertion "A allows B," A is a source of authority, and B is what A authorizes. B is expressed as a *filter*, *i.e.*, an interpreted program run on the action string within a safe wrapper. The PolicyMaker interpreter currently supports three filter languages: a regular expression system (similar to those used in Unix), a safe version of AWK [1], called AWKWARD, and a macro language that preprocesses into AWKWARD. Other "safe" languages, such as Java [4] or Safe-TCL [11], are easily added as desired.

In general, any language that can be safely interpreted can be used as a filter language. A distinguishing feature of PolicyMaker is that filters are allowed the full complexity and expressiveness of general programs. Designing and implementing a safe filter language is a much simpler task than designing a general-purpose language for remote agents (like Java), however, because filters generally have no need to issue "dangerous" system calls. For example, they do not need to open files or interact with the network. PolicyMaker wraps the filter-language interpreter in a resource-limited shell that prevents, *e.g.*, infinite loops in filters from consuming the entire host CPU. Most filters can be assumed to be very simple, and so their resource allocation can be modest.

Input to filters consists of the current action string and an "environment" containing information about the current context (*e.g.*, date, time, path length, application name, etc.). A filter can use the environment to enforce contextual constraints such as expiration times. A filter also has access to information about the rest of the trust structure in which it is being evaluated, which makes it possible to design assertions that limit the degree to which their authority can be deferred.

Although the interpreter for a filter language is external to PolicyMaker itself, the name of the language is given in assertions and must be known by anyone who needs to use the assertion.

New languages can be added easily as needed, provided that all recipients of assertions using the new language are configured to interpret it. PolicyMaker ignores assertions written in unknown or unsupported filter languages. Because the basic PolicyMaker model is *monotonic*, ignoring such assertions cannot cause the acceptance of an action that would otherwise be rejected, although it may cause the rejection of an otherwise acceptable action.

3.3 Summary

To integrate PolicyMaker with an application, the basic approach of Blaze, Feigenbaum, and Lacy [2] thus requires one to construct translators for the application's actions and credentials. Generally, the following sequence of steps is taken for each request:

1. Generate an action string for the action to be considered.

2. Find the application-specific "credentials" that purport to justify the action string.

3. Translate these application-specific "credentials" into PolicyMaker credentials, checking signatures if necessary.

4. Present the action string, the policy, and the credentials to the PolicyMaker interpreter.

5. Use the result of the PolicyMaker evaluation to perform or not perform the action.

6. Return to the user annotations that would make a rejected action string acceptable, if PolicyMaker supplied any.

4 Using PolicyMaker to Manage Trust in PICS

In the PICS application, an internet client generates an action string, and a "PolicyMaker interface" performs translations, fetches credentials, feeds the PolicyMaker interpreter, evaluates the results, and returns a *block or allow* result to the internet client. We flesh out our implementation of these steps in this section.

4.1 General Considerations in PICS Profiles

We start by considering several sets of user preferences and what sort of policies and credentials express them. Later we will see how to translate them into PolicyMaker.

Example 1. Typical

We begin with a typical current policy that can be implemented easily, using Microsoft's visual interface in Explorer 3.0. This is followed by a typical (matching) credential, easily expressed as a PICS label.

Suppose that, in practice, it turns out that almost everyone is honest in the labels that he or she provides. A trusting parent sets the policy that any document labeled with low levels of s and v is acceptable for his children to view.

- Policy: Fetch labels embedded in the document or from label bureau `http://labels.com/`. Allow viewing if a label using the `http://musac.com/` rating service rates s less than 3 and v less than 4.

- Credential: George, using service `http://musac.com/` states: `s=2` and `v=1`.

Example 2. Simple Deferral

On the other hand, it may turn out that not everyone is quite so honest in assessing the sights and sounds on the Internet. In particular, it seems that musac producers tend to underreport the number of violins used in their recordings and even that some independent reviewers seem to show bad judgment on this score. The parent, however, has found that GoodMouseClicking is a good judge of the independent reviewers. The parent sets a policy of trusting reviews from those reviewers for which GoodMouseClicking vouches.

