SIPPING
Internet-Draft
Expires: December 14, 2006

J. Rosenberg
Cisco Systems
G. Camarillo, Ed.
Ericsson
D. Willis
Cisco Systems
June 12, 2006

A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)
draft-ietf-sipping-consent-framework-05.txt

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) supports communications across many media types, 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 3
2. Definitions and Terminology 3
3. Relays and Translations 4
4. Architecture 5
4.1. Permissions at a Relay 6
4.2. Consenting Manipulations on a Relay's Transaction Logic . 6
4.3. Permission Servers 7
4.4. Recipients Grant Permissions 8
5. Framework Operations 8
5.1. Amplification Avoidance 9
5.2. Subscription to the Permission Status 10
5.3. Request for Permission 10
5.4. Permission Document Structure 11
5.5. Permission Requested Notification 12
5.6. Permission Grant 13
5.6.1. SIP Identity 13
5.6.2. P-Asserted-Identity 13
5.6.3. Return Routability 14
5.7. Permission Granted Notification 14
5.8. Permission Revocation 15
5.9. Request-contained URI Lists 16
5.10. Registrations 17
5.11. Relays Generating Traffic towards Recipients 20
6. IANA Considerations 20
7. Security Considerations 21
8. Acknowledges 22
9. References 22
9.1. Normative References 22
9.2. Informative References 23
Authors' Addresses 24
Intellectual Property and Copyright Statements 25

1. Introduction

The Session Initiation Protocol (SIP) [3] supports communications across many media types, 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 [5]) 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 SIP network 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 in SIP networks. These include amplification and DoS (Denial of Service) attacks. These problems are described in more detail in a companion requirements document [13].

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

2. Definitions and Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant implementations.

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 may have been the result of a translation operation.

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 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) which 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, which 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.

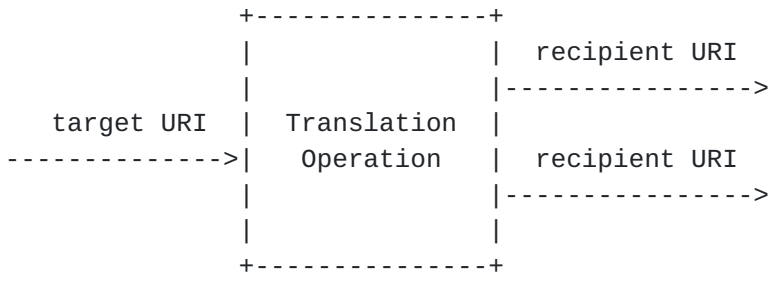

Figure 1: Translation operation

Thus, an essential aspect of a relay is that of translation. When a relay receives a request, it translates, following its translation logic, the Request-URI into one or more additional URIs. That is, the relay can create outgoing requests to one or more additional URIs. The translation operation is what creates the consent problem.

Additionally, since the translation operation can result in more than one URI, it is also the source of amplification. Servers that do not perform translations, such as outbound proxy servers, do not cause amplification.

Since the translation operation is based on local policy or local data (such as registrations), it is 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, '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 '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, which must 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.

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' (resource@example.com). This request may 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 may or may not belong to the domain 'example.com'. This relay may be, for instance, a proxy server or a URI-list service [14].

