Network Working Group
Internet-Draft
Expires: July 1, 2004

R. Hedberg
Stockholm University
O. Bandmann
L4i
January 2004

Restricted S-expressions for use in a generalized authorization service
draft-hedberg-spocp-sexp-00

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on July 1, 2004.

Copyright Notice

Table of Contents

1. Abstract

This document describes restricted S-expressions as they are used for storing and querying for access rights within the SPOCP (Simple POlicy Control Protocol) project. We describe the restrictions we have made to basic S-expressions and also the theory that allows us to use S-expressions in a policy engine.

2. Introduction

The aim of the SPOCP project is to first develop a model for a generalized authorization service server and then to implement such a server. Generalized in this context means that it shall be equally good in supporting several different types of applications and that one and the same server shall be able to simultaneously support several applications.

To achieve this goal we needed to design a policy engine that could evaluate policies without knowing what applications the policies referred to. The first step towards this goal was to pick a rule syntax that was independent of the applications, and we think we have found such a syntax in S-expressions [s-expression].

The goal of this document is to describe how S-expressions can be used in a generalized authorization service, and what restrictions we have applied to S-expressions to make them really useful.

The two companion documents [spocp_prot] and [spocp_prot_tcp] describes the Simple Policy Control Protocol and one implementation of it.

The terms used in this draft is defined in [RFC2828].

3. Background

S-expressions is not something new on the Internet arena, "The Simple Public Key Infrastructure" (SPKI) working group within the IETF, based its work on S-expressions. They also made restrictions on the syntax of the S-expressions (See for instance [RFC2693]), something we have built on in our work.

In contrast to the SPKI work we are not dealing with certificates but have instead concentrated on using S-expressions as a policy language syntax. A language suitable to express both access policies and queries for permissions.

The differences between restricted S-expressions as defined by SPKI and the restrcited S-expressions defined in this document are slight but significant. They can be enumerated as:

1. We have added a companion to prefix called suffix

2. We do not distinguish between ALPHA and BINARY, there are treated as one and the same

3. We have added the restriction that all lists in a set construct have to have different tags

An important change is that we have replaced the AIntersect operation with a partial order (pre-order, strictly speaking) compatible with AIntersect. In order to guarantee completeness of the decision algorithm described in section 6, the restriction in item 4 above is needed (cf. [spki authz]).

4. Problems with Access Control Lists

There are several problems with ACLs as they are normally used in applications, that disappear if access control is based on policy articulated in S-expressions. We list some of these problems below and explain how they can be handled in a authorization policy written using S-expressions.

1. The identity of future clients has to be known

   An application that wants to use S-expressions for authorization decisions, has a template for S-expression construction. Whether a token representing the identity of the client is part of that template or not, is a local matter and irrelevant to the use of S-expressions. Hence neither the application nor the authorization system needs to know the identity of the client.

2. ACLs are static

   When constructing the rules, you might not know or care about who will fulfill the restrictions when an access right is requested. Even if a rule appears to be static, the set of persons and/or entities that fulfills the restrictions might be highly dynamic.

3. The application has to have all the information necessary for making the access decision.

   It is not a problem if the application does not have access to all the necessary information, as long as the Spocp server, or an application it can use, has.

5. Restricted S-expressions for authorization

5.1 Simple S-expressions

A simple S-expression is a nested list enclosed in matching "(" and ")". The first element in the list MUST be an atom (string) and is the "tag" or "name" of the object represented by the list. With that exception, every element in the list may in turn be a S-expression. Note that empty lists are not allowed.

As in SPKI, we have chosen Rivest's compact "canonical form", see [s-expression], as our internal representation of an S-expression.

A complete description of restricted S-expressions using ABNF [RFC2234] is given in Appendix A .

S-expressions are used at the core in the authorization server, and may be sent from a client to a server. If they are, the canonical form is to be used [s-expression]. A canonical S-expression is formed from octet strings (that is every octet can assume any byte value between and including 0x00 and 0xFF), each prefixed by its length. The length of a byte string is a non-negative ASCII decimal number, with no leading "0" digits, terminated by ":". The canonical form is a unique representation of an S-expression and is used as the input to all hash and signature functions.

```
s-expr      = "(" tag *s-part ")"

tag         = octet-string

s-part      = octet-string / s-expr / star-form

octet-string = decimal ":" 1*octet ; The number of octets should
be equal to the decimal specification

decimal     = nzdigit *digit

nzdigit     = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

digit       = "0" / nzdigit

octet       = %x00-FF

star-form   = "(1:*" [ set / range / prefix / suffix ] ")"
```

The specification of the star forms can be found in Section 5.3.

