Network Working Group
Request for Comments: 5360
Category: Standards Track
October 2008

J. Rosenberg
Cisco Systems
G. Camarillo, Ed.
Ericsson
D. Willis
Unaffiliated

A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)

Status of This Memo

Abstract

SIP supports communications for several services, including real-time audio, video, text, instant messaging, and presence. In its current form, it allows session invitations, instant messages, and other requests to be delivered from one party to another without requiring explicit consent of the recipient. Without such consent, it is possible for SIP to be used for malicious purposes, including amplification and DoS (Denial of Service) attacks. This document identifies a framework for consent-based communications in SIP.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [RFC3261] supports communications for several services, including real-time audio, video, text, instant messaging, and presence. This communication is established by the transmission of various SIP requests (such as INVITE and MESSAGE [RFC3428]) from an initiator to the recipient with whom communication is desired. Although a recipient of such a SIP request can reject the request, and therefore decline the session, a network of SIP proxy servers will deliver a SIP request to its recipients without their explicit consent.

Receipt of these requests without explicit consent can cause a number of problems. These include amplification and DoS (Denial of Service) attacks. These problems are described in more detail in a companion requirements document [RFC4453].

This specification defines a basic framework for adding consent-based communication to SIP.

2. Definitions and 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 [RFC2119].

Recipient URI: The Request-URI of an outgoing request sent by an entity (e.g., a user agent or a proxy). The sending of such request can have been the result of a translation operation.

Relay: Any SIP server, be it a proxy, B2BUA (Back-to-Back User Agent), or some hybrid, that receives a request, translates its Request-URI into one or more next-hop URIs (i.e., recipient URIs), and delivers the request to those URIs.

Target URI: The Request-URI of an incoming request that arrives to a relay that will perform a translation operation.

Translation logic: The logic that defines a translation operation at a relay. This logic includes the translation's target and recipient URIs.

Translation operation: Operation by which a relay translates the Request-URI of an incoming request (i.e., the target URI) into one or more URIs (i.e., recipient URIs) that are used as the Request-URIs of one or more outgoing requests.

3. Relays and Translations

Relays play a key role in this framework. A relay is defined as any SIP server, be it a proxy, B2BUA (Back-to-Back User Agent), or some hybrid, that receives a request, translates its Request-URI into one or more next-hop URIs, and delivers the request to those URIs. The Request-URI of the incoming request is referred to as 'target URI' and the destination URIs of the outgoing requests are referred to as 'recipient URIs', as shown in Figure 1.

```
                   +---------------+  recipient URI
                   |               |---------------->
                   |               |
       target URI  |  Translation  |      [...]
     ------------->|   Operation   |
                   |               |  recipient URI
                   |               |---------------->
                   +---------------+
```

Figure 1: Translation Operation

Thus, an essential aspect of a relay is that of translation. When a relay receives a request, it translates the Request-URI (target URI) into one or more additional URIs (recipient URIs). Through this translation operation, the relay can create outgoing requests to one or more additional recipient URIs, thus creating the consent problem.

The consent problem is created by two types of translations: translations based on local data and translations that involve amplifications.

Translation operations based on local policy or local data (such as registrations) are the vehicle by which a request is delivered directly to an endpoint, when it would not otherwise be possible to. In other words, if a spammer has the address of a user, 'sip:user@example.com', it cannot deliver a MESSAGE request to the UA (user agent) of that user without having access to the registration data that maps 'sip:user@example.com' to the user agent on which that user is present. Thus, it is the usage of this registration data, and more generally, the translation logic, that is expected to be authorized in order to prevent undesired communications. Of course, if the spammer knows the address of the user agent, it will be able to deliver requests directly to it.

Translation operations that result in more than one recipient URI are a source of amplification. Servers that do not perform translations, such as outbound proxy servers, do not cause amplification. On the other hand, servers that perform translations (e.g., inbound proxies authoritatively responsible for a SIP domain) may cause amplification if the user can be reached at multiple endpoints (thereby resulting in multiple recipient URIs).

Figure 2 shows a relay that performs translations. The user agent client in the figure sends a SIP request to a URI representing a resource in the domain 'example.com' (sip:resource@example.com). This request can pass through a local outbound proxy (not shown), but eventually arrives at a server authoritative for the domain 'example.com'. This server, which acts as a relay, performs a translation operation, translating the target URI into one or more recipient URIs, which can (but need not) belong to the domain 'example.com'. This relay can be, for instance, a proxy server or a URI-list service [RFC5363].

```
                                                    +-------+
                                                    |       |
                                                   >|  UA   |
                                                  / |       |
                                                 /  +-------+
                                                /
                                               /
                  +-----------------------+   /
                  |                       |  /
    +-----+       |         Relay         | /       +-------+
    |     |       |                       |/        |       |
    | UA  |------>|                       |-------->| Proxy |
    |     |       |+---------------------+|\        |       |
    +-----+       ||     Translation     || \       +-------+
                  ||        Logic        ||  \
                  |+---------------------+|   \       [...]
                  +-----------------------+    \
                                                \
                                                 \  +-------+
                                                  \ |       |
                                                   >| B2BUA |
                                                    |       |
                                                    +-------+
```

Figure 2: Relay Performing a Translation

This framework allows potential recipients of a translation to agree to be actual recipients by giving the relay performing the translation permission to send them traffic.

