SIP, a widely accepted Internet standard protocol for VoIP, is flexible enough to allow different extensions. The SIP-ALG (SIP Application Layer Gateway) in NAT/Firewall or, more broadly, the SBC (Session Border Controller) needs to be configured to handle properly SIP messages with extensions. We propose an XML-based SIP Processing Shim.
SCL                    February 2005

Configuration Language, by which one can define the legitimate SIP message components and the way for the SBC to handle SIP messages, legitimate or not.

Table of Contents

1. Introduction................................................... 2
2. Related Work................................................... 4
3. Terminology.................................................... 5
4. An illustration of SCL Usage................................... 5
5. The Schema for SCL............................................ 10
   Security Considerations....................................... 16
   References..................................................... 16
   Author’s Addresses............................................ 17
   Intellectual Property Statement............................ 18
   Disclaimer of Validity........................................ 18
   Copyright Statement......................................... 18

1. Introduction

Ideally, all the networking entities involved in the same protocol session should be synchronized for the target protocol, by which interoperability is secured. However, sometimes not all entities sitting in the middle of the protocol message flow need to be synchronized in every detail of the protocol messages. The SIP-ALG (Session Initiation Protocol [1] Application Layer Gateway) in NAT/Firewall or, more broadly, the SBC (Session Border Controller) for SIP is a good example of such entities, since they do not have to know all the details of the SIP message content for its main purpose of security provisioning.

The SIP-ALG implemented in NAT/Firewall devices handles SIP messages, but it may not recognize all of them since SIP messages may contain proprietary extensions or recently defined extensions which are not supported by the SIP-ALG implementation. In particular, IP PBX systems often include proprietary extensions, which are not understood by the SIP-ALG in NAT/Firewall, causing interoperability problems, because the SIP-ALG is frequently implemented by a different vendor.

Therefore, it is desirable and necessary to have a method to dynamically configure SIP-ALG or SBC to support handling of certain SIP messages or their components. We thus propose the SIP Processing Configuration Language (SCL) as a tool, designed specifically for generating dynamic configuration information of the SBC or the SIP-ALG in NAT/Firewall devices for handling SIP messages with extensions.
Note that SCL-generated configuration information is not intended to replace the whole SIP message handling logic or algorithms which should be implemented in the internal code of the SBC or anything close to the whole protocol specification. The SCL is also not to be a programming language to describe complex algorithms.

In this memo we focus on SIP-ALG on NAT/Firewall as a specific type of SBC since its behavior is relatively easier to define than general SBCs.

To design SCL, we need to answer several questions:
- Should we define proper actions on every portion of all the messages or cover just whatever portions that may require special instructions?
- What is the granularity of message handling? For example, per message, per header and body or per line?
- What kinds of actions are performed in handling SIP messages by SIP-ALG?

It is preferable to make the configuration as short and simple as possible. So we propose the configuration that covers just the portions that require special instructions. Those items that are clearly explained in the standard specifications and the message portions compliant with the standard specifications SHOULD NOT be repeated in the configuration. When the implementation of the SIP-ALG is out of date, the configuration MAY cover the portions that changed since the SIP-ALG implementation.

We propose that the scope of each processing action SHOULD be flexibly defined. For example, an action MAY be defined for a whole message and another action for a specific body component of the message. When two or more actions have overlapping scopes and there is a conflict among some of them, the action with a smaller scope supersedes the actions with larger scopes among the actions covering the specific scope. Such an example will be provided in section 4.

The main functionality of the SIP-ALG is to enable passage of SIP messages through the NAT/Firewall device, for which the SIP-ALG performs address and port translation for the private address, takes part in authentication for the security of the VoIP session in some cases, in particular, when it is implemented as a B2B UA (back-to-back user agent). A SIP-ALG implemented as a B2B UA decodes an incoming SIP message and generates a new outgoing message. We assume the SIP-ALG, on receiving a SIP message, checks certain things such as integrity, modifies the message partially if necessary, and then forwards the message. The SIP-ALG may discard the SIP message when check-up of the message fails. The actions to be defined in the configuration instruct the SIP-ALG what should be done or can be done before forwarding the message.
The following actions are defined:
- KEEP-AS-IS
- TRANSLATE
- REMOVE
- IGNORE-MSG
- RETURN-ERROR

KEEP-AS-IS: The covered portion in the message MUST NOT be modified.