Note: Even though the canonical form is the one described by the

ABNF definition, the so called advanced form will be used in the examples in this document since it is much easier for humans to read.

Example:

(5:spocp(8:Resource6:mailer)) -- canonical form

(spocp (Resource mailer)) -- advanced form

These are two representations of the same S-expression, consisting of an octet string (the tag) "spocp" and another S-expression, that consists of two octet strings "Resource" and "mailer".

5.2 Basic theory

In order to be able to use S-expressions for authorization, two criteria have to be fulfilled. THe first is that S-expressions must have the expressive power needed for conveniently stating an authorization policy. Our practical experience has convinced us that this criterion is satisfied, The other thing needed is the definition of a binary relation '<=', that can be used to order S-expressions.

We want a relation where A '<=' B means that rule A is less permissive than rule B. Once the relation is defined, we also need an efficient way to decide (compute) if A '<=' B. A decision algorithm for restricted S-expressions is given in Section 6 We begin by defining '<=' for simple S-expressions; in the next subsection, '<=' will be extended to general restricted S-expressions.

In [spki_authz] '<=' has been defined inductively as follows: Let x and y be simple S-expressions then

1. if x and y are simple 'atomic' elements (strings) then x '<=' y if and only if x = y.

2. If x = (x[0] x[1] ... x[N]) and y = (y[0] y[1] ... y[M]), then x '<=' y if and only if N >= M and x[i] '<=' y[i], for i = 0, ..., M

Example 1, If,

x = (http (page index.html)(action GET)(user olav))

then x is intended to represent the authorization to the user olav to read (in HTTP terms GET) the page index.html using HTTP.

Let

y = (http (page index.html)(action GET)(user))

Then y means almost the same as x except for the fact that the permission to read index.html is given to any user. By definition x '<=' y. Furthermore, if

z = (http (page index.html)(action)(user olav))

then z means almost the same as x except for the fact that now Olav can perform any operation on index.html that HTTP supports. Note that y and z are unrelated with respect to the partial order '<='.

From the example above it should be obvious that the application generating these S-expressions has restrictions on the format of them, restrictions that correspond to the desired semantics. It is essential to the idea of a centralized authorization service that this semantic does not require a modification of the '<=' relation.

The intended use of S-expressions for authorization evaluation is as follows. Assume that a certain principal P wants to perform an action A requiring the authorization X. Then P has the authorization for A if and only if P has the some authorization Y satisfying X '<=' Y.

More about partial ordering in Section 6, we have to introduce you to star forms first.

So by the use of S-expressions, and the partial order we get an important benefit: we can build an authorization system that works independently of what the policies actually mean.

5.3 Star forms

To extend simple S-expressions to restricted S-expressions we have to add a new type of element: star forms. These can be divided into the following categories:

- wildcard
- set
- range
- prefix
- suffix

Despite their list-like apearence (see below), starforms are not lists. They are succinct ways of representing every element that fits

into a specific set. Hence, restricted S-expressions (simple S-expressions extended with star forms) really represent _sets_ of simple S-expressions.

In order to preserve the intended semantics for the ordering '<=' (from the previous subsection), the only possible way to extend this relation to _sets_ of simple S-expressions, is to define:

X '<=' Y if and only if every simple S-expression A in X is bounded by some simple S-expression B in Y (i.e. A '<=' B in the sense of previous subsection)

An algorithm for effective computation of this relation is given in section 6 (cf. [spki_authz]).

5.3.1 The wildcard star form

Is written '(*)' and matches any single octet string or s-expression.

5.3.2 The set star form

Described by the ABNF

```
set = "3:set" 1*s-expr
```

They are a way of specifying a limited set of elements, a group.

