SIPPING
Internet-Draft
Expires: April 8, 2006

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

October 5, 2005

A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)
draft-ietf-sipping-consent-framework-03.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 spam and 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) [1] 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 [3]) 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 the recipient without their explicit consent.

Receipt of these requests without explicit consent can cause a number of problems in SIP networks. These include spam and DoS (Denial of Service) attacks. These problems are described in more detail in a companion requirements document [11].

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

2. Definitions

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.

Target URI: The request-URI of an incoming request that arrives to an entity (e.g., a proxy) that will perform a translation operation.

Translation operation: Operation by which an entity (e.g., a proxy) 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

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 and translates the request URI into one or more next hop URIs to which it then delivers a request. The request URI of the incoming request is referred to as 'target URI' and the destination URI of the outgoing requests is referred to as 'recipient URIs', as shown in Figure 1.

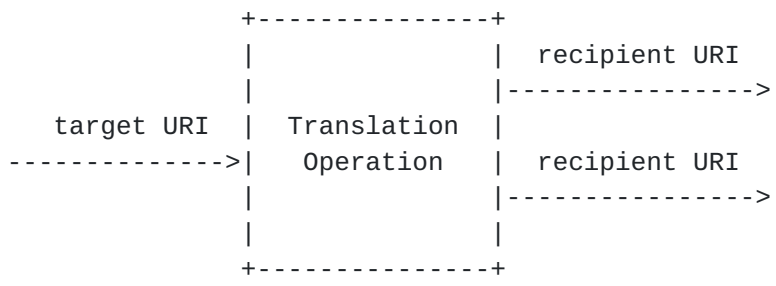

Figure 1: Translation operation

Thus, an essential aspect of a relay is that of translation. When a relay receives a request, it translates the request URI into one or more additional URIs. Or, more generally, it 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.

The reference architecture is shown in Figure 2. In this architecture, a user agent client (UAC) wishes to send a message 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 [13].

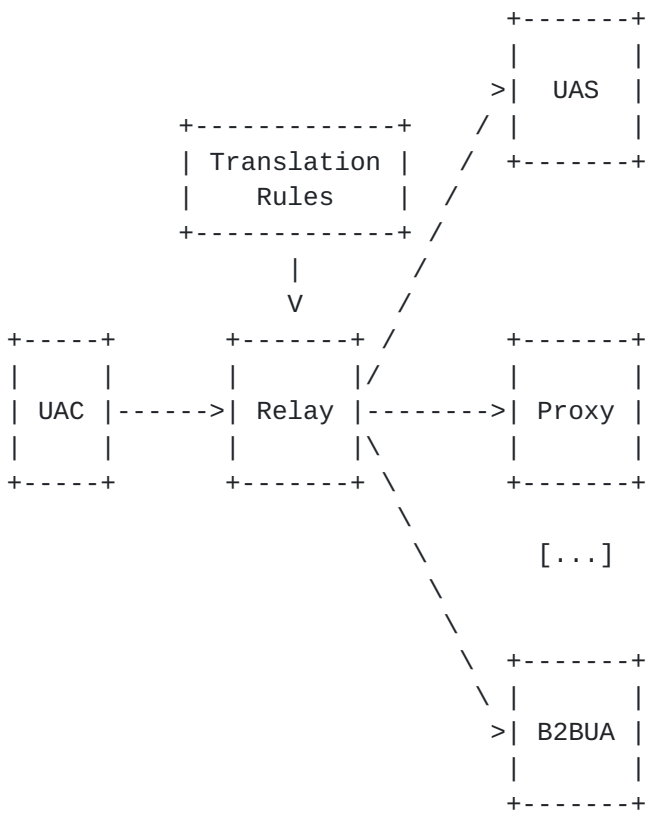

Figure 2: Relay performing a translation

3.1 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 permission to the Relay 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 needs to obtain permission from the new recipient in order to install the new translation logic. Relays ask recipients for permission using CONSENT 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 the administrator of that URI-list service adds 'bob@example.org' as a new recipient URI, the Relay sends a CONSENT request to 'bob@example.org' asking whether or not it is OK to perform the translation from 'friends@example.com' to 'bob@example.org' (CONSENT requests carry in their message bodies a permission document that describes the translation for which permissions are being requested). If the answer is positive, the new translation logic is installed at the relay. That is, the new recipient URI is added.

