# REFEREE: Trust Management for Web Applications

**Yang-Hua Chu**[1]
**Joan Feigenbaum**[2]
**Brian LaMacchia**[2]
**Paul Resnick**[2]
**Martin Strauss**[2]

[1]MIT/W3C, 545 Technology Square, Cambridge, MA 02139

[2]AT&T Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974

## Abstract

Digital signatures provide a mechanism for guaranteeing integrity and authenticity of Web content but not more general notions of security or trust. Web-aware applications must permit users to state clearly their own security policies and, of course, must provide the cryptographic tools for manipulating digital signatures. This paper describes the REFEREE trust management system for Web applications; REFEREE provides both a general policy-evaluation mechanism for Web clients and servers and a language for specifying trust policies. REFEREE places all trust decisions under explicit policy control; in the REFEREE model, every action, including evaluation of compliance with policy, happens under the control of some policy. That is, REFEREE is a system for writing policies about policies, as well as policies about cryptographic keys, PICS label bureaus, certification authorities, trust delegation, or anything else.

In this paper, we flesh out the need for *trust management* in Web applications, explain the design philosophy of the REFEREE trust management system, and describe a prototype implementation of REFEREE.

## Introduction

Web surfing is a dangerous sport. Downloaded software may contain viruses. Materials that appear to come from one source may in fact be spoofs provided by another source. Eavesdroppers may overhear credit card numbers or other sensitive information. Personal information may be collected legitimately but then used to violate one’s privacy.

One way to approach these problems is to attempt to eliminate dangers. For example, the Java applet interpreter tries to provide an execution environment in which programs can only perform "harmless" actions. The PCC system (for "proof carrying code") requires mobile programs to prove to potential hosts that they are "harmless" [N, NL]. A wealth of cryptographic protocols attempt to eliminate dangers by enforcing secrecy or authenticity.

A complementary approach focuses on trust. To trust is to undertake a potentially dangerous operation *knowing that it is potentially dangerous*. A user might prefer to have *proof* of harmlessness, but weaker forms of evidence may also be sufficient. A recommendation from a close friend may convince someone to trust that a piece of software is virus-free. Someone may trust an insecure channel for transmission of a credit card number if the credit card company assumes liability for any fraudulent uses of the number.

*Credentials* and *policies* are the raw materials for making trust decisions. A credential is a statement, purportedly made by some speaker. A policy determines the conditions under which a particular action is allowed.

Following [BFL], we use the term *trust management* to refer to the problem of deciding whether requested actions, supported by credentials, conform to policies. Just as a database manager coordinates the input and storage of data and processes queries, a trust manager coordinates the collection and storage of policies and credentials and processes requests for trust decisions. We refer to the processing of such a request as "evaluating compliance with a policy."

In current Web applications, the prototypical credential is a PICS label [RM], which states some properties of an Internet resource (e.g., executable code has been virus-checked). An important idea of [BFL] that we use in this work is that of "programmable credentials." Rather than simply making unconditional statements, programmable credentials can examine statements made by other credentials and fetch information from the network before deciding which statements to make. For example, a credential may examine existing statements, ask GoodMouseClicking whether the authors of those statements are reliable, and then decide whether the statements are trustworthy.

In its simplest form, a policy sets rules about which credentials must be present in order to permit an action. In the course of deciding whether a requested action conforms to a policy, however, it is often necessary to perform potentially dangerous operations. In particular, it may be useful to fetch credentials over the network. For example, if one wants to know a particular reviewer’s opinion about a video clip before deciding whether to watch it, it will be necessary to consult a server that distributes that reviewer’s opinions. One danger is that information will leak out unintentionally during the fetching process. Another danger is that spoofed credentials may be retrieved. Yet a third danger is that, since credentials are programs, running credentials retrieved over the network exposes us to the full gamut of risks from running unknown programs.