Example:

```
(* set apple orange lemon)
```

The important difference between this star form 'set' and the one in SPKI ([RFC2693]), is that here, 'set' is restricted in the following way: all lists appearing at the top level in a 'set'-construction MUST have different tags. This restriction implies completeness of the algorithm for computation of '<=' presented in Section 6. The following is an example of a valid restricted S-expression:

```
(t (* set (a x) (b (a y)) (c) a) a)
```

and this one is not:

```
(t (* set (a (x y)) (b c) (a d)))
```

Furthermore, to simplify and streamline the algorithm description in Section 6, we will also make the trivial restriction on the set star form, that a set is not permitted to contain a set as a top level element. E.g.

```
(* set (* set x y) z )
```

can not be part of a restricted S-expression. While, on the other hand,

```
(* set x y z )
```

can. Immediately nested sets can always be eliminated in this fashion (without changing the semantics). Note that deeper nestings (i.e. within lists) are permitted. E.g.

```
(* set (x (* set y z)) t)
```

can be part of a restricted S-expression.

5.3.3 The range star form

Since one needs to know the type when one deals with ranges, there are a couple of types predefined.

alpha: which is normal text

numeric: non-negative numbers between 0 and 4294967295 (UINT32_MAX)

date: date specification of the form YYYY-MM-DD_HH:MM:SS or using the notation used by strftime %G:%m:%d_%H:%M:%S

time: time of day specification HH:MM:SS

ipv4: the IPv4 address in the normal dot notation format

ipv6: IPv6 address in their normal notation

In the specification of a range you may use constants in these types in combination with relational operators in a straight forward way. The ABNF specification for range is:

```
rangespec      = alpha / numeric / date / time / ipv4 / ipv6

alpha          = "5:alpha" [lole utf8string [goge utf8string]] /
[goge utf8string [lole utf8string]]

numeric        = "7:numeric" [ lole number [ goge number ]] / [
goge number [ lole number ]]

date           = "4:date" [ goge dat [ lole dat ]] / [ lole dat [
goge dat ]]
```

```
time          = "4:time" [ lole hms [ goge hms ]] / [ goge hms [
lole hms ]]

ipv4          = "4:ipv4" [ lole ipnum [ goge ipnum ]] / [ goge
ipnum [lole ipnum ]]

ipv6          = "4:ipv6" [ lole ip6num [ goge ip6num ]] / [ goge
ip6num [lole ip6num ]]

lole          = "2:lt" / "2:le"

goge          = "2:gt" / "2:ge"

number        = decimal ":" 1*digit

dat           = decimal ":" date-time   ; date format as
specified by RFC3339

date-fullyear  = 4DIGIT

date-month     = 2DIGIT  ; 01-12

date-mday      = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 dependent on
month/year

time-hour      = 2DIGIT  ; 00-23

time-minute    = 2DIGIT  ; 00-59

time-second    = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap
second rules

time-secfrac   = "." 1*DIGIT

time-numoffset = ("+" / "-") time-hour ":" time-minute

time-offset    = "Z" / time-numoffset

partial-time   = time-hour ":" time-minute ":" time-second

full-date      = date-fullyear "-" date-month "-" date-mday

full-time      = partial-time time-offset

date-time      = full-date "T" full-time

hms            = decimal ":" partial-time
```

```
ipnum          = decimal ":" 1*3digit "." 1*3digit "." 1*3digit
"." 1*3digit

ip6num         = IPv6address ; as defined in [RFC2373]

utf8string     = decimal ":" 1*UTF8

UTF8           = %x01-09 / %x0B-0C / %x0E-7F / UTF8-2 / UTF8-3 /
UTF8-4 / UTF8-5 / UTF8-6

UTF8-1         = %x80-BF

UTF8-2         = %xC0-DF UTF8-1

UTF8-3         = %xE0-EF 2UTF8-1

UTF8-4         = %xF0-F7 3UTF8-1

UTF8-5         = %xF8-FB 4UTF8-1

UTF8-6         = %xFC-FD 5UTF8-1
```

Finally, note that there is the extra requirement (compared to SPKI) that a range star form always must contain at least two elements. In other words, redundant singleton ranges MUST be replaced by (single) atoms.

Example

(worktime (* range time ge 08:00:00 le 17:00:00))

or

(* range numeric l 15 ge 10)

which is the same as

(* set 10 11 12 13 14)

If in a date specification, time-offset is not 'Z' but a time-numoffset the equivalent date without time-numoffset must be calculated before the value is used. "2002-12-31T23:59:59+01" must be transform to "2003-01-01T00:59:59" before usage.

5.3.4 The prefix star form

Used to represent sets of strings that all have the same prefix ABNF:

prefix = "6:prefix" utf8string

Example

(file (* prefix conf))

This expression will match any expression with the tag "file", whose second element is an octet string that starts with the string "conf".