TRANSLATE: The address or the port or the both MAY be translated as necessary in the covered message portion.

REMOVE: The corresponding portion of the message MUST be removed.

IGNORE-MSG: The whole message MUST be silently ignored.

RETURN-ERROR: The whole message MUST be discarded and an error message should be sent to the sender of the discarded message.

The SCL is based on XML (eXtensible Markup Language) [2] and XML schema [3]. The processing configuration data becomes an XML document and the SCL is the XML schema for the XML documents.

2. Related Work

The CPL (Call Processing Language) [4] is a tool to describe general application-level call service logics (e.g. call forwarding for each callee) for the SIP servers (Proxies) and User Agents. On the other hand, The SCL (SIP Processing Configuration Language) is focused on processing by the SBC (session border controller) or SIP-ALG, in particular configuring SBCs how to handle the SIP messages containing extensions some of which may be proprietary. A major goal of the SCL is to allow interoperation of SBC (or SIP-ALG) with other SIP devices with proprietary extensions. The SCL has more limited capabilities than the CPL. For example, the SCL cannot be used to make a call or transfer a call. On the other hand, the SCL can describe what extensions are allowed or forbidden down to the level of the header field's attribute value or the MIME type of the body, which is not supported by the CPL.

The XCAP (XML Configuration Access Protocol) [5] proposed in the SIMPLE Working Group is a protocol based on HTTP to access and manipulate configuration data written with XML, which allows access or modification on element and attribute levels. Certainly the SIP Processing Configuration Data written with the SCL can be accessed and manipulated with the XCAP. The XCAP requires a number of specifications for the XML documents to be managed by the XCAP.
server: Application Unique ID (AUID), XML Schema, MIME Type, Validation Constraints, Data Semantics, Naming Conventions, Resource Interdependences and Authorization Policies. The current memo provides the XML schema and the data semantics but compliance with the XCAP requirements is out of scope of this memo.

D. Petrie proposed a framework for SIP User Agent profile delivery [6], in which the profiles are categorized into four types, "user", "device", "application" and "local network" and a new SIP event package [7] is defined to solve enrollment and profile change notification. The framework is not bound to any specific profile data set format. D. Petrie and S. Lawrence stepped forward to propose a general data set format in [8], in which profile data are grouped into four categories: "forbid", "set all", "set one" and "set any". A data set should define a new XML namespace to scope all of the properties defined in the data set. Their work focused on the SIP User Agent profile data, thus they listed candidate data sets such as SIP Protocol data set, media data set, and HTTP protocol data set.

SNMP [9] and COPS [10] are two protocols often used for configuration data delivery and MIB [9] and PIB [10] are used for representing configuration data with each protocol respectively. The XML-based configuration data format has several advantages against MIB and PIB. It is easier to input an XML document and manage the document in a central server than to set entry by entry of MIB or PIB for each device using NMS. XML, easy and flexible for extension as it is supposed to be, can express complex data more easily than MIB or PIB. Furthermore, it fits nicely into the new profile or configuration management mechanisms such as XCAP which are designed to serve SIP-based systems well.

3. 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 RFC-2119 [11].

AND The boolean logic operator returning true only when both of the two operands are true.

NOT The boolean logic operator that flips the value of the operand.

4. An illustration of SCL Usage

By seeing an example of processing configuration, we can get a sense of what the configuration data looks like and what processing can be defined with the SCL. Listing 1 and 2 show two examples of SIP
messages with proprietary extensions. The extension portions are enclosed in rectangular blocks. Block (1) and block (3) are proprietary header fields. Block (2) consists of proprietary attributes added to a standard header field. Block (4) is a proprietary body component as a MIME type. Block (5) is another proprietary body component as a MIME type containing two sub-components, block (6) and block (7). Block (6) is a standard body component, which is a SDP (Session Description Protocol [12]) message. Block (7) is a proprietary MIME type. All the proprietary body components and their sub-components are taking the same MIME type "application/X-VENDOREXTv2" or "multipart/X-VENDOREXTv2".