Because we are interested in managing trust, rather than in eliminating danger, we permit these dangerous actions but put them under policy control. Policies state when to fetch credentials. Policies also determine which statements must be made about a credential before it is safe to run it. For example, credentials written in some (safe) programming languages may always be permitted to execute, while credentials written in other languages may be executed only if vouched for by some trusted party. Finally, policies determine whether and how credentials are to be authenticated before the statements they create are deemed to be trustworthy. For example, some kinds of statements may be used only if the credentials were signed using a particular cryptographic key.

Our trust management system is called REFEREE. The name is an acronym: Rule-controlled Environment For Evaluation of Rules, and Everything Else. It is an environment for evaluating compliance with policies (rules), but the evaluation process itself may involve dangerous actions and hence is under policy control. The "Everything Else" refers to credentials, whose evaluation (execution) also needs to be under policy control. In placing everything under policy control, REFEREE differs from *PolicyMaker*, the trust management system described in [BFL]. In particular, PolicyMaker does not permit policies to control credential-fetching or signature-verification; it assumes that the calling application has gathered all of the relevant credentials and verified all digital signatures before calling the trust management system.

Two working groups of the World Wide Web Consortium, PICS and DSIG, are experimenting with REFEREE as a possible common platform for trust management in content selection and digital signature applications. Yang-hua Chu has implemented an early design of REFEREE as part of his masters' thesis research, and this paper reflects that early design.

We begin by presenting the REFEREE calling conventions for invocable programs (both policies and credentials). Next, we describe a simple language, called the *profiles-0.92* language, for writing policies and describe several other programs, including one that fetches PICS labels and converts them into REFEREE's internal statement format and one that verifies cryptographic hashes. Third, we present sample policies, expressed as REFEREE programs. This is followed by an execution trace of a sample policy that retrieves a set of PICS labels that are sufficient to authorize the requested action. The execution trace is taken from a working policy evaluator that can be accessed and exercised on the Web. Finally, we give a conclusion.

## REFEREE

There are three primitive data types in REFEREE:

1. Programs
2. Statement lists, and
3. Tri-values,

A tri-value is one of *true*, *false*, or *unknown*. A statement list is a collection of assertions expressed in a particular format, described later. Each program takes an initial statement list as an input and may also take additional arguments. A program may invoke another program during the course of its execution.

Intuitively, a policy governing a particular action is a program that returns true or false, depending on whether the available statements are sufficient to infer compliance or non-compliance with a policy, or returns unknown if no inference can be made.

Intuitively, a credential is a program that examines the inital statements passed to it and derives additional statements. This generalizes the usual notion of a credential as directly supplying statements: The new statements supplied by a credential may be conditional on the initial statements or other environmental factors, such as the amount of space available on the local hard disk.

Actually, REFEREE allows both policies and credentials to return both tri-values and statement lists. It is useful for policies to return a justification, which can be expressed as a list of statements, along with a tri-value answer. For example, a policy may reject downloading of code and provide statements indicating whether (a) the code is known to be malicious, or (b) the local machine is currently too heavily loaded to permit downloads. It is also useful for credential programs to indicate whether the execution was successful (a tri-value) in addition to returning a list of statements. Thus, in REFEREE, *every* program returns both a tri-value and a list of statements. However, we will continue to observe the distinction between policies and credentials in this paper, for purposes of exposition.

Applications running on a Web server, a proxy server, or a personal computer may invoke REFEREE. When an application program invokes REFEREE, it provides a database of available programs, provides an initial statement list (that may be empty), designates a particular program (policy) to run, and optionally provides additional arguments to the designated program.

## The Logic of Tri-Values

Notions of "true" and "false" are familiar from Boolean logic. When asking, however, about authorization of a particular action (for example, "Should the following purchase order be approved?"), there are typically three possible outcomes:

1. "Yes, the action may be taken because sufficient credentials exist for the action to be approved."
2. "No, the action may not be taken because sufficient credentials exist to deny the action."
3. "The trust management system was unable to find sufficient credentials either to approve or to deny the requested action."

In the third case, the value returned by the trust management system is neither "true" nor "false" but "unknown," for the policy that was in force during the evaluation was not able to make a determination about the requested action. It is up to the application calling the trust management system to decide what action (if any) should be taken when the trust management system returns "unknown."