Note that the mechanism to be used to manipulate the translation logic of a particular relay depends on the relay. Two possible mechanisms to manipulate translation logic are XCAP [7] and REGISTER requests.

4. Permission Servers

When a CONSENT request sent by a relay arrives to the recipient URI to which it was sent, the receiving user can give 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 CONSENT 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 CONSENT requests for a user.

Permission servers inform users about new CONSENT requests using the 'grant-permission' event package. Figure Figure 3 illustrates this point.

The user associated with the recipient URI for which the relay will ask for permission subscribes [2] (1) to the 'grant-permission' event package at the permission server. This event package models the state of all pending CONSENT requests for a particular resource. When a new CONSENT request (3) arrives to the permission server, a NOTIFY (5) is sent to the user. This informs them that permission is needed for a particular sender. The NOTIFY contains a description of the translation for which permissions are being requested.

> There is a strong similarity between the 'winfo' event template-package [6] and the 'grant-permission' event package. Indeed, the grant-permission package is effectively a superset of watcherinfo. Once in place, presentities could use the grant-permission event package for presence in addition to all other services for which opt-in is being provided.

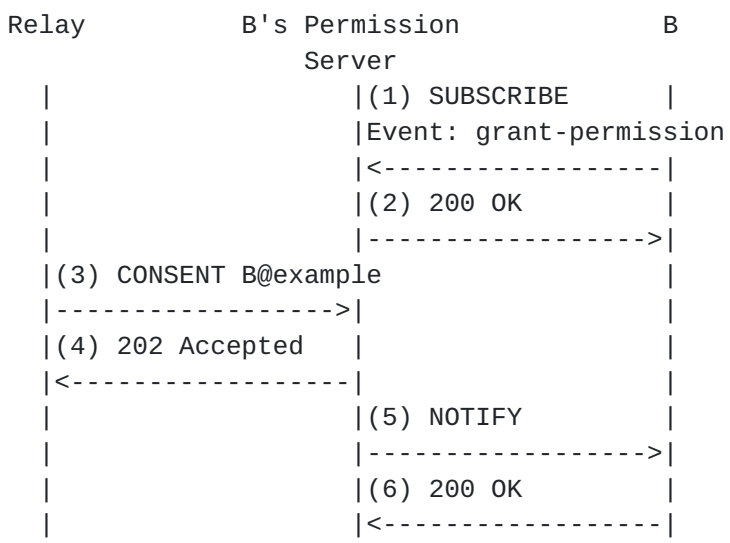

Figure 3: Permission server operation

5. Overview of Operations

This section provides an overview of this framework using an example of the prototypical call flow. The elements described in previous sections (i.e., relays, translations, and permission servers) play an essential role in this call flow.

Figure Figure 4 shows the complete process to add a recipient URI ('B@example.com') to the translation logic of a relay. The call flow starts with user B subscribing to the permission server using the 'grant-permission' event package. User B will be informed about the arrival of CONSENT requests addressed to 'B@example.com'.

User A attempts to add 'B@example.com' as a new recipient URI to the translation logic of the relay (3). User A uses XCAP [7] and the XML (Extensible Markup Language) format for representing resource lists [8] 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 'Permission Pending' state and informs user A (4).