4. Architecture

Figure 3 shows the architectural elements of this framework. The manipulation of a relay's translation logic typically causes the relay to send a permission request, which in turn causes the recipient to grant or deny the relay permissions for the translation. Section 4.1 describes the role of permissions at a relay. Section 4.2 discusses the actions taken by a relay when its translation logic is manipulated by a client. Section 4.3 discusses store-and-forward servers and their functionality. Section 4.4 describes how potential recipients can grant a relay permissions to add them to the relay's translation logic. Section 4.5 discusses which entities need to implement this framework.

```
                  +-----------------------+ Permission +-------------+
                  |                       |  Request   |             |
   +--------+     |         Relay         |----------->| Store & Fwd |
   |        |     |                       |            |   Server    |
   | Client |     |                       |            |             |
   |        |     |+-------+ +-----------+|            +-------------+
   +--------+     ||Transl.| |Permissions||                   |
       |          ||Logic  | |           ||        Permission |
       |          |+-------+ +-----------+|         Request   |
       |          +-----------------------+                   V
       |               ^           ^                   +-------------+
       | Manipulation  |           |  Permission Grant |             |
       +---------------+           +-------------------|  Recipient  |
                                                       |             |
                                                       +-------------+
```

Figure 3: Reference Architecture

4.1. Permissions at a Relay

Relays implementing this framework obtain and store permissions associated to their translation logic. These permissions indicate whether or not a particular recipient has agreed to receive traffic at any given time. Recipients that have not given the relay permission to send them traffic are simply ignored by the relay when performing a translation.

In principle, permissions are valid as long as the context where they were granted is valid or until they are revoked. For example, the permissions obtained by a URI-list SIP service that distributes MESSAGE requests to a set of recipients will be valid as long as the URI-list SIP service exists or until the permissions are revoked.

Additionally, if a recipient is removed from a relay's translation logic, the relay SHOULD delete the permissions related to that recipient. For example, if the registration of a contact URI expires or is otherwise terminated, the registrar deletes the permissions related to that contact address.

It is also RECOMMENDED that relays request recipients to refresh their permissions periodically. If a recipient fails to refresh its permissions for a given period of time, the relay SHOULD delete the permissions related to that recipient.

> This framework does not provide any guidance for the values of the refreshment intervals because different applications can have different requirements to set those values. For example, a relay dealing with recipients that do not implement this framework may choose to use longer intervals between refreshes. The refresh process in such recipients has to be performed manually by their users (since the recipients do not implement this framework), and having too short refresh intervals may become too heavy a burden for those users.

4.2. Consenting Manipulations on a Relay's Translation Logic

This framework aims to ensure that any particular relay only performs translations towards destinations that have given the relay permission to perform such a translation. Consequently, when the translation logic of a relay is manipulated (e.g., a new recipient URI is added), the relay obtains permission from the new recipient in order to install the new translation logic. Relays ask recipients for permission using MESSAGE [RFC3428] requests.

For example, the relay hosting the URI-list service at 'sip:friends@example.com' performs a translation from that target URI to a set of recipient URIs. When a client (e.g., the administrator of that URI-list service) adds 'bob@example.org' as a new recipient URI, the relay sends a MESSAGE request to 'sip:bob@example.org' asking whether or not it is OK to perform the translation from 'sip:friends@example.com' to 'sip:bob@example.org'. The MESSAGE request carries in its message body a permission document that describes the translation for which permissions are being requested and a human-readable part that also describes the translation. If the answer is positive, the new translation logic is installed at the relay. That is, the new recipient URI is added.

> The human-readable part is included so that user agents that do not understand permission documents can still process the request and display it in a sensible way to the user.

The mechanism to be used to manipulate the translation logic of a particular relay depends on the relay. Two existing mechanisms to manipulate translation logic are XML Configuration Access Protocol (XCAP) [RFC4825] and REGISTER transactions.

> Section 5 uses a URI-list service whose translation logic is manipulated with XCAP as an example of a translation, in order to specify this framework. Section 5.10 discusses how to apply this framework to registrations, which are a different type of translation.

In any case, relays implementing this framework SHOULD have a means to indicate that a particular recipient URI is in the states specified in [RFC5362] (i.e., pending, waiting, error, denied, or granted).

4.3. Store-and-Forward Servers

When a MESSAGE request with a permission document arrives to the recipient URI to which it was sent by the relay, the receiving user can grant or deny the permission needed to perform the translation. However, the receiving user may not be available when the MESSAGE request arrives, or it may have expressed preferences to block all incoming requests for a certain time period. In such cases, a store-and-forward server can act as a substitute for the user and buffer the incoming MESSAGE requests, which are subsequently delivered to the user when he or she is available again.

There are several mechanisms to implement store-and-forward message services (e.g., with an instant message to email gateway). Any of these mechanisms can be used between a user agent and its store-and-forward server as long as they agree on which mechanism to use. Therefore, this framework does not make any provision for the interface between user agents and their store-and-forward servers.

> Note that the same store-and-forward message service can handle all incoming MESSAGE requests for a user while they are offline, not only those MESSAGE requests with a permission document in their bodies.

Even though store-and-forward servers perform a useful function and they are expected to be deployed in most domains, some domains will not deploy them from the outset. However, user agents and relays in domains without store-and-forward servers can still use this consent framework.