An alternative approach using only Boolean values would be to extend the definition of "false" to include requests with insufficient credentials to make a trust management decision. That is, an answer of "false" from the system would indicate "the action may not be taken either because sufficient credentials exist to deny the action or because sufficient credentials could not be found either to permit or to deny the action." Notice, however, that this solution intrinsically biases the trust management system against affirming questions asked of it, which may force users to ask questions in twisted forms to gain the desired mode of operation. (Note that this bias is often considered appropriate, particularly in "high security" applications; because many Web applications do not fit easily into traditional "high security paradigms," we have chosen to avoid this bias in REFEREE. For a more thorough discussion of the difficulties caused by allowing the trust management system to return a value of "unknown," see [BFRS, Section 4].) Further, it may be counter-productive to hide the fact that a decision could not be reached under the current policy from calling applications, especially when the policy is invoked as a subprogram of another policy, which may have alternative policies for dealing with a lack of credentials.

## Statement Lists

A REFEREE statement is a two-element structure consisting of some *content* and a *context* for that content. The context and the content of a statement are each arbitrary s-expressions; the context determines how the content is to be interpreted, and the interpretation of the context itself is subject to agreement between the calling application and REFEREE. Statement lists are simply unordered lists of REFEREE statements.

## Invocations

The ability for one policy to call another policy is central to REFEREE for two reasons. First, policies in general often defer judgment on some point to other policies: Bob may choose to allow his children to view any Web page that Alice allows her children to view. Second, evaluation of a particular request may require dangerous activity (such as network access), and invocations allow such dangerous actions to be executed from within REFEREE policies. Thus dangerous activity is controlled by policy, one aspect of REFEREE’s central tenet of "everything is under policy control."

All programs that can be invoked by a REFEREE program must conform to the REFEREE calling conventions. The required input to a program is a statement list defining the current evaluation context; programs may also accept additional arguments (either required or optional). The output of an invocation is a REFEREE expression value: a tri-value and a statement list. Typically, a program will append returned statements to one of its own statement lists.

Note that, once a REFEREE program invokes another program, control of the REFEREE engine is transferred to that program until it chooses to exit. Thus, invocation is inherently dangerous, because the invoking program has no way to monitor or interrupt execution of the invoked program. It is important, therefore, to reason carefully about whether it is acceptable to invoke a program before actually doing so.

### Installation

The REFEREE environment is extensible: It permits addition of new programs, as long as they conform to the calling conventions. Thus, if a new cryptographic hash function is defined, a new hash-checking program can be added, without reworking other components of a policy. Programs can even be downloaded and installed dynamically during execution. Once a callable program is downloaded over the Web, there is a trust management decision that must be made: whether the controlling REFEREE policy should allow the downloaded module to be invoked. REFEREE requires that a program explicitly *install* the invocable program into the database of named programs before it can be called. Thus, the addition of a new, callable program to REFEREE is controlled by the currently running policy.

Installations are not persistent across multiple calls to REFEREE unless the calling application explicitly takes action to make them so. That is, the calling application would need to record the new program and pass it in as part of the initial program database in future calls to REFEREE. Within an execution of REFEREE, installations have dynamic scope; a program can by default only install new functionality for itself and other programs it calls, not the programs that called it. Particular programs are free to override this default within their own scope and make installations effectively have global scope.

## The Profiles-0.92 Language

W3C’s PICS and DSIG working groups are designing a language for expressing the most common policies involving PICS labels and digital signatures. The current incarnation, *profiles-0.92*, has four important features:

1. Appending of statements returned by invoked programs
2. The *Load-labels* invocable program
3. Tri-value Combinators and Operators
4. Statement-list Pattern Matching

Profiles-0.92 includes a language construct, *invoke*, for calling another REFEREE program. Programs are called using REFEREE’s underlying mechanism. The name of the invoked program is prepended to the context portion of each returned statement; the statements are then appended to the initial statement list that was passed to the invoked program. In this manner, a program can invoke a subprogram to generate additional statements but keep track of which program generated them. The execution trace section provides a concrete example.