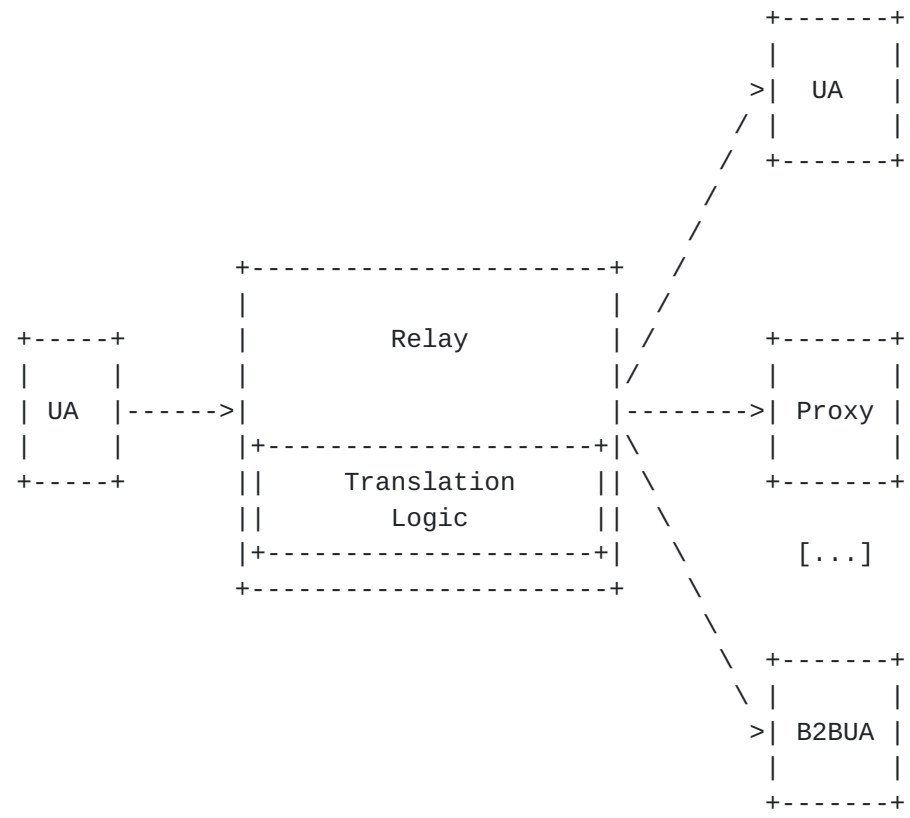

Figure 2: Relay performing a translation

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. 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 introduces permission 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.

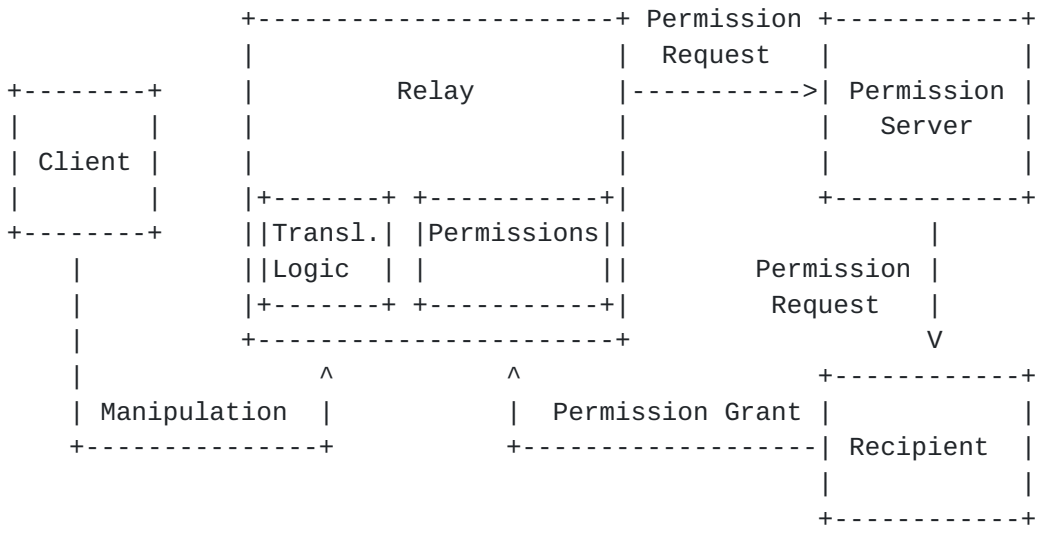

Figure 3: Reference Architecture

4.1. Permissions at a Relay

Relays implementing this framework obtain and store permissions associated to their translation logics. These permissions indicate if a particular recipient has agreed to receive traffic or not 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.

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.

4.2. Consenting Manipulations on a Relay's Transaction 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 [5] requests.

For example, the relay hosting the URI-list service at

'friends@example.com' performs a translation from that 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 'bob@example.org' asking whether or not it is OK to perform the translation from 'friends@example.com' to '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.

Note that 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 XCAP [11] and REGISTER transactions.

In any case, relays implementing this framework SHOULD have a means to indicate that a particular recipient URI is in the states specified in [10] (i.e., pending, waiting, error, denied, or granted).

4.3. Permission 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. Nevertheless, users are not on-line all the time and, so, sometimes are not able to receive MESSAGE requests.

