         Configuration of Access Control Policy in REsource LOcation And
                        Discovery (RELOAD) Base Protocol
                  draft-petithuguenin-p2psip-access-control-05

Abstract

   This document describes an extension to the REsource LOcation And
   Discovery (RELOAD) base protocol to distribute the code of new Access
   Control Policies without having to upgrade the RELOAD implementations
   in an overlay.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The RELOAD base protocol specifies an Access Control Policy as
"defin[ing] whether a request from a given node to operate on a given
value should succeed or fail." The paragraph continues saying that
"[i]t is anticipated that only a small number of generic access
control policies are required", but there is indications that this
assumption will not hold. On all the RELOAD Usages defined in other
documents than the RELOAD base protocol, roughly 50% defines a new
Access Control Policy.

The problem with a new Access Control Policy is that, because it is
executed when a Store request is processed, it needs to be
implemented by all the peers and so requires an upgrade of the
software. This is something that is probably not possible in large
overlays or on overlays using different implementations. For this
reason, this document proposes an extension to the RELOAD
configuration document that permits to transport the code of a new
Access Control Policy to each peer.

This extension defines a new element <access-control-code> that can
be optionally added to a <configuration> element in the configuration
document. The <access-control-code> element contains ECMAScript
[ECMA-262] code that will be called for each StoredData object that
use this access control policy. The code receives four parameters,
corresponding to the Resource-ID, Signature, Kind and StoredDataValue
of the value to store. The code returns true or false to signal to
the implementation if the request should succeed or fail.

For example the USER-MATCH Access Control Policy defined in the base
protocol could be redefined by inserting the following code in an
<access-control-code> element:

```
return resource.equalsHash(signer.user_name.bytes());
```