A profiles-0.92 program can invoke the load-labels program to look for PICS labels, either embedded in documents or retrieved over the network from a label bureau. Once found, the labels are parsed and written as REFEREE statements, which are returned.

Profiles-0.92 provides tri-value generalizations of the Boolean operators AND, OR, and NOT. For example, (AND true false) evaluates to false, but (AND true unknown) evaluates to unknown. In addition, the operators *true-if-unknown* and *false-if-unknown* coerce tri-values into conventional Boolean values.

Finally, profiles-0.92 provides a pattern-matcher that examines the statement list for statements of a particular form.

The next section illustrates features of the language.

## Sample Policies

We now present example policies written for REFEREE and explained in English. They illustrate the following three features of REFEREE:

- REFEREE can process and resolve conflicting information from multiple sources.
- Evaluation of policy itself may involve dangerous actions. REFEREE gives the user control over this.
- REFEREE allows a user to defer trust -- to authorize privileged credentials to decide the acceptability of another credential.

Some of the following examples use a hypothetical PICS rating service called "musac," with two dimensions, "sax" and "violins" (abbreviated "s" and "v" in PICS labels), and values 0, 1, and 2 along each dimension. Other examples use hypothetical rating services "http://www.e-trust.org/privacy-descriptions" and "http://w3.org/privacy" whose dimensions and values will be explained as needed. Throughout this section, we use the policy language "profiles-0.92" at the top level.

### Sample Policy 1: View any URL with a PICS label in the Musac system with "s < 2"

We are worried about excessive use of saxophones in music distributed over the Internet. We trust publishers and all others to declare honestly the number of saxophones used; we want to check this number before browsing the URL.

```
(invoke "load-label" STATEMENT-LIST URL "http://www.musac.org/" (EMBEDDED))
(false-if-unknown
 (match
  (("load-label" *)
   (* ((version "PICS-1.1") *
       (service "http://www.musac.org/") *
       (ratings (RESTRICT < s 2)))))
  STATEMENT-LIST))
```

This policy has two steps. First, we invoke `load-label` to find and download labels for the given URL; any labels found will be put on the statement list. We then run a pattern-matcher over the now-modified statement list, looking for any label using the rating service from "`http://www.musac.org/`" and with an s rating less than 2. If the matcher finds no musac label with an s dimension, it returns *unknown*, and, if it finds such a label, it returns *true* or *false* depending on whether or not the associated value is less than 2. The line `false-if-unknown` has the effect of converting a returned value *unknown* to *false*; this is a policy decision about which semantics to give to three-valued logic, specifically about the meaning of *unknown*. The overall effect is to allow viewing of any document for which we can find at least a single label with a musac-s rating less than 2.

The following examples will build on this policy template.

## Sample Policy 2: View URLs only if all PICS labels in the Musac system for the URL say "s < 2"

```
 (invoke "load-label" STATEMENT-LIST "http://www.musac.org/" URL (EMBEDDED))
 (false-if-unknown
  (match
   (("load-label" *)
    (* ((version "PICS-1.1") *
        (service "http://www.musac.org/") *
        (ratings (RESTRICT <! s 2)))))
    STATEMENT-LIST))
```

This policy is almost identical to Sample Policy 1 above, except that the restriction operator within the pattern matcher is "<!" instead of "<". The "!"-ending restriction operators require every matching statement that is tested against the operator to satisfy the operator. The effect is that *all* labels from all sources that indicate a sax level must indicate a low sax level. Thus a user can choose this policy instead of the last, depending on how the user wants to resolve simultaneous reception of a label from Alice saying musac-s level 1 and a label from Bob saying musac-s level 2.