- Policy: Ignore embedded labels and fetch labels from label bureau `http://labels.com/` Allow viewing if a label using the `http://musac.com/` rating service indicates s is less than 3 and v is less than 4 and GoodMouseClicking vouches for the label's veracity.
- Credentials:
  - George, using service `http://musac.com/` states: `s=2` and `v=1`.
  - GoodMouseClicking states: George is a trustworthy labeler

Example 3. Conditional Deferral

The musac rating service is not the only one available. In fact, all kinds of assertions about URLs can be made. In particular, there is a "CodeSigning" service used to make assertions that downloadable software has been virus-checked, Java applets are well-behaved, or the code uses only a certain amount of memory. GoodMouseClicking has discovered that some reviewers are more reliable in their judgments on the musac scales than on the CodeSigning scales. It does not wish to disqualify these reviewers entirely but merely to limit its endorsement of these reviewers to the musac rating service.

In this case, the following credentials are *not* sufficient to authorize downloading the document in question.

- Policy: Ignore embedded labels and fetch labels from label bureau `http://labels.com/` Allow downloading if a label using the `http://CodeSigning.com/` rating service indicates `Memory_required < 4,000,000` and GoodMouseClicking vouches for the label's veracity.
- Credentials:
  - George, using service `http://musac.com/` states: `s=2` and `v=1`.
  - George, using `http://CodeSigning.com/` states: `Memory_required = 1,000,000`.
  - GoodMouseClicking states: George is trustworthy to label content using the rating service `http://musac.com/`.

Besides the situations discussed above, the PICS domain allows users to express a preference about how to handle missing labels, which is important because many documents currently have no labels. To express these sorts of preferences, PICS uses three-valued logic with values *allow, block, unrated.* Primitive preferences are exemplified by

- Block if Alice says the sax content is at least 2
- Block if the publisher says the adult themes content is greater than 5
- Allow `http://www.whitehouse.gov/` (unconditionally)

A user would combine these using the three-valued combinators *any-allows* and *any-blocks* or with the two-valued combinators *all-allow* and *all-block*. For example, in the draft version of the PICS profiles language,

```
(any-blocks s 2:INF v 3:INF)
```

returns *block* if the rater under consideration supplies either a high sax or a high violins rating, returns *allow* if all supplied values are low, and returns *unrated* if no values are supplied. Instead, one could write

```
(all-allow s -INF:1 v -INF:2)
```

using the two-valued *all-allow*. This returns *allow* if both the sax and violins values are supplied and low and otherwise returns *block*. Since sax values are integers, these two examples differ only on how to handle missing labels. A profile author can use the combinators *any-allows*, *any-blocks*, *all-allow*, *all-block* and the unconditional *allow* or *block* to express a rich policy.

Profiles that use three-valued combinators, although in some ways natural and useful, suffer from an inherent problem. Given a particular viewing request and set of credentials, they may return *unrated* rather than *allow* or *block*. We believe strongly that the question of whether a given set of credentials authorizes a given viewing request should depend only on the profile and, specifically, should not depend on how a particular browser interprets an answer of *unrated* returned by the trust management engine. To achieve this, one should follow the "golden rule" of PICS profile design. Profiles should be "unambiguous": They can use the value *unrated* in their reasoning, but they should not return *unrated* as a final answer. A profile author can satisfy the golden rule by ultimately enclosing each *any-* rule within a two-valued *all-* rule, thus ensuring that the top-level return value is either *allow* or *block*. In particular, by restricting oneself to the *all-allow* and *any-allow* combinators, one can use traditional two-valued logic and write monotonic profiles – profiles that do not change from *allow* to *block* on the basis of additional credentials. A non-monotonic policy that says "block if there's a high sax rating" is translated into the monotonic policy "allow if there's a low sax rating or the document has no embedded labels."

Signatures are currently rare in PICS objects. In some applications, it is convenient to refer to principals only by their public key, but in the PICS application that would mean a lot of keys of the form `unknown_key1`, `unknown_key2`. PolicyMaker is flexible enough to let us refer to principals by any unique name (*e.g.*, a URL) and still conveniently handle the extra security conveyed in an occasional signed credential.

4.2 Integrating PICS and PolicyMaker

This section describes how PICS labels and browser requests are translated so that PolicyMaker can interpret them. Other PolicyMaker credentials are written directly in native PolicyMaker.

A browser will make a request to display a document named by a URL. The trust management job is to decide whether the browser should be allowed to display on the basis of the URL itself, PICS content labels for the URL, credentials that defer trust, and arbitrary locally-determinable conditions. For example, a document may be allowed if it is `http://www.whitehouse.gov/`, if it has a signed, unexpired content label specifying a low violins level, if Alice says it is ok, or if it is after 5pm and the machine is not too busy. In contrast with many other trust management problems, the question of whether a cryptographic key is authorized to sign a particular statement is currently not often an issue.