The <kind> parameters are also passed to the code, so the NODE-
MULTIPLE Access Control Policy could be implemented like this:

```
for (var i = 0; i < kind.max_node_multiple; i++) {
    if (resource.equalsHash(signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

[2](#). **Terminology**

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [[RFC2119](#)] .

   "SHOULD", "SHOULD NOT", "RECOMMENDED", and "NOT RECOMMENDED" are
   appropriate when valid exceptions to a general requirement are known
   to exist or appear to exist, and it is infeasible or impractical to
   enumerate all of them. However, they should not be interpreted as
   permitting implementors to fail to implement the general requirement
   when such failure would result in interoperability failure.

[3](#). **Processing**

   A peer receiving a configuration document containing one or more
   <access-control-code> elements, either by retrieving it from the
   configuration server or in a ConfigUpdateReq message, MUST reject
   this configuration if is not is not signed or if the signature
   verification fails.

   The Compact Relax NG Grammar for this element is:
   namespace acp = "http://implementers.org/access-control"

   parameter &= element acp:access-control-code {
     attribute name { xsd:string },
     xsd:base64Binary
   }?

   The "name" attribute defines the access control policy and can then
   be used in a <kind> element as if it was defined by IANA.

   If the <access-control-code> element is present in the namespace
   allocated to this specification, and the Access Control Policy is not
   natively implemented, then the code inside the element MUST be called
   for each DataValue found in a received StoreReq for a Kind that is
   defined with this access control policy. The content of the <access-
   control-code> element MUST be decoded using the base64 [[RFC4648](#)]
   encoding, uncompressed using gzip [[RFC1952](#)] then converted to
   characters using UTF-8. <access-control-code> elements that are not
   encoded using UTF-8, compressed with gzip or finally converted to the
   base64 format MUST be ignored. For each call to the code, the
   following ECMAScript objects, properties and functions MUST be
   available:

configuration.instance_name: The name of the overlay, as a String
   object.
configuration.topology_plugin: The overlay algorithm, as a String
   object.
configuration.node_id_length: The length of a NodeId in bytes, as a
   Number object.
configuration.kinds: An array of kinds (with the same definition
   than the kind object), indexed by id and eventually by name.
configuration.evaluate(String, String, String): A function that
   evaluates the first parameter as an XPath expression against the
   configuration element, and returns the result as a String object.
   The second parameter must contain a namespace prefix and the third
   parameter must contain a namespace.
kind.id: The id of the Kind associated with the entry, as a Number
   object.
kind.name: If the Kind associated with the entry is registered by
   IANA, contains the name as a String object. If not, this property
   is undefined.
kind.data_model: The name of the Data Model associated with the
   entry, as a String object.
kind.access_control: The name of the Access Control Policy
   associated with the entry, as a String object.
kind.max_count: The value of the max-count element in the
   configuration file, as a Number object.
kind.max_size: The value of the max-size element in the
   configuration file as a Number object.
kind.max_node_multiple: If the Access Control is MULTIPLE-NODE,
   contains the value of the max-node-multiple element in the
   configuration file, as a Number object. If not, this property is
   undefined.
kind.evaluate(String, String, String): A function that evaluates the
   first parameter as an XPath expression against the kind element,
   and returns the result as a String object. The second parameter
   must contain a namespace prefix and the third parameter must
   contain a namespace.
resource: An opaque object representing the Resource-ID, as an array
   of bytes.
resource.entries: An array of arrays of entry objects, with the
   first array level indexed by Kind-Id and kind names, and the
   second level indexed by index, key or nothing, depending on the
   data model of the kind. This permits to retrieve all the values
   of all Kinds stored at the same Resource-ID than the entry
   currently processed.
resource.equalsHash(Object...): A function that returns true if
   hashing the concatenation of the arguments according to the
   mapping function of the overlay algorithm is equal to the
   Resource-ID. Each argument is an array of bytes.

entry.index: If the Data Model is ARRAY, contains the index of the
    entry, as a Number object. If not, this property is undefined.
entry.key: If the Data Model is DICTIONARY, contains the key of the
    entry, as an array of bytes. If not, this property is undefined.
entry.storage_time: The date and time used to store the entry, as a
    Date object.
entry.lifetime: The validity for the entry in seconds, as a Number
    object.
entry.exists: Indicates if the entry value exists, as Boolean
    object.
entry.value: This property contains an opaque object that represents
    the whole data, as an array of bytes.
entry.signer.user_name: The rfc822Name stored in the certificate
    that was used to sign the request, as a String object.
entry.signer.node_id: The Node-ID stored in the certificate that was
    used to sign the request, as an array of bytes.

The properties SHOULD NOT be modifiable or deletable and if they are,
modifying or deleting them MUST NOT modify or delete the equivalent
internal values (in other words, the code cannot be used to modify
the elements that will be stored).

The value returned by the code is evaluated to true or false,
according to the ECMAScript rules. If the return value of one of the
call to the code is evaluated to false, then the StoreReq fails, the
state MUST be rolled back and an Error_Forbidden MUST be returned.

[4]. Security Considerations

Because the configuration document containing the ECMAScript code is
under the responsability of the same entity that will sign it, using
a scripting language does not introduce any additional risk if the
RELOAD implementers follow the rules in this document (no side effect
when modifying the parameters, only base classes of ECMAScript
implemented, etc...). It is even possible to deal with less than
perfect implementations as long as they do not accept a configuration
file that is not signed correctly. One way for the signer to enforce
this would be to deliberately send in a ConfigUpdate an incorrectly
signed version of the configuration file and blacklist all the nodes
that accepted it in a newly issued configuration file.

By permitting multiple overlay implementations to interoperate inside
one overlay, RELOAD helps build overlays that are not only resistant
to hardware or communication failures, but also to programmer errors.
Distributing the access control policy code inside the configuration
document reintroduces this single point of failure. To mitigate this
problem, new access control policies should be implemented natively

as soon as possible, but if all implementations uses the script as a
blueprint for the native code, an hidden bug can be duplicated. This
is why developers should implement new access control policies from
the normative text instead of using the code. That is anyway
probably not legal under most copyright laws so to help developers do
the right thing the code in the configuration is obfuscated by
compressing and encoding it as a base64 character string.

5. IANA Considerations

If this document is accepted as a standard track document this
section will request an URN in the "XML Namespaces" class of the
"IETF XML Registry" from IANA. Until this is done, implementions
should use the following URN:

http://implementers.org/access-control

6. References

6.1. Normative References

[RFC1952]   Deutsch, P., "GZIP file format specification version 4.3",
            RFC 1952, May 1996.

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4648]   Josefsson, S., "The Base16, Base32, and Base64 Data
            Encodings", RFC 4648, October 2006.

[I-D.ietf-p2psip-base]
            Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and
            H. Schulzrinne, "REsource LOcation And Discovery (RELOAD)
            Base Protocol", draft-ietf-p2psip-base-22 (work in
            progress), July 2012.

[ECMA-262]
            Ecma, "ECMAScript Language Specification 3rd Edition",
            December 2009.

6.2. Informative References

[I-D.ietf-p2psip-service-discovery]
            Maenpaa, J. and G. Camarillo, "Service Discovery Usage for
            REsource LOcation And Discovery (RELOAD)",
            draft-ietf-p2psip-service-discovery-06 (work in progress),

                    October 2012.

   [I-D.petithuguenin-vipr-reload-usage]
                Petit-Huguenin, M., Rosenberg, J., and C. Jennings, "A
                Usage of Resource Location and Discovery (RELOAD) for
                Public Switched Telephone Network (PSTN) Verification",
                draft-petithuguenin-vipr-reload-usage-04 (work in
                progress), March 2012.

   [I-D.ietf-p2psip-share]
                Knauf, A., Schmidt, T., Hege, G., and M. Waehlisch, "A
                Usage for Shared Resources in RELOAD (ShaRe)",
                draft-ietf-p2psip-share-00 (work in progress),
                October 2012.

Appendix A. Examples

A.1. Standard Access Control Policies

   This section shows the ECMAScript code that could be used to
   implement the standard Access Control Policies defined in
   [I-D.ietf-p2psip-base] .

A.1.1. USER-MATCH