An alternative way to handle conflicting labels is to specify the rater(s) we recognize. For example, the following variant tells the matcher to insist on labels from Alice:

```
 (invoke "load-label" STATEMENT-LIST URL "http://www.musac.org/" (EMBEDDED))
 (false-if-unknown
  (match
   (("load-label" *)
    (* ((version "PICS-1.1") *
        (service "http://www.musac.org/") *
        (by "mailto:alice")
        (ratings (RESTRICT < s 2)))))
   STATEMENT-LIST))
```

## Sample Policy 3: Combine simple filters; Use a label bureau

This time we are not worried about the content of URLs but instead about what a remote host will do with information we submit. Therefore we consult PICS labels in rating services that rate URLs according to the site's treatment of private information submitted by users. In the following, a value less than 2 in the "data-exchange" dimension of "http://www.e-trust.org/privacy-descriptions" indicates that the remote host gives no user-submitted data to anyone else. A value of 0 in the "personal-data-collected" dimension of "http://w3.org/privacy" indicates that no information at all is retained.

```
(invoke "load-label" STATEMENT-LIST
         "http://www.e-trust.org/privacy-descriptions" URL ("http://labels.com"))
(invoke "load-label" STATEMENT-LIST
        "http://w3.org/privacy" URL ("http://labels.com"))
(false-if-unknown
 (and
  (match
   (("load-label" *)
    (* ((version "PICS-1.1") *
        (service "http://www.e-trust.org/privacy-descriptions") *
        (ratings (RESTRICT < data-exchange 2)))))
   STATEMENT-LIST)
  (match
   (("load-label" *)
    (* ((version "PICS-1.1") *
        (service "http://w3.org/privacy") *
        (ratings (RESTRICT = personal-data-collected 0)))))
   STATEMENT-LIST)))
```

This policy uses labels from two services. In the first step, we load labels for `URL` by invoking "`load-label`" twice, once for each rating service; "`load-label`" adds statements to the statement-list. We then search the returned statements for labels using two rating services, E-Trust and w3.org's privacy service. Each pattern-match uses criteria specific to one rating service, and then the results of those two pattern-matches are combined using `and`. Thus, in order for the policy to be satisfied, there must be labels in both rating services.

In the previous examples, the requested document was downloaded and scanned for embedded labels, which would be used to decide whether or not to display the document. In a privacy application, what is at stake is our private information that could be passed to the remote host, including the fact that we accessed the site at all. For this reason, the user here has decided not to scan the document for labels (which would involve connecting to the remote site), but only to request labels separately from the document (from the bureau "http://labels.com," which the user trusts for this purpose). Using this policy, we won't even connect to the originally-requested remote site unless we've first decided that it is safe.

## Sample Policy 4: Defer trust to GoodMouseClicking

The alternate in sample policy 2 explicitly trusted a single rater of content, Alice. A more likely situation is that we would trust a single auditor (or a small number of auditors) and accept only labels from raters unknown to us but endorsed by the auditor. In this example, we assume that the user trusts GoodMouseClicking to endorse raters. The user sets a policy of trusting reviews from those reviewers for which GoodMouseClicking vouches.

```
 (invoke "load-label" STATEMENT-LIST URL
            "http://www.musac.org/v1.0" ("http://labels.com/"))
 (invoke "endorse-label" STATEMENT-LIST
        "mailto:GoodMouseClicking@w3.org" ("http://labels.com/"))
 (false-if-unknown
  (match
   (("endorse-label" *)
    (* "mailto:GoodMouseClicking@w3.org" *
     ((version PICS-1.1) *
      (service "http://www.musac.org/v1.0") *
      (ratings * (RESTRICT < v 2) * )))))
   STATEMENT-LIST))
```

The module `endorse-label` is a separate program that handles requests for deferral of trust. It takes a rater and a label bureau as arguments and contacts the label bureau to request labels from the specified rater that vouch for the author of each of the statements on STATEMENT-LIST. When GoodMouseClicking is found to vouch for a statement's author, GoodMouseClicking's identifier is added to the beginning of that statement, and the new statement is returned, to be added to the caller's STATEMENT-LIST. As in the previous examples, the *match* expression then searches the statement list, in this case for a statement added by the `endorse-label` program that begins with "mailto:GoodMouseClicking@w3.org."