5.3.5 The suffix star form

Used to represent sets of strings that all have the same suffix

ABNF:

suffix = "6:suffix" utf8string

Example

(file (* suffix pdf))

This expression will match any expression with the tag "file", whose second element is an octet string that ends with the string "pdf".

6. S-expression comparison

In this section we present an effective algorithm (from [spki_authz]) to decide the other relation '<=' defined in Section 5.2 and Section 5.3.

Recall the definition of '<=' for simple S-expressions from Section 5.2:

> For two octet strings A and B, A '<=' B if and only if A == B
>
> If S and T are lists, then S '<=' T if S has at least as many elements as T and every element in S is '<=' the corresponding element in T (if S has more elements than T, just ignore the extra elements in S).

Example:

> (fruit apple large red) '<=' (fruit apple)
>
> (fruit apple (size large) red) '<=' (fruit apple (size) red)

and these are not '<='

> (fruit apple large red) compared to (fruit apple (large) red)
>
> (fruit apple large red) compared to (fruit apple red large)

order is absolutely vital

> (apple (weight 100)(color red)) is not '<=' (apple (color red)(weight 100))

Thus, in the case of simple S-expressions the definition of '<=' immediately gives us an algorithm. For general restricted S-expressions the following recursive procedure gives us an algorithm. Before the algorithm can be applied, however, the restricted S-expressions which are to be compared need to be normalized.

To normalize an element of a restricted S-expression means that in each set star form, ranges of the same type and atoms are joined together in single ranges, whenever possible. E.g.

> (* set 44 (* range numeric ge 4 le 8) 11 (* range numeric ge 6 le 10))

normalizes to

(* set (* range numeric ge 4 le 11) 44)

The "normal form" is obviously not syntactically unique (even though it is semantically unique), but further reductions should not be possible.

After normalization, the algorithm proceeds as follows. If any of the nine cases below applies, the comparison returns true, otherwise it returns false.

S '<=' T, when S and T are normalized elements of S-expressions, if:

1. T = (*)

2. S and T are strings and S == T

3. S is a string and T is a set, range, suffix or prefix star form that contains S

4. S and T are range-forms where T contains S

5. S and T are prefix-forms where T contains S

6. S and T are suffix-forms where T contains S

7. S = (X[0] ... X[m]), T = (Y[0] ... Y[n]) n &lt= m and X[i] '<=' Y[i] for i = 0,...,n

8. S = (* set X[0] ... X[m]) and X[i] '<=' T for all i=0,..,m

9. T = (* set Y[0] ... Y[n]) and S '<=' Y[i] for some i=0,..,n

Strictly speaking, there are a few other (trivial) pathological cases to deal with, see [spki_authz]. In particular, here we have made the simplifying assumption that range and prefix/suffix star forms are incomparable w.r.t. '<='.

Finally, a proof of soundness and completeness for this algorithm, when applied to restricted S-expressions, can also be found in [spki_authz].

7. Security considerations

Authorization decisions obviously have an immediate impact on security. Concerning the choice of S-expressions as a syntax for representing access policies, the only real security concern, on this level, is whether using S-expressions in some way, is inherently insecure. On a theoretical level it has been shown (see [spki_authz]) that the algorithm to decide the '<=' relation on restricted S-expressions is both sound (never falsely claims that the relation holds) and complete (whenever the relation holds, the algorithm returns 'true').

8. Acknowledgment

This work originated at at the Swedish Institute of Computer Science (SICS). Babak Sadighi had the original thoughts on management of rigths, Olav Bandmann brought S-expressions into the process and together with Mads Dam he did the mathematical evaluation of the less permissive relationship between S-expressions.

The Spocp project is funded by SUNET (The Swedish University Network), UNINETT (The Norwegian University Network), the universities in Ume , Uppsala, Stockholm and Lund, The Karolinska Intitute and the NyA project.

Torbj÷rn Wiberg is the project leader for the Spocp project and has been very active in the project work. Leif Johansson and Ola Gustafsson has been heavily involved in the technical development of the project.

References

[RFC1738] Berners-Lee, T., Masinter, L. and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC2828] Shirey, R., "Internet Security Glossary", RFC 2828, May 2000.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2252] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[RFC2253] Wahl, M., Kille, S. and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[RFC2693] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. and T. Ylonen, "SPKI Certificate Theory", RFC 2693, September 1999.