REGISTER sip:registrar.mycompany.com SIP/2.0
Via: SIP/2.0/UDP 10.2.20.31;5060;branch=abcdefghijklmn
Call-ID: bae66456328@10.2.20.1
CSeq: 11768 REGISTER
From: <sip:alice@10.2.20.31:5060>;tag=zyx123456789012345
To: <sip:alice@10.2.20.31:5060>
Contact: <sip:alice@10.2.20.31:5060>
Max-Forwards: 70
Expires: 65535
DeviceType: SipSoftPhone(VENDOREXTv2) <----(1)
Authorization: Digest realm="sipserver0004",
nonce="1345654d6876186rrf", opaque="qjfsksdjjj94589389widsk",
qop="auth", algorithm=MD5, cnonce="045015", nc=00000001,
uri="sip:10.2.20.1:5060", username="sipuser",
response="lkofiosiodfjiosj458495893rosdk",
X-param1="948378833CF932AF6",
X-param2="f8eur8u4390ei0we208es98e9w" <----(2)
X-SERVICE:CONFIGURATION <----(3)
Content-Type: application/X-VENDOREXTv2 <----(4)
Content-Length: 397
TerminalType=terminal
TerminalID=MAC:00004C01044B
DisplayType=24x3
Functionkey=1:feature 2:recall 3:redial 4:hook 5:speaker 6:hold
Lamp=1:call-indication 1:message-waiting

[Listing 1. An example SIP message - REGISTER]

INVITE sip:bob@mycompany.com SIP/2.0
Via: SIP/2.0/UDP 10.2.20.1;5060;branch=abcdefghijklmn
From: <sip:alice@10.2.20.1:5060>;tag=zyx123456789012345
We assume that all these proprietary extensions (note that block (6) is not proprietary.) are not known to the SIP-ALG implementation and the SIP-ALG system administrator chooses the following policy, somewhat arbitrary for illustration purpose.

(i) Remove the proprietary header field "DeviceType".
(ii) Keep the proprietary header field "X-SERVICE".
(iii) Keep the additional attributes "X-param1" and "X-param2" in the standard header field "Digest".
(iv) Keep the body components of MIME type "application/X-VENDOREXTv2" and "multipart/X-VENDOREXTv2".
(v) Translate the SDP message within the body extension of "multipart/X-VENDOREXT2" in the INVITE messages.
(vi) Silently ignore any message of unknown type or any message with
unknown header fields or body components.
(vii) Do not allow the REGISTRATION messages more than once per
minute per sender. Discard any excessive REGISTRATION messages.
(viii) The REGISTRATION message must be less than 1500 bytes long.
Otherwise, the message is discarded.
(ix) Allow REGISTRATION messages only if it includes the
"Authorization" header field with the value "Digest". Otherwise,
discard the message.

The above policies can be represented as a SCL-based configuration
shown in Listing 3.