This issue is also found in presence, where a user's status is reported by a presence server instead of by the user's user agents, which can go on and off line. Similarly, we define permission servers, which are a key element of this framework. Permission servers are network elements that act as SIP user agents and handle MESSAGE requests for a user.

So, a permission server stores incoming MESSAGE requests when the user is unavailable and delivers them when the user is available again. Conceptually, a permission server provides a store-and-forward message service.

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 permission server as long as they agree on which mechanism to use. Therefore, this framework does not make any recommendation on the interface between user agents and their permission servers.

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

4.4. Recipients Grant Permissions

Relays include in the permission documents they generate 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 may include HTTP [2] URIs for this purpose. Consequently, recipients provide relays with permissions using SIP PUBLISH requests or HTTP GET requests.

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 permission servers) play an essential role in this call flow.

Figure 4 shows the complete process to add a recipient URI ('B@example.com') to the translation logic of a relay. User A attempts to add 'B@example.com' as a new recipient URI to the translation logic of the relay (1). User A uses XCAP [11] and the XML (Extensible Markup Language) format for representing resource lists [12] to perform this addition. Since the relay does not have permission from 'B@example.com' to perform translations towards that URI, the relay places 'B@example.com' in the pending state, as specified in [10].

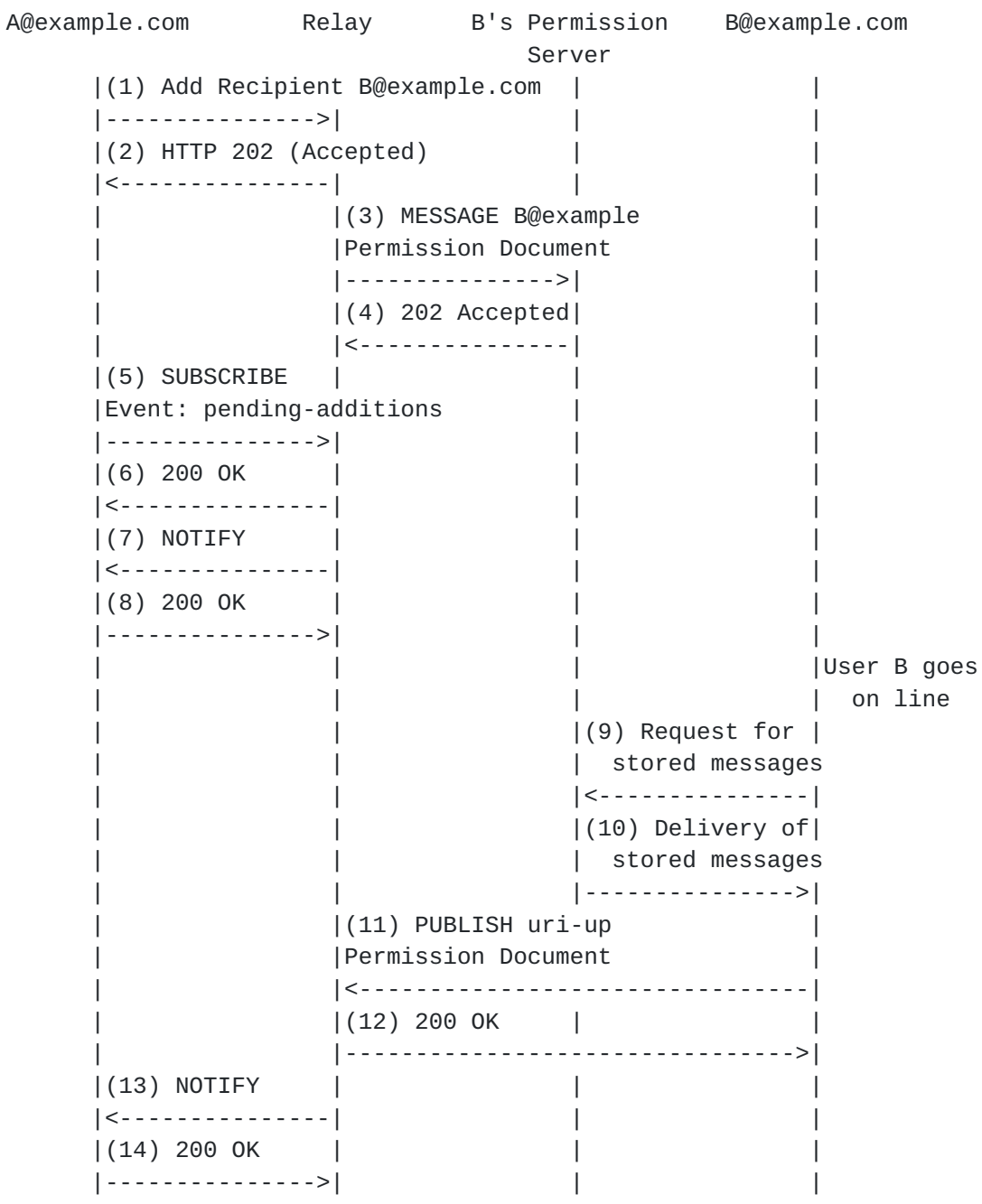

Figure 4: Prototypical call flow

5.1. Amplification Avoidance

Once 'B@example.com' is in the pending state, the relay needs to ask user B for permission by sending a MESSAGE request to '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 URIs. The bandwidth generated by the relay would be much higher than the bandwidth used by the attacker to add those 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.

Therefore, relays implementing this framework MUST NOT allow clients to add more than one URI per HTTP transaction. If a client attempts to add more than one URI in a single HTTP transaction, the XCAP server SHOULD return an HTTP 403 (Forbidden) response. The XCAP server SHOULD describe the reason for the refusal (i.e., only one URI can be added at a time) in the entity, as described in [2].

5.2. Subscription to the Permission Status

Clients can use the Pending Additions SIP event package [10] 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 URI to the translation logic of a relay) is authorized (and thus executed) or rejected.

OPEN ISSUE: how do clients obtain the URI to subscribe to the Pending Additions event package, both when using XCAP and when using REGISTERs to manipulate the translation?

In our example, after receiving the response from the server (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.

5.3. Request for Permission

Relays MUST obtain permissions from potential recipients before adding them to their translation logics. Relays request permissions from potential recipients using MESSAGE requests.