The browser makes its request to an interface to the PolicyMaker interpreter. The interface is responsible for gathering PICS content labels and other credentials, translating them into native PolicyMaker, and calling the PolicyMaker interpreter.

A typical action string would be:

```
view: http://www.greatdocs.com/foo.html
```

This action string will be augmented with annotations in a `name: value` format. There are commands written in AWKWARD for referring to the value associated with a name and modifying a `name: value` pair. Typical examples of these functions will be used below and explained as needed. A typical name value pair will look like

```
'George http://musac s': 2
```

to indicate "George says the s content is 2 (where the semantics of s can be found at http://musac)." The name itself may contain additional components:

```
'George stale http://musac s': 2
```

means "George once said the s content is 2" (but the credential has expired). Normally the name is everything from the beginning of line to the first colon; a singly-quoted name can contain a colon.

Credentials can come from a secure environment on the local machine (for example, to assert the activity level of the machine, or, if it can be done securely on the local machine, the time). Integrating such credentials with PolicyMaker amounts simply to writing a local program that produces PolicyMaker credentials directly. For example, suppose we trust the local machine to say what time it is. Consider the following shell script:

```
echo localdate ALLOWS
echo {
echo '   annotate_override( "date",' \
         \"`date -u +%Y.%m%.%dT%H:%M-0000`\" ')'
echo '   accept()'
echo }
```

The above produced the following PolicyMaker credential on August 29th:

```
localdate ALLOWS
{
   annotate_override( "date", "1996.08.29T12:33-0000" )
   accept()
}
```

The date, in UTC, is produced in the same format as PICS labels. This format allows for comparison of dates as strings. The `annotate_override` makes sure that `date: 1996.08.29T12:33-0000` is the only date string appearing in the action string. Some of these credentials need to be changed when the user changes a policy (for example, this date credential could be added the first time the user enters a policy that needs the date). It is therefore convenient to produce these credentials by shell scripts. Permanent additions may be compiled directly into the PolicyMaker interface.

The primary credential is a PICS label. A PICS label in the musac rating service, for example, might look like

```
(PICS-1.1 "http://www.musac.org/" labels
 exp "1997.12.31T23:59-0000"
 for "http://www.greatdocs.com/foo.html"
 by "George"
 ratings (s 2 v 1 b 1))
```

(An explanation of this label appears in Section 2.2.)

This would be translated into the following PolicyMaker credential:

```
"George" ALLOWS
{
  if( lookup( "date") > "1997.12.31T23:59-0000")
    staleness = "stale "
  else
    staleness = ""
  if( lookup( "view") == "http://www.greatdocs.com/foo.html"  &&
      annotate_require( "'George " staleness "http://musac " "s'", 2) &&
      annotate_require( "'George " staleness "http://musac " "v'", 1) &&
      annotate_require( "'George " staleness "http://musac " "b'", 1))
    accept()
}
```

(Here, for clarity of presentation only, `http://www.musac.org/` has been shortened.)

In the PICS domain, it is possible that a user would want to view a document with an expired credential but not to view a document with no credentials. This particular label will generate annotations like

```
'George http://musac s': 2
```

because its expiration date has not yet past; if it were stale the annotation would read

```
'George stale http://musac s': 2
```

This uses the function `annotate_require`, written in AWKWARD, that does the following: If the sax annotation is missing, supply it and accept; if *this* s annotation is already present, accept; if a *different* s annotation is present, reject. Note that "http://www.musac.org/ s" uniquely specifies the dimension along which the URL is being rated, and George should supply at most one unexpired rating for this URL's "http://www.musac.org/ s" content. The above assumes George only issued one label, which may be stale; it is also possible to plan for multiple stale labels and keep only the most recent.

A PICS label may (occasionally) arrive signed. In that case, the signature is checked by the interface, and the veracity of the signature affects the translation, but in no case is the actual signature string part of the translation. If the signature is good, the PolicyMaker credential generated looks like:

```
"George" ALLOWS
{
  if( lookup( "date") > "1995.12.31T23:59-0000")
    staleness = "stale "
  else
```

```
    staleness = ""
  if( lookup( "view") == "http://www.greatdocs.com/foo.html"  &&
      annotate_require( "'George signed " staleness "http://musac " "s'", 2) &&
         ...)
    accept()
}
```

If the signature is bad, the word `forged` is substituted for `signed`.

