

Network Working Group	M.P.H. Petit-Huguenin
Internet-Draft	Stonyfish, Inc.
Intended status: Standards Track	July 06, 2011
Expires: January 07, 2012	

Configuration of Access Control Policy in REsource LOcation And
Discovery (RELOAD) Base Protocol
draft-petithuguenin-p2psip-access-control-03

[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](#)

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet- Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on January 07, 2012.

[Copyright Notice](#)

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

[Table of Contents](#)

- *1. [Introduction](#)

- *2. [Terminology](#)
- *3. [Processing](#)
- *4. [Security Considerations](#)
- *5. [IANA Considerations](#)
- *6. [Acknowledgements](#)
- *7. [References](#)
 - *7.1. [Normative References](#)
 - *7.2. [Informative References](#)
- *Appendix A. [Examples](#)
 - *Appendix A.1. [Standard Access Control Policies](#)
 - *Appendix A.1.1. [USER-MATCH](#)
 - *Appendix A.1.2. [NODE-MATCH](#)
 - *Appendix A.1.3. [USER-NODE-MATCH](#)
 - *Appendix A.1.4. [NODE-MULTIPLE](#)
 - *Appendix A.2. [Service Discovery Access Control Policy NODE-ID-MATCH](#)
 - *Appendix A.3. [VIPR Access Control Policy](#)
- *Appendix B. [Release notes](#)
 - *Appendix B.1. [Modifications between -03 and -02](#)
 - *Appendix B.2. [Modifications between -02 and -01](#)
 - *Appendix B.3. [Modifications between -01 and -00](#)
 - *Appendix B.4. [Running Code Considerations](#)
 - *Appendix B.5. [TODO List](#)
- *[Author's Address](#)

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 [\[RFC2119\]](#).

[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.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.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.

signer.user_name: The rfc822Name stored in the certificate that was used to sign the request, as a String object.

signer.node_id: The Node-ID stored in the certificate that was used to sign the request, as 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.

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 responsibility 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 use 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 but 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, implementations should use the following URN:

<http://implementers.org/access-control>

6. Acknowledgements

This document was written with the xml2rfc tool described in [\[RFC2629\]](#).

7. References

7.1. Normative References

[RFC1952]	Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L.P. and G. Randers-Pehrson, "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" , Internet-Draft draft-ietf-p2psip-base-15, May 2011.
[ECMA-262]	Ecma, , "ECMAScript Language Specification 3rd Edition" , December 2009.

7.2. Informative References

[RFC2629]	Rose, M.T., "Writing I-Ds and RFCs using XML" , RFC 2629, June 1999.
[I-D.ietf-p2psip-service-discovery]	Maenpaa, J and G Camarillo, "Service Discovery Usage for REsource LOcation And

	Discovery (RELOAD) ", Internet-Draft draft-ietf-p2psip-service-discovery-03, July 2011.
[I-D.petithuguenin-vipr-reload-usage]	Rosenberg, J, Jennings, C and M Petit-Huguenin, " A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification ", Internet-Draft draft-petithuguenin-vipr-reload-usage-01, June 2011.
[I-D.knauf-p2psip-share]	Knauf, A, Hege, G, Schmidt, T and M Waehlich, " A Usage for Shared Resources in RELOAD (ShaRe) ", Internet-Draft draft-knauf-p2psip-share-00, March 2011.

[Appendix A.](#) Examples

[Appendix 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 [\[I-D.ietf-p2psip-base\]](#).

[Appendix 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(signer.user_name.bytes());
```

[Appendix A.1.2.](#) NODE-MATCH

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

[Appendix 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(signer.user_name.bytes())
    && equals(entry.key, signer.node_id);
```

[Appendix 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(signer.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

[[Note that base-15 still does not state exactly the length of i when concatenated in the hash input]]

[Appendix 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, 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 `BigInteger` object was removed from this example, as the licensing terms are unclear. The code is available at <http://jsfromhell.com/classes/bignumber>.

Appendix 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), signer.node_id);

```

Appendix B. Release notes

This section must be removed before publication as an RFC.

Appendix B.1. Modifications between -03 and -02

- *Moved the access-control-code element from the kind element to the configuration element so the code can be shared between kinds. A new "name" attribute is used to name the access control policy.

- *Added configuration object to pass information about the whole overlay.

- *Added evaluate functions to retrieve extensions parameters.
- *Renamed the signature attribute to signer.
- *Filled Security section.
- *Added temporary namespace to IANA section.
- *The content of the access-control-code is now UTF-8 encoded, compressed with gzip and converted back to characters with base64.
- *Fixed the implementation of the service discovery access control policy.
- *Added code for VIPR policy.

[Appendix B.2.](#) **Modifications between -02 and -01**

- *Made clear that an unsigned kind with this extension must be rejected.
- *Removed the kind.params array, and converted the max-count, max-size and max-node-multiple as Number objects. Fixed the examples.
- *Removed the parsing of extensions in the kind element. The former system did not work with namespaces or attributes, and the right solution (xpath) is probably too complex. The value of the parameters can still be manually mirrored in the script, so there is perhaps no need for the added complexity. Also fixed the examples.
- *Reference draft-p2psip-share instance of draft-p2psip-disco.
- *Added a "Running Code Considerations" section that contain the reference to the reference implementation and script tester.
- *Nits

[Appendix B.3.](#) **Modifications between -01 and -00**

- *Changed reference from JavaScript to ECMAScript.
- *Changed signature from equals() to equalsHash().
- *Fixed the examples following implementation.
- *Replaced automatic decoding of value by ECMAScript code.
- *Added the type of each property.

*Specified that the code cannot be used to modify the value stored.

Appendix B.4. Running Code Considerations

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

Appendix B.5. TODO List

*The access control policy in [ShaRe](#) [*I-D.knauf-p2psip-share*] requires an access to the list of all values stored at the same Resource-ID than the value currently verified. A major update of this specification is needed for this, so more time is needed to do it right.

Author's Address

Marc Petit-Huguenin Petit-Huguenin Stonyfish, Inc. EMail:
petithug@acm.org