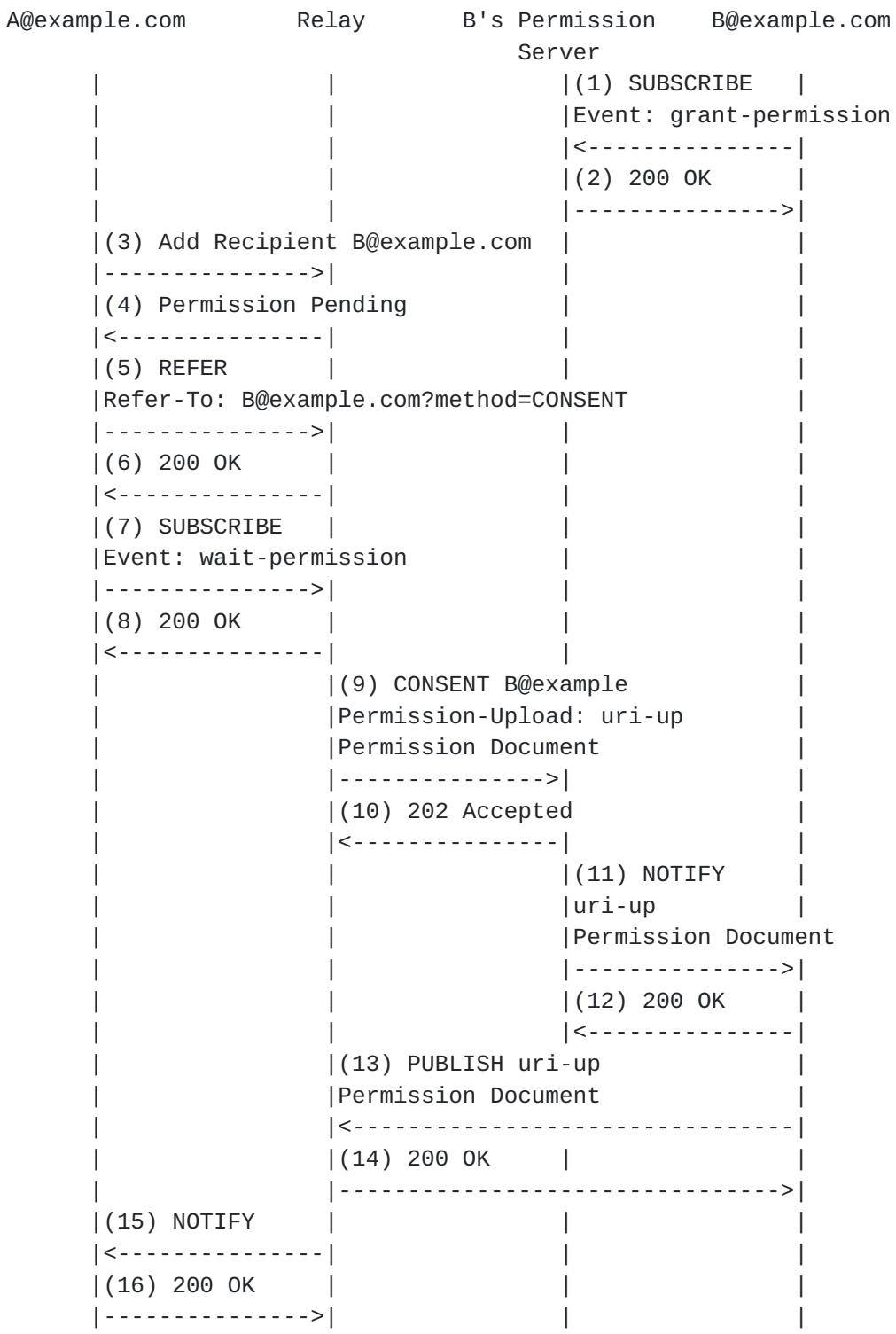

Figure 4: Prototypical call flow

5.1 Amplification Avoidance

Once 'B@example.com' is in the 'Permission Pending' state, the relay needs to ask user B for permission by sending a CONSENT 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 CONSENT 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 CONSENT requests towards those recipient URIs. This requirement is met by having users generate REFER requests [4] towards the relay. Each REFER request triggers the sending of a CONSENT request by the relay.

So, the relay sends user A the URI (4) where user A needs to send a REFER request. User A generates such a REFER request (5) and sends it to the relay. User A uses the 'norefersub' extension [5], which supreses the implicit subscription that is associated with REFER transactions. This is because user A is not interested in the result of the CONSENT transaction, but in whether or not user B will authorize the translation by providing the requested permission.

The relay provides a URI (4) where user A can subscribe to obtain information on whether or not user B provides the requested permission. User A subscribes to that URI using the 'wait-permission' event package (6).

5.2 Request for Permission

On receiving the REFER request (5), the relay generates a CONSENT request (9) towards 'B@example.com'. This CONSENT request carries a permission document, which describes the translation that needs to be authorized, and a URI where to upload the permission for that translation. User B will authorize the translation by uploading the permission document received in the CONSENT request into this URI.

When the permission document is uploaded to the URI provided by the relay (13), the relay needs to make sure that the permission document received was generated by user B and not by an attacker. The relay can use two methods to authenticate the permission document: SIP

identity or a return routability test. Both methods are described in Section 5.5. Relays using a return routability test to perform this authentication need to send the CONSENT request to a 'sips' URI.