MESSAGE requests sent to request permissions MUST include a permission document and SHOULD include a human-readable part in their bodies. MESSAGE requests also carry a body part that contains the same information as the permission document but in a human-readable format 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.

Section 5.6 describes three methods a relay can use to authenticate recipients giving the relay permission to perform a particular translation. 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.

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 '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. User B will authorize the translation by using one of those URIs, as described in Section 5.6. The MESSAGE request also carry 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 the one using that URI, and not an attacker. The relay can use three methods to authenticate the permission document: SIP identity [7], P-Asserted-Identity [4], or a return routability test. These methods are described in Section 5.6.

5.4. Permission Document Structure

A permission document is the XML representation 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. At least one of these URIs MUST be a SIP or SIPS URI. HTTP and HTTPS URIs MAY also be used.

URIs to Deny Permission: URIs that recipients can use to deny the relay permission to perform the translation described in the document. At least one of these URIs MUST be a SIP or SIPS URI. HTTP and HTTPS URIs MAY also be used.

Permission documents may contain wildcards. For example, a permission document may 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.

Entities implementing this framework MUST support the format for permission documents defined in [9].

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

Identity of the Sender: Any.

Identity of the Original Recipient: friends@example.com

Identity of the Final Recipient: 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, may require permission documents that apply to a specific sender. Of course, 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 permission server stores it because User B is off line at that point. When User B goes

on line, User B fetches all the requests its permission server has stored (9).

5.6. Permission Grant

A client 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 client 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.

In our example, User B obtains the permission document (10) that was received earlier by its permission 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.

Relays need to 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 three methods to authenticate those requests: SIP identity, P-Asserted-Identity [4], or a return routability test. While return routability tests can be used to authenticate both SIP PUBLISH and HTTP GET requests, SIP identity and P-Asserted-Identity can only be used to authenticate SIP PUBLISH requests.

5.6.1. SIP Identity

The SIP identity [7] 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.

5.6.2. P-Asserted-Identity

The P-Asserted-Identity [4] mechanism can also be used to authenticate the sender of a PUBLISH request. However, as discussed in RFC 3325 [4], this mechanism should only be used 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.