The same principle can be used to insist on signed labels, in case we are worried about forgeries. Instead of invoking `endorse-labels`, we could invoke a module that checks a digital signature or performs other types of cryptographic checks.

### Sample policies -- summary

These examples illustrate three key features of the referee system. First, we saw in sample policy 2 how REFEREE can process and resolve conflicting information from multiple sources, by indicating whether all labels must agree or the existence of a single label of the desired type carries the decision. In sample 3, we saw that the very process of deciding when a URL is acceptable may involve similar trust decisions; REFEREE puts the entire process under policy control. Finally, in sample 4, we saw how REFEREE can easily adapt to new cryptographic algorithms and new paradigms for security (deferral of trust).

## An Execution Trace

We present below an execution trace extracted from a working REFEREE demo. An on-line version of this demo can be found in `http://www.w3.org/pub/WWW/PICS/TrustMgt/demo/Overview.html`.

The policy used in this section extends sample policy 4 of the previous section to include checking of hash functions. Here we emphasize the way different modules interact with each other as well as the data flow. Below is the policy:

```
   (invoke "load-label" STATEMENT-LIST URL
         "http://www.musac.org/v1.0" ("http://labels.com/"))
(invoke "check-hash" STATEMENT-LIST)
(invoke "endorse-label" STATEMENT-LIST
         "mailto:GoodMouseClicking@w3.org" ("http://labels.com/"))
(false-if-unknown (match (("endorse-label" "check-hash" *)
                          (* "mailto:GoodMouseClicking@w3.org" *
                            ((version PICS-1.1) *
                             (service "http://www.musac.org/v1.0") *
                             (ratings * (RESTRICT < v 2) * ))))
                         STATEMENT-LIST))
```

There are four modules used here: `profiles-0.92`, `load-label`, `check-hash` and `endorse-label`. The `profiles-0.92` module is the top-level module called by the REFEREE invoker to interpret the policy as described above, the `check-hash` module verifies hash values in PICS labels, and the `load-label` and `endorse-label` modules are explained in the previous Section. When the caller invokes REFEREE with this policy, `profiles-0.92` first invokes `load-label`. Now assume `load-label` actually gets one label from the label bureau `http://labels.com` and returns the following to `profiles-0.92`:

```
tri-value = true
statement-list = ((()
                   ((version "PICS-1.1")
                    (service "http://www.musac.org/v1.0")
                    (by "John")
                    (md5 "7A2B1a2bA72BxyzyplehJQ==")
                    (original (PICS-1.1 ...))
                    (ratings (s 1) (v 0)))))
```

The module `load-label` returns *true*, because a label is found. The statement list contains a single statement describing the PICS label. The context of the statement is empty, because it is produced by the module itself. The caller `profiles-0.92` records this statement by prepending the name of the called module, "load-label," onto each context and appending the result

```
statement-list = ((("load-label")
                    ((version "PICS-1.1")
                     (service "http://www.musac.org/v1.0")
                     (by "John")
                     (md5 "7A2B1a2bA72BxyzyplehJQ==")
                     (original (PICS-1.1 ...))
                     (ratings (s 1) (v 0)))))
```

onto its local copy of `STATEMENT-LIST`.

Now `profiles-0.92` proceeds to the second line to check MD5 with the parsed statement above. The module `check-hash` is passed the above statement-list and, assuming the hash is good, returns

```
tri-value = true
```

and the same statement list it was passed. The caller, `profiles-0.92,` prepends "check-hash" onto each context and appends the following statement to its copy of `STATEMENT-LIST`:

```
statement-list = ((("check-hash" "load-label")
                    ((version "PICS-1.1")
                     (service "http://www.musac.org/v1.0")
                     (by "John")
                     (md5 "7A2B1a2bA72BxyzyplehJQ==")
                     (original (PICS-1.1 ...))
                     (ratings (s 1) (v 0)))))
```