5.3 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.

Operations Permitted: A set of specific methods or qualifiers for which the permission applies. For example, the permission may only grant relaying for INVITE requests and not for MESSAGE requests.

Signature: A digital signature over the rest of the permission, signed by an entity that can identify itself as the recipient URI. The signature is not always present.

Permission documents may contain wildcards. For example, a permission document may authorize any relay to forward INVITE 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.

The permission document in the CONSENT request (9) 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

Operations Permitted: INVITE

5.4 Permission Requested Notification

On receiving the CONSENT request (9), B's permission server sends a NOTIFY request (11) to user B, who had previously subscribed to the grant-permission event package (1). This NOTIFY request contains, the permission document, which describes the translation that needs to be authorized, and a URI where to upload the permission for that translation. Both the permission document and the URI to upload the permission are copied from the CONSENT request (9) into the NOTIFY request (11).

5.5 Permission Upload

On receiving the NOTIFY request (11), user B authorizes the translation described in the permission document received by uploading this permission document to the relay. User B uses a PUBLISH request (13) to upload the permission document to URI received in the NOTIFY request.

When the permission document is uploaded to the URI provided by the relay (13), the relay needs to make sure that the permission document received was generated by user B and not by an attacker. The relay can use two methods to authenticate the permission document: SIP identity or a return routability test.

5.5.1 SIP Identity

The SIP identity mechanism can be used to authenticate the sender of the PUBLISH request uploading the permission document. The relay checks that the originator of the PUBLISH request is the owner of the recipient URI in the permission document. Otherwise, the permission document is discarded.

5.5.2 Return Routability

SIP identity provides a good authentication mechanism for this type of scenario. 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 (e.g., with a long and random-looking user part) and places it in the CONSENT request (9). The recipient needs to upload the permission document to that URI.

Relays using a return routability test to perform this authentication need to send the CONSENT request to a SIPS URI. This ensures that attackers do not get access to the (unguessable) URI. Thus, the only user able to upload the permission document to the (unguessable) URI is the receiver of the CONSENT request.

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 Permission Granted Notification

On receiving the PUBLISH request (13), the relay sends a NOTIFY request (15) to inform user A that the permission for the translation has been received that the translation logic at the relay has been updated. That is, 'B@example.com' has been added as a recipient URI.

6. Permission Revocation

At any time, if a client wants to revoke any permission, it uses the same URI as before to upload, using a PUBLISH request, a new permission document that does not authorize the translation at the relay any longer. If a client loses this URI for some reason, it needs to wait until it receives a new request product of the translation. Such request which will contain a Permission-Upload header field and may contain a Permission-Used header field.

The client uses the URI in the Permission-Upload header field to upload the new permission document. The Permission-Used header field contains a URI (e.g., an HTTP URI) where the permission document the relay has for the translation can be obtained.

When permission document authorization is based on a return routability test, requests with a Permission-Upload header field need to be sent to a SIPS URI.

7. 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.

This type of URI-list services 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. Additionally, Figure 5 shows another way to add entries to that list.

If the relay receives a request that contains URIs for which the relay does not have permission, the relay ignores those URIs (i.e., does not send any request to them) and informs the user with a 470 (Consent Needed) response. Such a response contains a Consent-Needed header field with the URIs for which there is no permission, as shown in Figure 5. Additionally, the response also contains a Call-Info header field with two entries. The first entry is the URI where the user needs to send the REFER request that will trigger the relay to send a CONSENT request to those URIs. The value of the purpose parameter for this entry is 'trigger-permission-request'. The second entry is the URI where the user can subscribe in order to be informed on whether or not the relay receives permission from user B. The value of the purpose parameter for this entry is 'wait-permission'.

The rest of the process is similar to the one described in Section 5. Note, however, that for simplicity, Figure 5 does not show the split between user B's permission server and user agent.