<?xml version="1.0"?>
<SCL  xmlns="http://ns.ietf.org/scl"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://ns.ietf.org/scl/scl.xsd">
  <PROCESSING-CONFIG>
    <HEADER name="DeviceType" action="REMOVE"/>                <---(8)
    <HEADER name="X-SERVICE" action="KEEP-AS-IS"/>            <---(9)
    <HEADER name="Digest"/>                                   <---(10)
    <ATTRIBUTE name="X-param1" action="KEEP-AS-IS"/>          <---(11)
    <ATTRIBUTE name="X-param2" action="KEEP-AS-IS"/>          <---(12)
    </HEADER>                                                  <---(13)
    <BODY name="application/X-VENDOREXTv2" action="KEEP-AS-IS"/> <---(14)
    <BODY name="multi/X-VENDOREXTv2" action="KEEP-AS-IS"/>     <---(15)
    <MESSAGE name="INVITE">                                    <---(16)
    <SUBBODY name="application/sdp" action="TRANSLATE"/>      <---(18)
    </MESSAGE>                                                  <---(19)
    <MESSAGE name="" legitimate="false" action="IGNORE-MSG"/>   <---(21)
    <MESSAGE name="REGISTER">
      <CONDITION satisfy="false" action="IGNORE-MSG">
        <msg-min-interval> 60 </msg-min-interval>
        <max-length> 1500 </max-length>
      </CONDITION>
      <INCLUDE satisfy="false" action="RETURN-ERROR">
        <HEAD name="Authorization" value="Digest"/>
      </INCLUDE>
    </MESSAGE>
  </PROCESSING-CONFIG>
</SCL>
In Listing 3, lines (8) and (9) express the policies (i) and (ii) respectively. Both lines are standalone: they are not nested within any MESSAGE element, which means they are applied to header fields of "DeviceType" and "X-Service" appearing in any message. Line (10) to (13) express the policy (iii). Since line (10) is standalone, the policy is again effective within the header field "Digest" appearing in any message. Line (14) and (15) express the policy (iv), effective in any message. Line (16) to (20) express the policy (v), in which the policy on the SDP message is effective only within a body component "multipart/VENDOREXTv2" appearing in the INVITE message because line (18) is nested within line (17) that itself is nested within line (16). Lines (17) and (15) overlap but do not conflict because line (17) does not specify the action, which makes the policy in line (15) still effective for the body component "multipart/VENDOREXTv2" in the INVITE message. On the other hand, the policy of line (15) and that of line (18) conflict if the SDP message appears within "multipart/VENDOREXTv2" of the INVITE message but the policy of line (18) overrides that of line (15) because the former has a more narrow scope than that of line (15). Line (21) expresses the policy (vi). The value of the message name is null, which makes this policy effective for any message. The configuration data does not fully specify whether a message is legitimate or not in details. It is intended to specify only a subset of the message formats that are required to specify the relevant processing rules. For example, line (11) and (12) means the header field "Digest" can contain two additional attributes "X-param1" and "X-param2", which may not be known to the SIP processor previously. When the message formats and processing rules programmed into the processor conflict with those in the configuration data, the latter SHOULD override the former. Back to line (21), if the processor decides any message does not fit into any allowed message format specified (partially) in the configuration data and its internal program, the message is deemed illegitimate and the processor follows the policy of line (21), ignoring the message silently. Line (23) to (26) represent the policy (vii) and (viii). The CONDITION element is a container element for a set of conditions, each of which is included as a child element. The logical AND relation is given for the multiple conditions included under one CONDITION element. In the example, the aggregated condition is '(msg-max-interval == 1 per minute) AND (max-length == 1500 bytes)'. The attribute "satisfy" of the CONDITION element with its value "false" switches the logical value of the aggregated condition, which turns the condition into 'NOT ((msg-max-interval == 1 per minute) AND (max-length == 1500 bytes))'. Line (27) to (29) represent the policy (ix). The INCLUDE element is also a container element specifying the sub-components of the parent element. In the example, the parent element
is the REGISTRATION message and the sub-component is the header field 'Authorization'.

5. The Schema for SCL

The SCL is an XML schema for the configuration generation, which is defined in Listing 4. The elements in the SCL have a relatively simple tree-like hierarchy shown in Figure 1. In each tree of Figure 1, the left element can contain as its child elements none or multiple of each element in the right.

```
PROCESSING-CONFIG --> MESSAGE(S)
                    --> HEADER(S)
                    --> BODY(S)
MESSAGE           --> CONDITION(S)
                    --> INCLUDE(S)
                    --> HEADER(S)
                    --> BODY(S)
CONDITION         --> msg-min-interval
                    --> max-length
INCLUDE           --> HEADER(S)
                    --> BODY(S)
HEADER            --> ATTRIBUTE(S)
BODY              --> SUBBODY(S)
```

[Figure 1. Parent-children relationship between SCL elements]