The source of the PICS label is not part of the label itself but is known and meaningful to us. Labels may be embedded directly in a document or come from an independent label bureau (at the request of the PolicyMaker interface). The PICS domain explicitly allows users to take shortcuts and use heuristics for security, in the name of simplicity. For example, a user may want to trust unsigned labels from an independent label bureau but not trust embedded labels at all. The source of the label, known to the PolicyMaker interface, affects the translation of a label into PolicyMaker credentials in a way similar to signatures and expiration dates. It was omitted above for brevity.

The PICS profile language and the language for credentials other than simple labels is now under development in the PICS community. When these languages are standardized, we will write translators to convert these credentials and profiles into native PolicyMaker. For the time being, our PolicyMaker interface accepts native PolicyMaker credentials and policies and feeds them directly to the interpreter.

We now revisit the preferences mentioned above and discuss the way they would be translated into PolicyMaker.

Example 1. Typical

- Policy: Fetch labels embedded in the document or from label bureau `http://labels.com/`. Allow viewing if a label using the `http://musac.com/` rating service rates s less than 3 and v less than 4.
- Credential: George, using service `http://musac.com/` states: `s=2` and `v=1`.

This policy would be translated into PolicyMaker as

```
POLICY ALLOWS
{
 if(( lookup_regexp(
      ".* (embedded|http://labels\.com/) .* http://www\.musac\.org/ s")
         < 3) &&
    ( lookup_regexp(
      ".* (embedded|http://labels\.com/) .* http://www\.musac\.org/ v")
         < 4))
     accept()
}
```

Example 2. Simple Deferral

- Policy: Ignore embedded labels and fetch labels from label bureau `http://labels.com/` Allow viewing if a label using the `http://musac.com/` rating service indicates s is less than 3 and v is less than 4 and GoodMouseClicking vouches for the labeler's veracity.

- Credentials:
  - George, using service `http://musac.com/` states: `s=2` and `v=1`.
  - GoodMouseClicking states: George is a trustworthy labeler

The GoodMouseClicking assertion would be translated into PolicyMaker as

```
GoodMouseClicking ALLOWS
{
   component_override( "'George .*'", endorsed)
   accept()
}
```

The `component_override` function, written in AWKWARD, puts the word "endorsed" into the action string in its proper place.

The policy would be translated into PolicyMaker as

```
POLICY ALLOWS
{
   check_auth( "GoodMouseClicking");
   if(( lookup_regexp(
           ".* endorsed .* http://labels\.com/ http://www\.musac\.org/ s")
         < 3) &&
      ( lookup_regexp(
           ".* endorsed .* http://labels\.com/ http://www\.musac\.org/ v")
         < 4))
     atleast(1)
}
```

This is looking for a line that was built up in stages. When the PolicyMaker interface fetches a label from `http://labels.com/`, it arranges that `http://labels.com/` be tacked onto the action string in the appropriate place. The rating service MUSAC and the dimension sax come from the PICS content label (as does the irrelevant name of the rater that here matches `.*`). The word "endorsed" was supplied by GoodMouseClicking after it checked the name of the rater.

If we were to consider more than one endorser, then GoodMouseClicking would have to annotate with a unique version of the word "endorsed" to avoid ambiguity. Similarly, if we cared about a cryptographic signature from GoodMouseClicking, our PolicyMaker interface would check a signature on receipt of the GoodMouseClicking credential and substitute "endorsed-with-signature" for "endorsed."

Example 3. Conditional Deferral

Recall that, in this case, the credentials do not authorize downloading the document in question.

- Policy: Ignore embedded labels and fetch labels from label bureau `http://labels.com/` Allow downloading if a label using the `http://CodeSigning.com/` rating service indicates `Memory_required < 4,000,000` and GoodMouseClicking vouches for the label's veracity.

- Credentials:
  - George, using service `http://musac.com/` states: `s=2` and `v=1`.

– George, using `http://CodeSigning.com/` states: `Memory_required = 1,000,000`.

– GoodMouseClicking states: George is trustworthy to label content using the rating service `http://musac.com/`.

The GoodMouseClicking assertion would be translated into PolicyMaker as

```
GoodMouseClicking ALLOWS
{
     component_require( "'George .* http://www\.musac\.org/ .*'",
                         endorsed)
     accept()
}
```

Our policy would be translated as

```
POLICY ALLOWS
{
   check_auth( "GoodMouseClicking");
   if( lookup_regexp( ".* endorsed http://www\.CodeSigning\.com/ Memory_required)
       <= 4,000,000)
     atleast(1)
}
```

The functions `check_auth` and `atleast(1)` insist on the GoodMouseClicking signature. The `lookup_regexp` checks for the word "endorsed" and the proper rating name and value.