5.6.3. 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 good-enough 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 long and random-looking 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. The user part of these URIs 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.

5.7. 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, 'B@example.com' has been added as a recipient URI.

5.8. Permission Revocation

At any time, if a client wants to revoke any permission, it uses the URI it received in the permission document to deny the permissions it previously granted. If a client 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 URI will have an escaped Refer-To header field identifying the client (i.e., the recipient of the translation). The client needs to send a REFER request 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 the that header field, the user sends a REFER request (4) to the relay. On receiving the REFER 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.

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 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 framework and providing this type of URI-list services MUST maintain a list of recipient URIs from which permission have been received. This list is manipulated in the same way as described in Section 5 and represents the set of URIs that will be accepted if a request containing a URI-list arrives to the relay.

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.

The following is the augmented Backus-Naur Form (BNF) [6] syntax of the Permission-Missing header field. Some of its elements are defined in RFC 3261 [3].

```
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>
```

Figure 6 shows a relay that receives a request (1) that contains URIs for which the relay does not have permission. 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.

Figure 6: INVITE with a URI list in its body

5.10. Registrations

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 Records) 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 [8]). However, this schema creates some potential attacks which 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 'a@ws123.example.com' to perform translations towards that URI, the relay places 'a@ws123.example.com' in the 'pending' state. Once 'a@ws123.example.com' is in the 'Permission Pending' state, the registrar needs to ask 'a@ws123.example.com' for permission by sending a MESSAGE request (3).

After receiving the response from the server (2), user A subscribes to the Pending Additions event package at the registrar (4). 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.

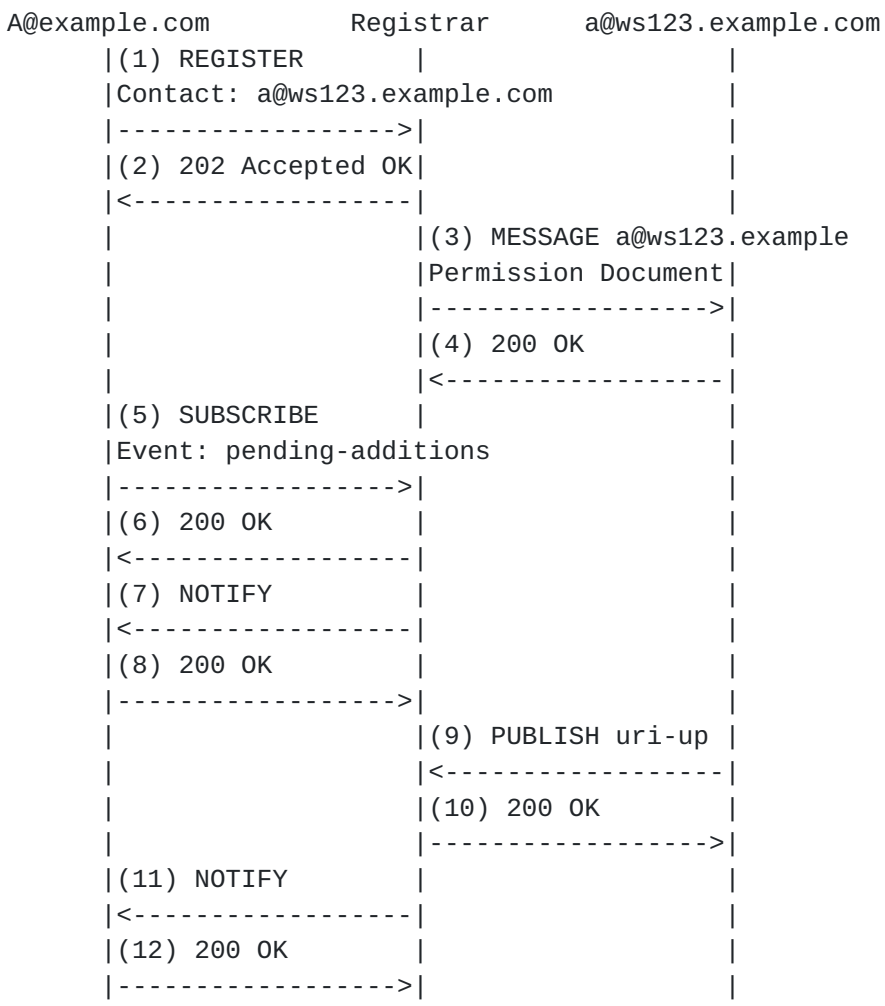

Figure 7: Registration

Permission documents generated by registrars are typically very general. For example, in one such document a registrar may 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 should only be accepted between 9:00 and 11:00) should use other mechanisms such as CPL [15].