[RFC2712] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M. and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[SDSI] Rivest, R. and B. Lampson, "SDSI - A Simple Distributed Security Infrastructure", <http://theory.lcs.mit.edu/~cis/sdsi.htm>.

[sexp_code] Rivest, R., "code and description of S-expressions", <http://theory.lcs.mit.edu/~rivest/sexp.html>.

[s-expression] Rivest, R., "S-Expressions", ID draft-rivest-sexp-00.txt,

May 1997, <http://theory.lcs.mit.edu/~rivest/sexp.txt>.

[spocp_prot]
Hedberg, R., "The Simple Policy Protocol".

[spocp_prot_tcp]
Hedberg, R., "The Simple Policy Control Protocol over TCP/IP".

[spki_authz]
Bandmann, O. and M. Dam, "A Note On SPKI's Authorisation syntax", <http://www.cs.dartmouth.edu/~pki02/Bandmann/>.

Authors' Addresses

Roland Hedberg
Stockholm University
Kasamark 114
Umea 90586
Sweden

Phone: +46 90 147275
EMail: roland@it.su.se

Olav Bandmann
Industrilogik L4i AB
Odengatan 87
Stockholm 11322
Sweden

EMail: olav@L4i.se

Appendix A. Collected Grammar

This appendix contains the complete ABNF [RFC2234] grammar for all the syntax specified by this document.

By itself, however, this grammar is incomplete. It refers by name to syntax rules that are defined by RFC 3339. Rather than reproduce those definitions here, and risk unintentional differences between the two, this document simply refers the reader to RFC 3339 for the remaining definitions.

```
s-expr          = "(" tag *s-part ")"

tag             = octet-string

s-part          = octet-string / s-expr / star-form

octet-string    = decimal ":" 1*octet ; The number of octets must
be equal to the decimal specification

decimal         = nzdigit *digit

nzdigit         = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" /
"9"

digit           = "0" / nzdigit

octet           = %x00-FF

star-form       = "(1:*" [ set / range / prefix / suffix ] ")"

set             = "3:set" 1*s-expr

range           = "5:range" rangespec

rangespec       = alpha / numeric / date / time / ipv4 / ipv6

alpha           = "5:alpha" [lole utf8string [goge utf8string]] /
[goge utf8string [lole utf8string]]

numeric         = "7:numeric" [ lole number [ goge number ]] / [
goge number [ lole number ]]

date            = "4:date" [ goge dat [ lole dat ]] / [ lole dat [
goge dat ]]

time            = "4:time" [ lole hms [ goge hms ]] / [ goge hms [
lole hms ]]
```

```
ipv4          = "4:ipv4" [ lole ipnum [ goge ipnum ]] / [ goge
ipnum [lole ipnum ]]

ipv6          = "4:ipv6" [ lole ip6num [ goge ip6num ]] / [ goge
ip6num [lole ip6num ]]

lole          = "2:lt" / "2:le"

goge          = "2:gt" / "2:ge"

number        = decimal ":" 1*digit

dat           = decimal ":" date-time ; date-time format as
specified by RFC3339

hms           = decimal ":" partial-time  ; partial-time as
define by RFC3339

ipnum         = decimal ":" 1*3digit "." 1*3digit "." 1*3digit
"." 1*3digit

ip6num        = IPv6address ; as defined in [RFC2373]

utf8string    = decimal ":" 1*UTF8

UTF8          = %x01-09 / %x0B-0C / %x0E-7F / UTF8-2 / UTF8-3 /
UTF8-4 / UTF8-5 / UTF8-6

UTF8-1        = %x80-BF

UTF8-2        = %xC0-DF UTF8-1

UTF8-3        = %xE0-EF 2UTF8-1

UTF8-4        = %xF0-F7 3UTF8-1

UTF8-5        = %xF8-FB 4UTF8-1

UTF8-6        = %xFC-FD 5UTF8-1

prefix        = "6:prefix" utf8string

suffix        = "6:suffix" utf8string

typespecific  = *UTF8
```

Appendix B. Representing hierarchies

When we have been working with S-expression we have found it useful to split queries and rules into three parts:

Resource The resource that someone want to use or perform some action on

Action The action that is to be performed on the said resource

Subject The entity that wants to perform the action on the resource

In many situations your application has organised and named both subjects, sctions and resources as hierarchies. If you want to take full advantage of the hierarchical names in rules and queries you have to study carefully how S-expressions are evaluated by the policy engine. Assume that a name is represented as (name p[0] ... p[n]) where p[0] is the part of the name that is closest to the root of the hierarchy. Then you can represent the whole space of names below p[0], by just specifying the top part of the namespace: (name p[0]). Correspondingly you can represent a specific part of the namespace by defining a larger part of the hierarchy (name p[0] ... p[m]), m < n.