5 Conclusions

Trust management is most commonly associated with cryptography-based applications. Typically, the goal is to establish that one or more cryptographic keys are authorized for certain actions. A very basic example is that in which A wants to communicate privately with B. In the absence of a secure communication link to B, A must trust some intermediary with whom she does have a secure communication link to provide the needed cryptographic key and vouch that it is authorized for private communication with B. Trust management systems provide a mechanism for expressing the conditions under which one defers trust to others.

There are many other situations, however, in which trust management is needed because one party lacks direct experience and needs to defer trust to others who do. PICS highlights one such situation: A person controlling a host computer may prefer that it not download or display certain materials (*e.g.*, viruses, sexually explicit pictures, or long boring texts). It is necessary to trust the statements of others who have had experience with those materials, and it is often useful to defer trust to someone else (*e.g.*, GoodMouseClicking) who vouches for the statements of others.

There is a natural connection between these two types of trust management. If someone expresses trust in certain kinds of statements by GoodMouseClicking, he or she might want to check the authenticity of any statements that purportedly come from GoodMouseClicking. One way to do so is through digital signatures, which necessitates a trust management mechanism for deciding whether a particular key really belongs to GoodMouseClicking.

Despite the natural connection, however, it is important to distinguish the need for authentication from the need for trust. There may be reasons other than signature verification to trust the authenticity of statements, such as a distribution method that is expensive to subvert, an expectation that no one is likely to forge statements, or a judgment that even if a forgery is accepted, the costs are not high. Moreover, there may be good reasons to distrust authenticated statements: A neo-Nazi can sign a statement that his propaganda is historically accurate, but most people will choose not to believe him.

The PolicyMaker trust management framework, although designed for applications based on authorization of cryptographic keys, is general enough to handle other trust management applications. PICS labels and policies can be translated into the PolicyMaker framework. If desired, the policies can demand signatures on labels for authentication purposes, but the rules for trusting the authenticity of labels are distinct from the rules for trusting their contents. Once the notion of trust is separated from the notion of cryptographic authorization, it is easy to envision a wealth of other trust management applications, including delegation of authority in business settings, evaluation of controversial political policies, setting conditions under which the owner of a piece of intellectual property permits downloads, and setting rules for disclosure of medical information. The success in adapting the PolicyMaker framework to the PICS application augurs well for its adaptability to other trust management applications.

6 Acknowledgements

It is a pleasure to thank Lewis McCarthy, Rebecca Wright, and Jack Lacy for their comments on an earlier draft of this paper.

References

[1] A. V. Aho, B. W. Kernighan, and P. J. Weinberger, *The AWK Programming Language*, Addison-Wesley, Reading, 1988.

[2] M. Blaze, J. Feigenbaum, and J. Lacy, *Decentralized Trust Management*, in Proceedings of the Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, 1996, pp. 164–173.

[3] D. Crocker, *RFC-822: Standard for the Format of ARPA Internet Text Messages*, `http://ds.internic.net/rfc/rfc822.txt`, August 1982.

[4] J. Gosling and H. McGilton, *The Java Language Environment, A White Paper*, Sun Microsystems, Inc., Mountain View, 1995.

[5] HyperText Markup Language (HTML), `http://www.w3.org/pub/WWW/MarkUp`.

[6] HyperText Transfer Protocol (HTTP), `http://www.w3.org/pub/WWW/Protocols`.

[7] *Information Technology – Open Systems Interconnection – The Directory:Authentication Framework*, Recommendation X.509, ISO/IEC 9594-8.

[8] T. Krauskopf, J. Miller, P. Resnick, and G. W. Treese, *Label Syntax and Communication Protocols*, World Wide Web Consortium, `http://w3.org/PICS/labels.html`, May 1996.

[9] R. Levien, L. McCarthy, and M. Blaze, *Transparent Internet E-mail Security*, submitted for publication.

[10] J. Miller, P. Resnick, and D. Singer, *Rating Services and Rating Systems (and Their Machine Readable Descriptions)*, World Wide Web Consortium, `http://w3.org/PICS/services.html`, May 1996.

[11] J. K. Ousterhout, *TCL and the TK Toolkit*, Addison-Wesley, Reading, 1994.

[12] P. Resnick and J. Miller, *PICS: Internet Access Controls Without Censorship*, Communications of the ACM, October 1996.

[13] P. Zimmermann, *PGP User's Guide*, MIT Press, Cambridge, 1994.