When a relay requests permissions from an offline user agent that does not have an associated store-and-forward server, the relay will obtain an error response indicating that its MESSAGE request could not be delivered. The client that attempted to add the offline user to the relay's translation logic will be notified about the error (e.g., using the Pending Additions event package [RFC5362]). This client MAY attempt to add the same user at a later point, hopefully when the user is online. Clients can discover whether or not a user is online by using a presence service, for instance.

4.4. Recipients Grant Permissions

Permission documents generated by a relay include URIs that can be used by the recipient of the document to grant or deny the relay the permission described in the document. Relays always include SIP URIs and can include HTTP [RFC2616] URIs for this purpose. Consequently, recipients provide relays with permissions using SIP PUBLISH requests or HTTP GET requests.

4.5. Entities Implementing This Framework

The goal of this framework is to keep relays from executing translations towards unwilling recipients. Therefore, all relays MUST implement this framework in order to avoid being used to perform attacks (e.g., amplification attacks).

This framework has been designed with backwards compatibility in mind so that legacy user agents (i.e., user agents that do not implement this framework) can act both as clients and recipients with an acceptable level of functionality. However, it is RECOMMENDED that user agents implement this framework, which includes supporting the Pending Additions event package specified in [RFC5362], the format for permission documents specified in [RFC5361], and the header fields and response code specified in this document, in order to achieve full functionality.

The only requirement that this framework places on store-and-forward servers is that they need to be able to deliver encrypted and integrity-protected messages to their user agents, as discussed in Section 7. However, this is not a requirement specific to this framework but a general requirement for store-and-forward servers.

5. Framework Operations

This section specifies this consent framework using an example of the prototypical call flow. The elements described in Section 4 (i.e., relays, translations, and store-and-forward servers) play an essential role in this call flow.

Figure 4 shows the complete process to add a recipient URI ('sip:B@example.com') to the translation logic of a relay. User A attempts to add 'sip:B@example.com' as a new recipient URI to the translation logic of the relay (1). User A uses XCAP [RFC4825] and the XML (Extensible Markup Language) format for representing resource lists [RFC4826] to perform this addition. Since the relay does not have permission from 'sip:B@example.com' to perform translations towards that URI, the relay places 'sip:B@example.com' in the pending state, as specified in [RFC5362].

```
   A@example.com        Relay       B's Store & Fwd   B@example.com
                                         Server

         |(1) Add Recipient                |                |
         |    sip:B@example.com            |                |
         |--------------->|                |                |
         |(2) HTTP 202 (Accepted)          |                |
         |<---------------|                |                |
         |                |(3) MESSAGE sip:B@example        |
         |                |    Permission Document          |
         |                |--------------->|                |
         |                |(4) 202 Accepted|                |
         |                |<---------------|                |
         |(5) SUBSCRIBE   |                |                |
         |    Event: pending-additions     |                |
         |--------------->|                |                |
         |(6) 200 OK      |                |                |
         |<---------------|                |                |
         |(7) NOTIFY      |                |                |
         |<---------------|                |                |
         |(8) 200 OK      |                |                |
         |--------------->|                |                |
         |                |                |                |User B goes
         |                |                |                |  online
         |                |                |(9) Request for |
         |                |                |  stored messages
         |                |                |<---------------|
         |                |                |(10) Delivery of|
         |                |                |  stored messages
         |                |                |--------------->|
         |                |(11) PUBLISH uri-up              |
         |                |<--------------------------------|
         |                |(12) 200 OK     |                |
         |                |-------------------------------->|
         |(13) NOTIFY     |                |                |
         |<---------------|                |                |
         |(14) 200 OK     |                |                |
         |--------------->|                |                |
```

Figure 4: Prototypical Call Flow

5.1. Amplification Avoidance

Once 'sip:B@example.com' is in the pending state, the relay needs to ask user B for permission by sending a MESSAGE request to 'sip:B@example.com'. However, the relay needs to ensure that it is not used as an amplifier to launch amplification attacks.

In such an attack, the attacker would add a large number of recipient URIs to the translation logic of a relay. The relay would then send a MESSAGE request to each of those recipient URIs. The bandwidth generated by the relay would be much higher than the bandwidth used by the attacker to add those recipient URIs to the translation logic of the relay.

This framework uses a credit-based authorization mechanism to avoid the attack just described. It requires users adding new recipient URIs to a translation to generate an amount of bandwidth that is comparable to the bandwidth the relay will generate when sending MESSAGE requests towards those recipient URIs. When XCAP is used, this requirement is met by not allowing clients to add more than one URI per HTTP transaction. When a REGISTER transaction is used, this requirement is met by not allowing clients to register more than one contact per REGISTER transaction.

5.1.1. Relay's Behavior

Relays implementing this framework MUST NOT allow clients to add more than one recipient URI per transaction. If a client using XCAP attempts to add more than one recipient URI in a single HTTP transaction, the XCAP server SHOULD return an HTTP 409 (Conflict) response. The XCAP server SHOULD describe the reason for the refusal in an XML body using the <constraint-failure> element, as described in [RFC4825]. If a client attempts to register more than one contact in a single REGISTER transaction, the registrar SHOULD return a SIP 403 response and explain the reason for the refusal in its reason phrase (e.g., maximum one contact per registration).

5.2. Subscription to the Permission Status

Clients need a way to be informed about the status of the operations they requested. Otherwise, users can be waiting for an operation to succeed when it has actually already failed. In particular, if the target of the request for consent was not reachable and did not have an associated store-and-forward server, the client needs to know to retry the request later. The Pending Additions SIP event package [RFC5362] is a way to provide clients with that information.