Note that, as indicated previously, user agents using the same connection to register and to receive traffic from the registrar, as described in [8] do not need to use the mechanism described in this section.

A user agent being registered by a third party may not be able to use the SIP Identity or P-Asserted-Identity 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

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 an escaped Refer-To header field identifying the recipient of the translation so that a REFER request sent to that URI will cause a MESSAGE request to be sent to the recipient.

On receiving a REFER request addressed to the URI a relay placed in a Trigger-Consent header field, the relay SHOULD send a MESSAGE request to the URI in the Refer-To header field with a permission document.

The following is the augmented Backus-Naur Form (BNF) [6] syntax of the Trigger-Consent header field. Some of its elements are defined in RFC 3261 [3].

```
Trigger-Consent     =  "Trigger-Consent" HCOLON trigger-cons-spec
                       *( COMMA trigger-cons-spec )
trigger-cons-spec   =  ( name-addr / addr-spec )
                      *( SEMI generic-param )
```

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

```
Trigger-Consent:<sip:relay@example.com
                ?Refer-To=<sip:recipient%40example.net>>
```

6. IANA Considerations

The IANA is requested to add 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:              [RFCxxxx]
```

Note to the RFC editor: substitute xxxx with the RFC number of this document.

The IANA is requested to add the following new SIP header field to the Header Fields subregistry under the SIP Parameters registry.

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

Note to the RFC editor: substitute xxxx with the RFC number of this document.

The IANA is requested to add the following new SIP header field to the Header Fields subregistry under the SIP Parameters registry.

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

Note to the RFC editor: substitute xxxx with the RFC number of this document.

7. Security Considerations

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

The specifications of mechanisms to manipulate translation logic at relays usually stress the importance of client authentication and authorization. Having relays authenticate and authorize clients manipulating their translation logic keeps unauthorized clients from adding recipients to a translation. However, this does not prevent authorized clients to add recipients to a translation without their consent. Additionally, some relays provide web interfaces for any client to add new recipients to the translation (e.g., many email mailing lists are operated in this way). In this situation, every client is considered authorized to manipulate the translation logic at the relay. This makes the use of this framework even more important. Therefore, it is RECOMMENDED that relays performing translations implement this framework.

As pointed out in Section 5.6.3, 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 [9].

The information provided by the Pending Additions event package can be sensitive. For this reason, as described in [10], relays need to use strong means for authentication and information confidentiality. SIPS URIs are a good mechanism to meet this requirement.

8. Acknowledges

Henning Schulzrinne, Jon Peterson, and Cullen Jennings provided useful ideas on this document.

9. References

9.1. Normative References

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

[2] 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.

[3] 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.

[4] 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.

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

[6] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[7] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-06 (work in progress), October 2005.

[8] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-03 (work in progress), March 2006.

[9] Camarillo, G., "A Document Format for Requesting Consent", draft-camarillo-sipping-consent-format-01 (work in progress),

June 2006.

[10] Camarillo, G., "The Session Initiation Protocol (SIP) Pending Additions Event Package", draft-camarillo-sipping-pending-additions-00 (work in progress), June 2006.

[11] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-11 (work in progress), May 2006.

[12] Rosenberg, J., "Extensible Markup Language (XML) Formats for Representing Resource Lists", draft-ietf-simple-xcap-list-usage-05 (work in progress), February 2005.

[13] Rosenberg, J., "Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP)", draft-ietf-sipping-consent-reqs-04 (work in progress), January 2006.

[14] Camarillo, G. and A. Roach, "Framework and Security Considerations for Session Initiation Protocol (SIP) Uniform Resource Identifier (URI)-List Services", draft-ietf-sipping-uri-services-05 (work in progress), January 2006.

9.2. Informative References

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

Authors' Addresses

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
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
Cisco Systems
2200 E. Pres. George Bush Turnpike
Richardson, TX 75082
USA

Email: dean.willis@softarmor.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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