The presence of "check-hash" in the *context* of the statement indicates the hash verification. Now `profiles-0.92` proceeds to the third line, invoking the module `endorse-label` to check for an endorsement. Assuming the endorsing label is found, `endorse-label` returns a statement-list that gets "endorse-label" prepended to the context, resulting in the following:

```
tri-value = true
statement-list = ((("endorse-label" "check-hash" "load-label")
                    (("mailto:GoodMouseClicking@w3.org")
                     ((version "PICS-1.1")
                      (service "http://www.musac.org/v1.0")
                      (by "mailto:John")
                      (md5 "7A2B1a2bA72BxyzyplehJQ==")
                      (original (PICS-1.1 ...))
                      (ratings (s 1) (v 0))))))
```

Again, the string "endorse-label" is added to the context of this statement to indicate that the rater of this PICS label statement is approved by `endorse-label` policy. The passed content is wrapped in an expression containing "mailto:GoodMouseClicking@w3.org" (the name of the endorser).

Finally, `profiles-0.92` proceeds to the last line to check ratings. The match looks for a context with both "endorse-label" and "check-hash" -- if either is missing, the match fails. Because the match succeeds, `profiles-0.92` returns to the application a tri-value of *true* and a statement-list of the statements produced by the last form, the *match*:

```
tri-value = true
statement-list = ((("endorse-label" "check-hash" "load-label")
                  (("mailto:GoodMouseClicking@w3.org")
                   ((version "PICS-1.1")
                    (service "http://www.musac.org/v1.0")
                    (by "mailto:John")
                    (md5 "7A2B1a2bA72BxyzyplehJQ==")
                    (original (PICS-1.1 ...))
                    (ratings (s 1) (v 0))))))
```

The returned values say the action should be taken (tri-value is *true*), and it is justified by the rater-endorsed, hash-checked PICS label in the statement-list. They are returned to the caller application (such as a browser), and it interprets the action appropriately (e.g., by displaying the URL on the screen).

## Execution Trace -- Summary

This execution trace demonstrates that a typical decision about viewing a URL may involve (at least) the tasks of

- fetching credentials
- authenticating credentials
- deciding which credentials are trustworthy.

Users may want to customize their policies for these tasks individually, and REFEREE allows a user to express each policy in a separate module. The modules communicate via a common API that consists of passing statements in the convenient context-content format and returning both a *true/false/unknown* result and a justification in the form of additional statements.

## Conclusion

The most important feature of REFEREE is that it places all dangerous operations under policy control, rather than making arbitrary decisions to accept certain dangers and avoid others. For example, it might seem natural to avoid the danger of spoofed labels by requiring that the authenticity of all labels be verified by checking digital signatures. The likelihood of spoofed labels and their negative consequences, however, depend on the context. REFEREE makes it possible to write a policy that requires signature checking, one that does not, or one that does only in certain circumstances.

The Web will benefit greatly from a common platform for trust management, because different organizations will be able to develop component programs. REFEREE is a promising candidate platform: Its calling conventions impose minimal requirements on component programs while ensuring that they interoperate. As new safety features are desired, they can be added as installable programs, without rewriting all the other features.

## References

[BFL] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in Proceedings of the 1996 IEEE Symposium on Security and Privacy, pp. 164-173.

[BFRS] M. Blaze, J. Feigenbaum, P. Resnick, and M. Strauss, "Managing Trust in an Information-Labeling System," to appear in European Transactions on Telecommunications. Available as AT&T Technical Report 96.15.1.

[N] G. Necula, "Proof-Carrying Code," to appear in Proceedings of the 1997 ACM Symposium on Principles of Programming Languages.

[NL] G. Necula and P. Lee, "Safe Kernel Extensions without Run-time Checking," in Proceedings of the 1996 Usenix Symposium on Operating System Design and Implementation, pp. 229-243.

[RM] P. Resnick and J. Miller, "PICS: Internet Access Controls without Censorship," Communications of the ACM, 39 (1996), pp. 87-93. Also available as http://www.w3.org/pub/WWW/PICS/iacwcv2.htm.