Clients can use the Pending Additions SIP event package to be informed about the status of the operations they requested. That is, the client will be informed when an operation (e.g., the addition of a recipient URI to a relay's translation logic) is authorized (and thus executed) or rejected. Clients use the target URI of the SIP translation being manipulated to subscribe to the 'pending-additions' event package.

In our example, after receiving the response from the relay (2), user A subscribes to the Pending Additions event package at the relay (5). This subscription keeps user A informed about the status of the permissions (e.g., granted or denied) the relay will obtain.

5.2.1. Relay's Behavior

Relays SHOULD support the Pending Additions SIP event package specified in [RFC5362].

5.3. Request for Permission

A relay requests permissions from potential recipients to add them to its translation logic using MESSAGE requests. In our example, on receiving the request to add user B to the translation logic of the relay (1), the relay generates a MESSAGE request (3) towards 'sip:B@example.com'. This MESSAGE request carries a permission document, which describes the translation that needs to be authorized and carries a set of URIs to be used by the recipient to grant or to deny the relay permission to perform that translation. Since user B is offline, the MESSAGE request will be buffered by user B's store-and-forward server. User B will later go online and authorize the translation by using one of those URIs, as described in Section 5.6. The MESSAGE request also carries a body part that contains the same information as the permission document but in a human-readable format.

When user B uses one of the URIs in the permission document to grant or deny permissions, the relay needs to make sure that it was actually user B using that URI, and not an attacker. The relay can use any of the methods described in Section 5.6 to authenticate the permission document.

5.3.1. Relay's Behavior

Relays that implement this framework MUST obtain permissions from potential recipients before adding them to their translation logic. Relays request permissions from potential recipients using MESSAGE requests.

Section 5.6 describes the methods a relay can use to authenticate those recipients giving the relay permission to perform a particular translation. These methods are SIP identity [RFC4474], P-Asserted-Identity [RFC3325], a return routability test, or SIP digest. Relays that use the method consisting of a return routability test have to send their MESSAGE requests to a SIPS URI, as specified in Section 5.6.

MESSAGE requests sent to request permissions MUST include a permission document and SHOULD include a human-readable part in their bodies. The human-readable part contains the same information as the permission document (but in a human-readable format), including the URIs to grant and deny permissions. User agents that do not understand permission documents can still process the request and display it in a sensible way to the user, as they would display any other instant message. This way, even if the user agent does not implement this framework, the (human) user will be able to manually click on the correct URI in order to grant or deny permissions. The following is an example of a MESSAGE request that carries a human-readable part and a permission document, which follows the format specified in [RFC5361], in its body. Not all header fields are shown for simplicity reasons.

```
MESSAGE sip:bob@example.org SIP/2.0
From: <sip:alices-friends@example.com>;tag=12345678
To: <sip:bob@example.org>
Content-Type: multipart/mixed;boundary="boundary1"

--boundary1
Content-Type: text/plain

If you consent to receive traffic sent to
<sip:alices-friends@example.com>, please use one of the following
URIs: <sips:grant-1awdch5Fasddfce34@example.com> or
<https://example.com/grant-1awdch5Fasddfce34>.  Otherwise, use one of
the following URIs: <sips:deny-23rCsdfgvdT5sdfgye@example.com> or
<https://example.com/deny-23rCsdfgvdT5sdfgye>.
--boundary1
Content-Type: application/auth-policy+xml

<?xml version="1.0" encoding="UTF-8"?>
     <cp:ruleset
         xmlns="urn:ietf:params:xml:ns:consent-rules"
         xmlns:cp="urn:ietf:params:xml:ns:common-policy"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
         <cp:rule id="f1">
      <cp:conditions>
          <cp:identity>
              <cp:many/>
          </cp:identity>
          <recipient>
              <cp:one id="sip:bob@example.org"/>
          </recipient>
          <target>
              <cp:one id="sip:alices-friends@example.com"/>
          </target>
      </cp:conditions>
      <cp:actions>
          <trans-handling
              perm-uri="sips:grant-1awdch5Fasddfce34@example.com">
              grant</trans-handling>
          <trans-handling
              perm-uri="https://example.com/grant-1awdch5Fasddfce34">
              grant</trans-handling>
          <trans-handling
              perm-uri="sips:deny-23rCsdfgvdT5sdfgye@example.com">
              deny</trans-handling>
          <trans-handling
              perm-uri="https://example.com/deny-23rCsdfgvdT5sdfgye">
              deny</trans-handling>
      </cp:actions>
      <cp:transformations/>
  </cp:rule>
  </cp:ruleset>
--boundary1--
```

5.4. Permission Document Structure

A permission document is the representation (e.g., encoded in XML) of a permission. A permission document contains several pieces of data:

Identity of the Sender: A URI representing the identity of the sender for whom permissions are granted.

Identity of the Original Recipient: A URI representing the identity of the original recipient, which is used as the input for the translation operation. This is also called the target URI.

Identity of the Final Recipient: A URI representing the result of the translation. The permission grants ability for the sender to send requests to the target URI and for a relay receiving those requests to forward them to this URI. This is also called the recipient URI.

URIs to Grant Permission: URIs that recipients can use to grant the relay permission to perform the translation described in the document. Relays MUST support the use of SIP and SIPS URIs in permission documents and MAY support the use of HTTP and HTTPS URIs.