The PROCESSING-CONFIG element has with no meaning on processing policies or rules but is a container element for the whole processing configuration data. It is ignored for the description of other elements in the below. There are five levels of message components: MESSAGE, HEADER, BODY, SUBBODY and ATTRIBUTE. The configuration processing data takes a kind of object-oriented structure, that is, a message component is first defined and then followed by the policies to handle the component. Note that HEADER means a header field rather a whole header and BDOY can be a whole body or a component of the body. BODY and SUBBODY are each a MIME type data. ATTRIBUTE represents the individual attribute in the header field. Between these five elements, the parent element, if any, limits the scope of the child element. Therefore, an ATTRIBUTE element can be a child element of a HEADER element but not of a BODY element. The SUBBODY
elements can exist only as nested in the BODY elements. To identify the "From" header field of the "INVITE" message, a HEADER element with name "From" is nested within a MESSAGE element with name "INVITE". Then, the policy defined for the HEADER element is applied only to the header field in the "INVITE" message. Only the three types of elements, MESSAGE, HEADER and BODY, can appear below the PROCESSING-CONFIG element, or without a parent element, which are called standalone elements. The standalone HEADER element contains the policy for the specified type of the header field in any message and the same is with the standalone BODY element. A standalone HEADER element is same as having a parent of the MESSAGE element with null name.

Any number of standalone elements MAY appear and the appearance order has no significance. For example, it is fine to list two MESSAGE elements, one BODY element and then another MESSAGE element. An empty configuration, i.e., no element is fine, too. Actually the appearance order of the children elements does not matter in any parent–children relationship currently and multiple elements of the same type MAY appear under the same parent.

The MESSAGE element MAY have two other types of child elements: CONDITION and INCLUDE. The CONDITION element is a container element having child elements representing conditions other than the message format or message components. In this memo, two elements of such conditions are defined: "msg-min-interval" and "max-length", where the former defines the minimum time interval in seconds between the consecutive messages of the same type from the same sender and the latter is the maximum message length in bytes. The INCLUDE element is another container element having HEADER elements and BODY elements as child elements. It defines what components the message should include.

Each element MAY have a number of attributes, some mandatory and others optional. For the MESSAGE element, the "name" attribute is mandatory and its value is the SIP message type such as "INVITE", "REGISTER" or "200". If the name has the null value, i.e. "", it means all messages. The "name" attribute MAY be followed by the "action" attribute or the "legitimate" attribute. If the "legitimate" attribute, whose value is of xsd:boolean type, is "true" (or "false"), it indicates that the action specified in the "action" attribute MUST be taken if the processor recognizes the message as legitimate (or not). If the "legitimate" attribute appears, the "action" attribute MUST NOT be skipped.

The HEADER element MAY have the "value" attribute as optional in addition to the three attributes, "name", "legitimate" and "action". If the "value" attribute appears, its value narrows down the applied header fields. The value of the header field is what is appended to the header field name and a colon in the SIP message header.
The BODY element has three attributes: "name", "legitimate" and "action", and their semantics are the same as those of the MESSAGE element. The SUBBODY element has the same set of attributes as the BODY element. The ATTRIBUTE element has three attributes: "name", "value" and "action", where only the "name" attribute is mandatory.

The CONDITION element has two attributes: "satisfy" and "action", where the former is of xsd:boolean type. If the value of the "satisfy" attribute is "false" (or "true"), the specified action is taken when the conditions specified by the child elements of the CONDITION element are not satisfied (or are satisfied).

The INCLUDE element has the same two attributes as the CONDITION element. If the value of the "satisfy" attribute is "true", the specified action is taken when the message includes all the components specified by the child elements of the INCLUDE element. If "false", the specified action is taken when any of the specified components is missing in the message.

Note that the action IGNORE-MSG or RETURN-ERROR can be specified not only for the MESSAGE element but also for the elements of HEADER, BODY, SUBBODY or ATTRIBUTE. If any subcomponent of the message meets the condition for IGNORE-MSG, it overrides other policies such as TRANSLATE and KEEP-AS-IS on the message or other components of the message because the message is discarded as a whole. If the processing requires the two actions RETURN-ERROR and IGNORE-MSG simultaneously on the same message, RETURN-ERROR is taken. If there is conflict of actions to be taken on the same scope among TRANSLATE, KEEP-AS-IS and REMOVE, it is considered as an error and thus the processor MUST take the RETURN-ERROR action and notify the error to the management system, if any. The format and the content of the error message for the RETURN-ERROR action are out of scope of this memo. It is going to be covered later.