But what if you would like to represent every object who has the same last name p[n] ?

An example of when this would be is if you defined role names within a organization as a concatenation of the organization name, the name of all the organizational units from the top with the roletype. Like this: (role o ou[0] ... ou[n] r)

"(role UmU Umdac boss)" would then be the rolename for the boss of the organizational unit Umdac within the organization UmU.

Using this structure you could say (role UmU Umdac) and mean every role within that organizational unit and all the organizational units below. But if you said (role UmU boss) you would refer to the boss of UmU and not all the bosses within UmU. This since (role UmU umdac boss) is not '<=' (role UmU boss). So adding a role type to a list of O and OU's would mean exactly that role at that level in the organization.

If you instead would define the role name to be represented as (role r o ou[0] .. ou[n]), then you could address every specific roletype within the organization by writing things like (role boss UmU), which would then mean every 'boss' within the organization UmU. This follows since (role boss UmU OU) is '<=' (role boss UmU). On the other hand you could not specifically target the boss at UmU using

this representation.

One can add complexity to this by using role types that are hierarchical such that the name would be (role o ou[0] ... ou[n] r[0] ... r[m]) or (role r[0] ... r[m] o ou[0] ... ou[n]). By using the first form you could address every role within a role hierarchy at a specific place in the organization hierarchy but not in the whole organization tree. Using the later role you could address one whole subtree of the role hierarchy anywhere within a subtree of the organizational hierarchy.

(role UmU admin finance) '<=' (role UmU admin)

(role UmU umdac admin) is not '<=' (role UmU admin)

and

(role admin UmU umdac) '<=' (role admin UmU)

(role admin finance UmU) is not '<=' (role admin UmU)

Remember that the decision of the meaning of a particular rule is taken when modelling the authorisation policy for a particular application. The Policy Engine does not know anything about the application. It only compares queries to rules according to builtin evaluation rules for restricted S-expressions, as described in this document. What we are discussing in this section are the consequences of choosing certain meanings of a particular S-expression, given how the Policy Engine tests for the '<='-relation. These properties of the Policy Engine must be fully understood by those deciding the structures of rules and queries.

When you have two hierarchies that are linked to each other it might be best to decouple them and make two lists of them, (role (org o ou[0] ... ou[n])(type r[0] ... r[m])) which gives you freedome to express the relationship "any role whithin a role hierarchy anywhere within a organization hierarchy".

(role (org UmU) (type admin finance)) '<=' (role (org UmU) (type admin))

(role (org UmU umdac) (type admin)) '<=' (role (org UmU) (type admin))

There is of course nothing that prevents you from using one nameform in one set of rules and another form in another as long as the queries you pose to the policy engine use the appropriate one. What you should make certain though is that the form you choose gives you

the possibility to express exactly what you are aiming for.

Appendix C. Representing Security Connection

Lots of applications uses SSL/TLS to protect the connection between a client and a server. This is a good reason for specify how the information about such a connection should be represented in a S-expression.

The information present are:

- SSL/TLS version
- Cipher Suite used
- SubjectDN
- IssuerDN

So a plausible structure which then would describe the connection as viewed from one of the partners (either the client of the server) would be:

(TransportLayerSec (protocolVersion <major> <minor>) (chipherSuite <ciphersuite>) (autname "X509" (subject <subjectDN>) (issuer <issuerDN>)))

If X.509 certificates are in use, if instead kerberos [RFC2712] was used that would only change the later part of the structure:

(TransportLayerSec (protocolVersion <major> <minor>) (chipherSuite <ciphersuite>) (autname "gss-name" (uid <uid>) (realm <realm>)))

Remembering that this connection information is about the connection between a client and a application server that gives access to some resource, and that the application server probably has its restrictions on what kind of connections, ciphersuites and clientcertificates combinations it will accept. So this is about having a second opinion from the owner of the resource on which combination it allows. If it is more restrictive than the application server you might end up with the situation where the client gets a SSL/TLS protected connection to the server but no data will flow over the connection because the resource owner demands that a different ciphersuite must be used.

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.