URIs to Deny Permission: URIs that recipients can use to deny the relay permission to perform the translation described in the document. Relays MUST support the use of SIP and SIPS URIs in permission documents and MAY support the use of HTTP and HTTPS URIs.

Permission documents can contain wildcards. For example, a permission document can request permission for any relay to forward requests coming from a particular sender to a particular recipient. Such a permission document would apply to any target URI. That is, the field containing the identity of the original recipient would match any URI. However, the recipient URI MUST NOT be wildcarded.

Entities implementing this framework MUST support the format for permission documents defined in [RFC5361] and MAY support other formats.

In our example, the permission document in the MESSAGE request (3) sent by the relay contains the following values:

Identity of the Sender: Any sender

Identity of the Original Recipient: sip:friends@example.com

Identity of the Final Recipient: sip:B@example.com

URI to Grant Permission: sips:grant-1awdch5Fasddfce34@example.com

URI to Grant Permission: https://example.com/grant-1awdch5Fasddfce34

URI to Deny Permission: sips:deny-23rCsdfgvdT5sdfgye@example.com

URI to Deny Permission: https://example.com/deny-23rCsdfgvdT5sdfgye

It is expected that the Sender field often contains a wildcard. However, scenarios involving request-contained URI lists, such as the one described in Section 5.9, can require permission documents that apply to a specific sender. In cases where the identity of the sender matters, relays MUST authenticate senders.

5.5. Permission Requested Notification

On receiving the MESSAGE request (3), user B's store-and-forward server stores it because user B is offline at that point. When user B goes online, user B fetches all the requests its store-and-forward server has stored (9).

5.6. Permission Grant

A recipient gives a relay permission to execute the translation described in a permission document by sending a SIP PUBLISH or an HTTP GET request to one of the URIs to grant permissions contained in the document. Similarly, a recipient denies a relay permission to execute the translation described in a permission document by sending a SIP PUBLISH or an HTTP GET request to one of the URIs to deny permissions contained in the document. Requests to grant or deny permissions contain an empty body.

In our example, user B obtains the permission document (10) that was received earlier by its store-and-forward server in the MESSAGE request (3). User B authorizes the translation described in the permission document received by sending a PUBLISH request (11) to the SIP URI to grant permissions contained in the permission document.

5.6.1. Relay's Behavior

Relays MUST ensure that the SIP PUBLISH or the HTTP GET request received was generated by the recipient of the translation and not by an attacker. Relays can use four methods to authenticate those requests: SIP identity, P-Asserted-Identity [RFC3325], a return routability test, or SIP digest. While return routability tests can be used to authenticate both SIP PUBLISH and HTTP GET requests, SIP identity, P-Asserted-Identity, and SIP digest can only be used to authenticate SIP PUBLISH requests. SIP digest can only be used to authenticate recipients that share a secret with the relay (e.g., recipients that are in the same domain as the relay).

5.6.1.1. SIP Identity

The SIP identity [RFC4474] mechanism can be used to authenticate the sender of a PUBLISH request. The relay MUST check that the originator of the PUBLISH request is the owner of the recipient URI in the permission document. Otherwise, the PUBLISH request SHOULD be responded with a 401 (Unauthorized) response and MUST NOT be processed further.

5.6.1.2. P-Asserted-Identity

The P-Asserted-Identity [RFC3325] mechanism can also be used to authenticate the sender of a PUBLISH request. However, as discussed in [RFC3325], this mechanism is intended to be used only within networks of trusted SIP servers. That is, the use of this mechanism is only applicable inside an administrative domain with previously agreed-upon policies.

The relay MUST check that the originator of the PUBLISH request is the owner of the recipient URI in the permission document. Otherwise, the PUBLISH request SHOULD be responded with a 401 (Unauthorized) response and MUST NOT be processed further.

5.6.1.3. Return Routability

SIP identity provides a good authentication mechanism for incoming PUBLISH requests. Nevertheless, SIP identity is not widely available on the public Internet yet. That is why an authentication mechanism that can already be used at this point is needed.

Return routability tests do not provide the same level of security as SIP identity, but they provide a better-than-nothing security level in architectures where the SIP identity mechanism is not available (e.g., the current Internet). The relay generates an unguessable URI (i.e., with a cryptographically random user part) and places it in the permission document in the MESSAGE request (3). The recipient needs to send a SIP PUBLISH request or an HTTP GET request to that URI. Any incoming request sent to that URI SHOULD be considered authenticated by the relay.

> Note that the return routability method is the only one that allows the use of HTTP URIs in permission documents. The other methods require the use of SIP URIs.

Relays using a return routability test to perform this authentication MUST send the MESSAGE request with the permission document to a SIPS URI. This ensures that attackers do not get access to the (unguessable) URI. Thus, the only user able to use the (unguessable) URI is the receiver of the MESSAGE request. Similarly, permission documents sent by relays using a return routability test MUST only contain secure URIs (i.e., SIPS and HTTPS) to grant and deny permissions. A part of these URIs (e.g., the user part of a SIPS URI) MUST be cryptographically random with at least 32 bits of randomness.

Relays can transition from return routability tests to SIP identity by simply requiring the use of SIP identity for incoming PUBLISH requests. That is, such a relay would reject PUBLISH requests that did not use SIP identity.