<?xml version="1.0"?>
xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns="http://ns.ietf.org/scl"
   targetNamespace="http://ns.ietf.org/scl"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>
   <xsd:element name="PROCESSING-CONFIG" type="ConfigType"/>

   <xsd:complexType name="ConfigType">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
      
      </xsd:complexType>
<xsd:element name="CONDITION" type=ConditionType/>
</xsd:choice>

</xsd:sequence>
</xsd:complexType>

</xsd:complexType>

<xsd:complexType   name="BodyType">
<xsd:choice>
<xsd:sequence>
 <xsd:attribute name="name"    type="xsd:string" use="required"/>
 <xsd:attribute name="action"  type="ActionType" use="optional"/>
</xsd:sequence>

<xsd:sequence>
 <xsd:attribute name="name"    type="xsd:string" use="required"/>
 <xsd:attribute name="legitimate"  type="xsd:boolean" use="required"/>
 <xsd:attribute name="action"  type="ActionType" use="required"/>
</xsd:sequence>
</xsd:choice>

<xsd:sequence minOccurs="0" maxOccurs="unbounded">
 <xsd:element name="SUBBODY" type=SubbodyType/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType   name="SubbodyType">
<xsd:choice>
<xsd:sequence>
 <xsd:attribute name="name"    type="xsd:string" use="required"/>
</xsd:sequence>
</xsd:choice>
</xsd:complexType>

Shim Expires - August 2005 [Page 14]
<xsd:complexType name="ActionType">
  <xsd:enumeration value="KEEP-AS-IS"/>
  <xsd:enumeration value="TRANSLATE"/>
  <xsd:enumeration value="REMOVE"/>
  <xsd:enumeration value="IGNORE-MSG"/>
  <xsd:enumeration value="RETURN-ERROR"/>
</xsd:complexType>

<xsd:complexType name="ConditionType">
  <xsd:sequence>
    <xsd:attribute name="satisfy" type="xsd:boolean" use="required"/>
    <xsd:attribute name="action" type="ActionType" use="required"/>
  </xsd:sequence>
  <xsd:all>
    <xsd:element name="msg-min-interval" type="xsd:integer" minOccurs="0"/>
    <xsd:element name="max-length" type="xsd:integer" minOccurs="0"/>
  </xsd:all>
</xsd:complexType>

<xsd:complexType name="IncludeType">
  <xsd:sequence>
    <xsd:attribute name="satisfy" type="xsd:boolean" use="required"/>
    <xsd:attribute name="action" type="ActionType" use="required"/>
  </xsd:sequence>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="HEADER" type="HeaderType"/>
    <xsd:element name="BODY" type="BodyType"/>
  </xsd:choice>
</xsd:complexType>
Security Considerations

The SIP-ALG or SBC plays an important role for network security. By compromising the SIP processing configuration data, an attacker may use SIP messages to transfer sensitive but unauthorized information in the form of proprietary extensions or block normal operation of the SIP-based services. Therefore, access to the SIP processing configuration data for the SIP-ALG or SBC MUST be secured. If any remote access or transfer mechanism for the configuration data, stronger attention should be paid to the security. Such protection mechanisms are independent of the configuration data format proposed in the memo or can be provisioned independently of the data format and are out of scope of this memo.

References


Author's Addresses

Eunsoo Shim
NEC Labs America, Inc.
4 Independence Way
Princeton, NJ 08540 USA
Phone: 1-609-951-2909
Email: eunsoo@nec-labs.com

Kyungtae Kim
NEC Labs America, Inc.
4 Independence Way
Princeton, NJ 08540 USA
Phone: 1-609-951-2988
Email: kyungtae@nec-labs.com

Rauf Izmailov
NEC Labs America, Inc.
4 Independence Way
Princeton, NJ 08540 USA
Phone: 1-609-951-2454
Email: rauf@nec-labs.com
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.