```
String.prototype['bytes'] = function () {
    var bytes = [];
    for (var i = 0; i < this.length; i++) {
        bytes[i] = this.charCodeAt(i);
    }
    return bytes;
};

return resource.equalsHash(entry.signer.user_name.bytes());
```

A.1.2. NODE-MATCH

```
return resource.equalsHash(entry.signer.node_id);
```

A.1.3. USER-NODE-MATCH

```
    String.prototype['bytes'] = function () {
        var bytes = [];
        for (var i = 0; i < this.length; i++) {
            bytes[i] = this.charCodeAt(i);
        }
        return bytes;
    };

    var equals = function (a, b) {
        if (a.length !== b.length) return false;
        for (var i = 0; i < a.length; i++) {
            if (a[i] !== b[i]) return false;
        }
        return true;
    };

    return resource.equalsHash(entry.signer.user_name.bytes())
      && equals(entry.key, entry.signer.node_id);
```

A.1.4. NODE-MULTIPLE

```
    Number.prototype['width'] = function (w) {
        var bytes = [];
        for (var i = 0; i < w; i++) {
            bytes[i] = (this >>> ((w - i - 1) * 8)) & 255;
        }
        return bytes;
    };

    for (var i = 0; i < kind.max_node_multiple; i++) {
        if (resource.equalsHash(entry.signer.node_id, i.width(4))) {
            return true;
        }
    }
    return false;
```

A.2. Service Discovery Access Control Policy NODE-ID-MATCH

[I-D.ietf-p2psip-service-discovery] defines a specific Access Control
Policy (NODE-ID-MATCH) that need to access the content of the entry
to be written. If implemented as specified by this document, the
ECMAScript code would look something like this:

```
    /* Insert here the code from
       http://jsfromhell.com/classes/bignumber
     */

    var toBigNumber = function (node_id) {
```

```
   var bignum = new BigNumber(0);
   for (var i = 0; i < node_id.length; i++) {
       bignum = bignum.multiply(256).add(node_id[i]);
   }
   return bignum;
};

var checkIntervals = function (node_id, level, node, factor) {
   var size = new BigNumber(2).pow(128);
   var node = toBigNumber(node_id);
   for (var f = 0; f < factor; f++) {
       var temp = size.multiply(new BigNumber(f)
         .pow(new BigNumber(level).negate()));
       var min = temp.multiply(node.add(new BigNumber(f)
         .divide(factor)));
       var max = temp.multiply(node.add(new BigNumber(f + 1)
         .divide(factor)));
       if (node.compare(min) === -1 || node.compare(max) == 1
         || node.compare(max) == 0) return false;
   }
   return true;
};

var equals = function (a, b) {
   if (a.length !== b.length) return false;
   for (var i = 0; i < a.length; i++) {
       if (a[i] !== b[i]) return false;
   }
   return true;
};

var level = function (value) {
   var length = value[16] * 256 + value[17];
   return value[18 + length] * 256 + value[18 + length + 1];
};

var node = function (value) {
   var length = value[16] * 256 + value[17];
   return value[18 + length + 2] * 256
     + value[18 + length + 3];
};

var namespace = function (value) {
   var length = value[16] * 256 + value[17];
   return String.fromCharCode.apply(null,
     value.slice(18, length + 18));
};
```

```
  var branching_factor =
    kind.evaluate('/branching-factor',
    'redir', 'urn:ietf:params:xml:ns:p2p:redir');
  return equals(entry.key, entry.signer.node_id)
   && (!entry.exists || checkIntervals(entry.key,
     level(entry.value), node(entry.value),
     branching_factor))
   && (!entry.exists
     || resource.equalsHash(namespace(entry.value),
       level(entry.value), node(entry.value)));
```

Note that the code for the BigNumber object was removed from this
example, as the licensing terms are unclear. The code is available
at <http://jsfromhell.com/classes/bignumber> .

A.3. VIPR Access Control Policy

[I-D.petithuguenin-vipr-reload-usage] defines a specific Access
Control Policy. If implemented as specified by this document, the
ECMAScript code would look something like this:

```
  var equals = function (a, b) {
    if (a.length !== b.length) return false;
    for (var i = 0; i < a.length; i++) {
      if (a[i] !== b[i]) return false;
    }
    return true;
  };
  var length = configuration.node_id_length;
  return equals(entry.key.slice(0, length),
    entry.value.slice(4, length + 4))
    && equals(entry.key.slice(0, length), entry.signer.node_id);
```

A.4. ShaRe Access Control Policy USER-CHAIN-ACL

[I-D.ietf-p2psip-share] defines a new Access Control Policies, USER-
CHAIN-ACL. If implemented as specified by this document, the
ECMAScript code would look something like this:

```
   var pattern = kind.evaluate('/share:pattern',
     'share', 'urn:ietf:params:xml:ns:p2p:config-share');
   var username = entry.signer.user_name.match(/^([^@]+)@(.+)$/);
   var new_pattern = new RegExp(
     pattern.replace('$USER', username[1])
       .replace('$DOMAIN', username[2]));
   var length =  entry.value[0] * 256 + entry.value[1];
   var resource_name = String.fromCharCode.apply(null,
     entry.value.slice(2, length + 2));
   return new_pattern.test(resource_name);\n"));
```

   [[Note: the code is incomplete]]

Appendix B. Release notes

   This section must be removed before publication as an RFC.

B.1. Modifications between -05 and -04

   o Resurrected the draft.

B.2. Running Code Considerations

   o Reference Implementation and Access Control Policy script tester
      (<http://debian.implementers.org/testing/source/reload.tar.gz>).
      Marc Petit-Huguenin. Implements version -03.

B.3. TODO List

   o Finish the code for ShaRe.

Author's Address

   Marc Petit-Huguenin
   Impedance Mismatch

   Email: petithug@acm.org