5.6.1.4. SIP Digest

The SIP digest mechanism can be used to authenticate the sender of a PUBLISH request as long as that sender shares a secret with the relay. The relay MUST check that the originator of the PUBLISH request is the owner of the recipient URI in the permission document. Otherwise, the PUBLISH request SHOULD be responded with a 401 (Unauthorized) response and MUST NOT be processed further.

5.7. Permission Granted Notification

On receiving the PUBLISH request (11), the relay sends a NOTIFY request (13) to inform user A that the permission for the translation has been received and that the translation logic at the relay has been updated. That is, 'sip:B@example.com' has been added as a recipient URI.

5.8. Permission Revocation

At any time, if a recipient wants to revoke any permission, it uses the URI it received in the permission document to deny the permissions it previously granted. If a recipient loses this URI for some reason, it needs to wait until it receives a new request produced by the translation. Such a request will contain a Trigger-Consent header field with a URI. That Trigger-Consent header field will have a target-uri header field parameter identifying the target URI of the translation. The recipient needs to send a PUBLISH request with an empty body to the URI in the Trigger-Consent header field in order to receive a MESSAGE request from the relay. Such a MESSAGE request will contain a permission document with a URI to revoke the permission that was previously granted.

Figure 5 shows an example of how a user that lost the URI to revoke permissions at a relay can obtain a new URI using the Trigger-Consent header field of an incoming request. The user rejects an incoming INVITE (1) request, which contains a Trigger-Consent header field. Using the URI in that header field, the user sends a PUBLISH request (4) to the relay. On receiving the PUBLISH request (4), the relay generates a MESSAGE request (6) towards the user. Finally, the user revokes the permissions by sending a PUBLISH request (8) to the relay.

```
           Relay                     B@example.com
             |(1) INVITE                    |
             |    Trigger-Consent: sip:123@relay.example.com
             |     ;target-uri="sip:friends@relay.example.com"
             |----------------------------->|
             |(2) 603 Decline               |
             |<-----------------------------|
             |(3) ACK                       |
             |----------------------------->|
             |(4) PUBLISH sip:123@relay.example.com
             |<-----------------------------|
             |(5) 200 OK                    |
             |----------------------------->|
             |(6) MESSAGE sip:B@example     |
             |    Permission Document       |
             |----------------------------->|
             |(7) 200 OK                    |
             |<-----------------------------|
             |(8) PUBLISH uri-deny          |
             |<-----------------------------|
             |(9) 200 OK                    |
             |----------------------------->|
```

Figure 5: Permission Revocation

5.9. Request-Contained URI Lists

In the scenarios described so far, a user adds recipient URIs to the translation logic of a relay. However, the relay does not perform translations towards those recipient URIs until permissions are obtained.

URI-list services using request-contained URI lists are a special case because the selection of recipient URIs is performed at the same time as the communication attempt. A user places a set of recipient URIs in a request and sends it to a relay so that the relay sends a similar request to all those recipient URIs.

Relays implementing this consent framework and providing request-contained URI-list services behave in a slightly different way than the relays described so far. This type of relay also maintains a list of recipient URIs for which permissions have been received. Clients also manipulate this list using a manipulation mechanism (e.g., XCAP). Nevertheless, this list does not represent the recipient URIs of every translation performed by the relay. This list just represents all the recipient URIs for which permissions have been received -- that is, the set of URIs that will be accepted if a request containing a URI-list arrives to the relay. This set of URIs is a superset of the recipient URIs of any particular translation the relay performs.

5.9.1. Relay's Behavior

On receiving a request-contained URI list, the relay checks whether or not it has permissions for all the URIs contained in the incoming URI list. If it does, the relay performs the translation. If it lacks permissions for one or more URIs, the relay MUST NOT perform the translation and SHOULD return an error response.

A relay that receives a request-contained URI list with a URI for which the relay has no permissions SHOULD return a 470 (Consent Needed) response. The relay SHOULD add a Permission-Missing header field with the URIs for which the relay has no permissions.

Figure 6 shows a relay that receives a request (1) that contains URIs for which the relay does not have permission (the INVITE carries the recipient URIs in its message body). The relay rejects the request with a 470 (Consent Needed) response (2). That response contains a Permission-Missing header field with the URIs for which there was no permission.

```
    A@example.com               Relay

          |(1) INVITE             |
          |    sip:B@example.com  |
          |    sip:C@example.com  |
          |---------------------->|
          |(2) 470 Consent Needed |
          |    Permission-Missing: sip:C@example.com
          |<----------------------|
          |(3) ACK                |
          |---------------------->|
```

Figure 6: INVITE with a URI List in Its Body

5.9.2. Definition of the 470 Response Code

A 470 (Consent Needed) response indicates that the request that triggered the response contained a URI list with at least one URI for which the relay had no permissions. A user agent server generating a 470 (Consent Needed) response SHOULD include a Permission-Missing header field in it. This header field carries the URI or URIs for which the relay had no permissions.

A user agent client receiving a 470 (Consent Needed) response without a Permission-Missing header field needs to use an alternative mechanism (e.g., XCAP) to discover for which URI or URIs there were no permissions.

A client receiving a 470 (Consent Needed) response uses a manipulation mechanism (e.g., XCAP) to add those URIs to the relay's list of URIs. The relay will obtain permissions for those URIs as usual.

5.9.3. Definition of the Permission-Missing Header Field

Permission-Missing header fields carry URIs for which a relay did not have permissions. The following is the augmented Backus-Naur Form (BNF) [RFC5234] syntax of the Permission-Missing header field. Some of its elements are defined in [RFC3261].

```
   Permission-Missing  =  "Permission-Missing" HCOLON per-miss-spec
                          *( COMMA per-miss-spec )
   per-miss-spec       =  ( name-addr / addr-spec )
                         *( SEMI generic-param )
```

The following is an example of a Permission-Missing header field:

```
   Permission-Missing: sip:C@example.com
```

5.10. Registrations

Even though the example used to specify this framework has been a URI-list service, this framework applies to any type of translation (i.e., not only to URI-list services). Registrations are a different type of translations that deserve discussion.

Registrations are a special type of translations. The user registering has a trust relationship with the registrar in its home domain. This is not the case when a user gives any type of permissions to a relay in a different domain.

Traditionally, REGISTER transactions have performed two operations at the same time: setting up a translation and authorizing the use of that translation. For example, a user registering its current contact URI is giving permission to the registrar to forward traffic sent to the user's AoR (Address of Record) to the registered contact URI. This works fine when the entity registering is the same as the one that will be receiving traffic at a later point (e.g., the entity receives traffic over the same connection used for the registration as described in [OUTBOUND]). However, this schema creates some potential attacks that relate to third-party registrations.

An attacker binds, via a registration, his or her AoR with the contact URI of a victim. Now the victim will receive unsolicited traffic that was originally addressed to the attacker.

The process of authorizing a registration is shown in Figure 7. User A performs a third-party registration (1) and receives a 202 (Accepted) response (2).

Since the relay does not have permission from 'sip:a@ws123.example.com' to perform translations towards that recipient URI, the relay places 'sip:a@ws123.example.com' in the 'pending' state. Once 'sip:a@ws123.example.com' is in the 'Permission Pending' state, the registrar needs to ask 'sip:a@ws123.example.com' for permission by sending a MESSAGE request (3).

After receiving the response from the relay (2), user A subscribes to the Pending Additions event package at the registrar (5). This subscription keeps the user informed about the status of the permissions (e.g., granted or denied) the registrar will obtain. The rest of the process is similar to the one described in Section 5.

```
   A@example.com         Registrar     a@ws123.example.com

         |(1) REGISTER        |                    |
         |    Contact: sip:a@ws123.example.com     |
         |------------------->|                    |
         |(2) 202 Accepted OK |                    |
         |<-------------------|                    |
         |                    |(3) MESSAGE sip:a@ws123.example
         |                    |    Permission Document
         |                    |------------------->|
         |                    |(4) 200 OK          |
         |                    |<-------------------|
         |(5) SUBSCRIBE       |                    |
         |    Event: pending-additions             |
         |------------------->|                    |
         |(6) 200 OK          |                    |
         |<-------------------|                    |
         |(7) NOTIFY          |                    |
         |<-------------------|                    |
         |(8) 200 OK          |                    |
         |------------------->|                    |
         |                    |(9) PUBLISH uri-up  |
         |                    |<-------------------|
         |                    |(10) 200 OK         |
         |                    |------------------->|
         |(11) NOTIFY         |                    |
         |<-------------------|                    |
         |(12) 200 OK         |                    |
         |------------------->|                    |
```

Figure 7: Registration

Permission documents generated by registrars are typically very general. For example, in one such document a registrar can ask a recipient for permission to forward any request from any sender to the recipient's URI. This is the type of granularity that this framework intends to provide for registrations. Users who want to define how incoming requests are treated with a finer granularity (e.g., requests from user A are only accepted between 9:00 and 11:00) will have to use other mechanisms such as Call Processing Language (CPL) [RFC3880].

Note that, as indicated previously, user agents using the same connection to register and to receive traffic from the registrar, as described in [OUTBOUND], do not need to use the mechanism described in this section.

A user agent being registered by a third party can be unable to use the SIP Identity, P-Asserted-Identity, or SIP digest mechanisms to prove to the registrar that the user agent is the owner of the URI being registered (e.g., sip:user@192.0.2.1), which is the recipient URI of the translation. In this case, return routability MUST be used.

5.11. Relays Generating Traffic towards Recipients

Relays generating traffic towards recipients need to make sure that those recipients can revoke the permissions they gave at any time. The Trigger-Consent helps achieve this.

5.11.1. Relay's Behavior

A relay executing a translation that involves sending a request to a URI from which permissions were obtained previously SHOULD add a Trigger-Consent header field to the request. The URI in the Trigger-Consent header field MUST have a target-uri header field parameter identifying the target URI of the translation.

On receiving a PUBLISH request addressed to the URI that a relay previously placed in a Trigger-Consent header field, the relay SHOULD send a MESSAGE request to the corresponding recipient URI with a permission document. Therefore, the relay needs to be able to correlate the URI it places in the Trigger-Consent header field with the recipient URI of the translation.

5.11.2. Definition of the Trigger-Consent Header Field

The following is the augmented Backus-Naur Form (BNF) [RFC5234] syntax of the Trigger-Consent header field. Some of its elements are defined in [RFC3261].

```
   Trigger-Consent     =  "Trigger-Consent" HCOLON trigger-cons-spec
                          *( COMMA trigger-cons-spec )
   trigger-cons-spec   =  ( SIP-URI / SIPS-URI )
                          *( SEMI trigger-param )
   trigger-param       =  target-uri / generic-param
   target-uri          =  "target-uri" EQUAL
                              LDQUOT *( qdtext / quoted-pair ) RDQUOT
```

The target-uri header field parameter MUST contain a URI.

The following is an example of a Trigger-Consent header field:

```
   Trigger-Consent: sip:123@relay.example.com
                    ;target-uri="sip:friends@relay.example.com"
```

6. IANA Considerations

Per the following sections, IANA has registered a SIP response code, two SIP header fields, and a SIP header field parameter.

6.1. Registration of the 470 Response Code

IANA has added the following new response code to the Methods and Response Codes subregistry under the SIP Parameters registry.

Response Code Number:	470
Default Reason Phrase:	Consent Needed
Reference:	[RFC5360]

6.2. Registration of the Trigger-Consent Header Field

IANA has added the following new SIP header field to the Header Fields subregistry under the SIP Parameters registry.

Header Name:	Trigger-Consent
Compact Form:	(none)
Reference:	[RFC5360]

6.3. Registration of the Permission-Missing Header Field

IANA has added the following new SIP header field to the Header Fields subregistry under the SIP Parameters registry.

Header Name:	Permission-Missing
Compact Form:	(none)
Reference:	[RFC5360]

6.4. Registration of the target-uri Header Field Parameter

IANA has registered the 'target-uri' Trigger-Consent header field parameter under the Header Field Parameters and Parameter Values subregistry within the SIP Parameters registry:

Header Field	Parameter Name	Predefined Values	Reference
Trigger-Consent	target-uri	No	[RFC5360]

7. Security Considerations

Security has been discussed throughout the whole document. However, there are some issues that deserve special attention.

Relays generally implement several security mechanisms that relate to client authentication and authorization. Clients are typically authenticated before they can manipulate a relay's translation logic. Additionally, clients are typically also authenticated and sometimes need to perform SPAM prevention tasks [RFC5039] when they send traffic to a relay. It is important that relays implement these types of security mechanisms. However, they fall out of the scope of this framework. Even with these mechanisms in place, there is still a need for relays to implement this framework because the use of these mechanisms does not prevent authorized clients to add recipients to a translation without their consent. Consequently, relays performing translations MUST implement this framework.

> Note that, as indicated previously, user agents using the same connection to register and to receive traffic from the registrar, as described in [OUTBOUND], do not need to use this framework. Therefore, a registrar that did not accept third-party registrations would not need to implement this framework.

As pointed out in Section 5.6.1.3, when return routability tests are used to authenticate recipients granting or denying permissions, the URIs used to grant or deny permissions need to be protected from attackers. SIPS URIs provide a good tool to meet this requirement, as described in [RFC5361]. When store-and-forward servers are used, the interface between a user agent and its store-and-forward server is frequently not based on SIP. In such a case, SIPS cannot be used to secure those URIs. Implementations of store-and-forward servers MUST provide a mechanism for delivering encrypted and integrity-protected messages to their user agents.

The information provided by the Pending Additions event package can be sensitive. For this reason, as described in [RFC5362], relays need to use strong means for authentication and information confidentiality. SIPS URIs are a good mechanism to meet this requirement.

Permission documents can reveal sensitive information. Attackers may attempt to modify them in order to have clients grant or deny permissions different from the ones they think they are granting or denying. For this reason, it is RECOMMENDED that relays use strong means for information integrity protection and confidentiality when sending permission documents to clients.

The mechanism used for conveying information to clients SHOULD ensure the integrity and confidentially of the information. In order to achieve these, an end-to-end SIP encryption mechanism, such as S/MIME, as described in [RFC3261], SHOULD be used.

If strong end-to-end security means (such as above) are not available, it is RECOMMENDED that hop-by-hop security based on TLS and SIPS URIs, as described in [RFC3261], is used.

8. Acknowledgments

Henning Schulzrinne, Jon Peterson, and Cullen Jennings provided useful ideas on this document. Ben Campbell, AC Mahendran, Keith Drage, and Mary Barnes performed a thorough review of this document.

9. References

9.1. Normative References

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3428] Campbell, B., Ed., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5361] Camarillo, G., "A Document Format for Requesting Consent", RFC 5361, October 2008.

[RFC5362] Camarillo, G., "The Session Initiation Protocol (SIP) Pending Additions Event Package", RFC 5362, October 2008.

[RFC5363] Camarillo, G. and A.B. Roach, "Framework and Security Considerations for Session Initiation Protocol (SIP) URI-List Services", RFC 5363, October 2008.

9.2. Informative References

[RFC3325] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[RFC3880] Lennox, J., Wu, X., and H. Schulzrinne, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services", RFC 3880, October 2004.

[RFC4453] Rosenberg, J., Camarillo, G., Ed., and D. Willis, "Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP)", RFC 4453, April 2006.

[RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

[RFC4825] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", RFC 4825, May 2007.

[RFC4826] Rosenberg, J., "Extensible Markup Language (XML) Formats for Representing Resource Lists", RFC 4826, May 2007.

[RFC5039] Rosenberg, J. and C. Jennings, "The Session Initiation Protocol (SIP) and Spam", RFC 5039, January 2008.

[OUTBOUND] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", Work in Progress, June 2007.

Authors' Addresses

Jonathan Rosenberg
Cisco
Iselin, NJ 08830
USA

EMail: jdrosen@cisco.com
URI: http://www.jdrosen.net

Gonzalo Camarillo (editor)
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Dean Willis
Unaffiliated
3100 Independence Pkwy #311-164
Plano, TX 75075
USA

EMail: dean.willis@softarmor.com