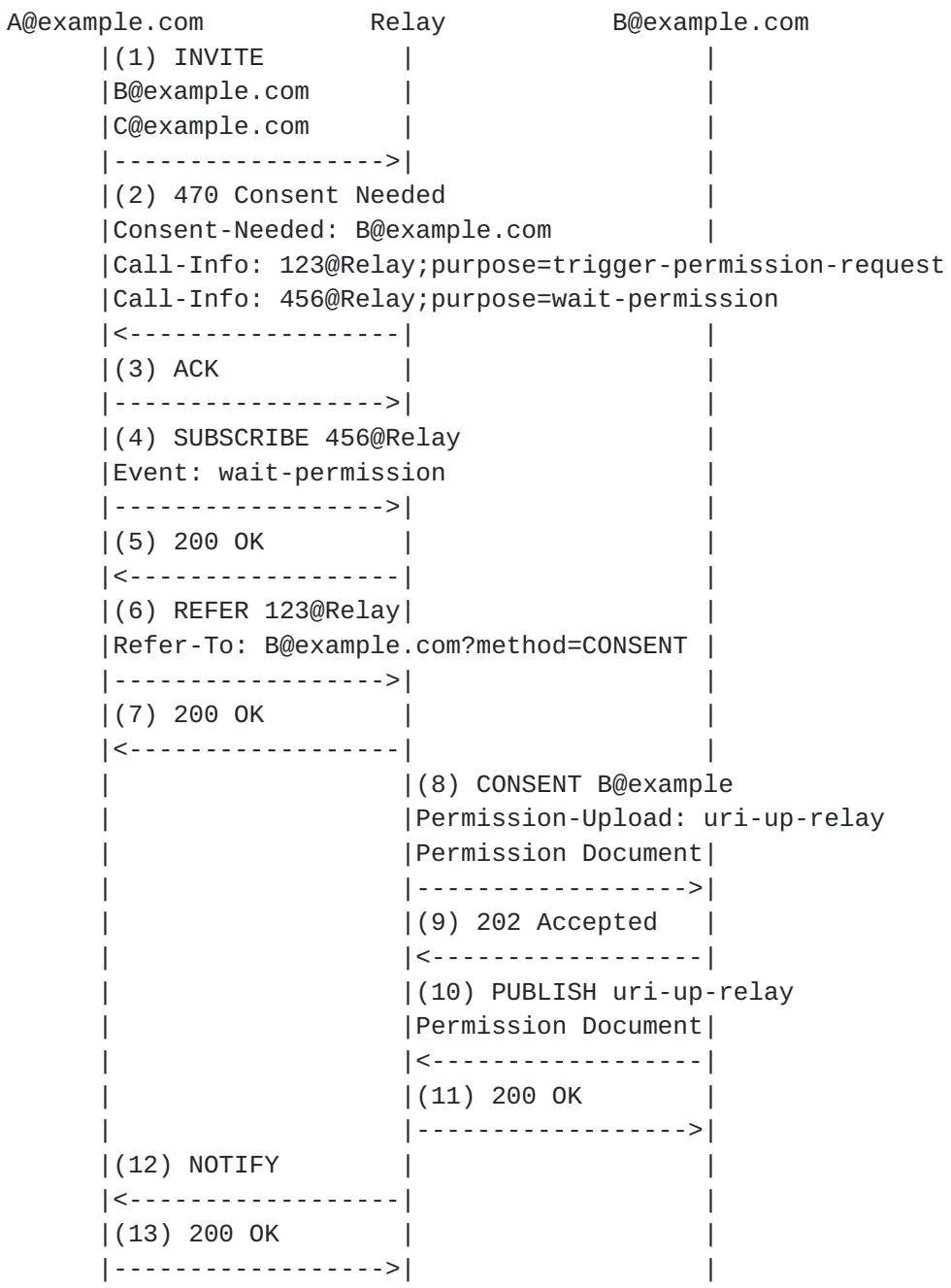

Figure 5: INVITE with a URI list in its body

8. 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., over the same connection as the registration). 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 6. User A performs a third-party registration (1) and receives a 200 (OK) response (2) with a Consent-Needed header field. This header field contains the URI for which there is no permission. That is, the URI the user is attempting to register. Additionally, the response also contains a Call-Info header field with the URI where the user needs to send the REFER request that will trigger the registrar to send a CONSENT request to the URI being registered. The purpose parameter for this entry in the Call-Info header field is 'trigger-permission-request'.

The user sends a REFER request (3) to the URI received in the Call-Info header field. In order to know whether or not the registrar receives the permission needed, the user subscribes (5) to the 'reg-event' event package. The rest of the process is similar to the one described in Section 5.

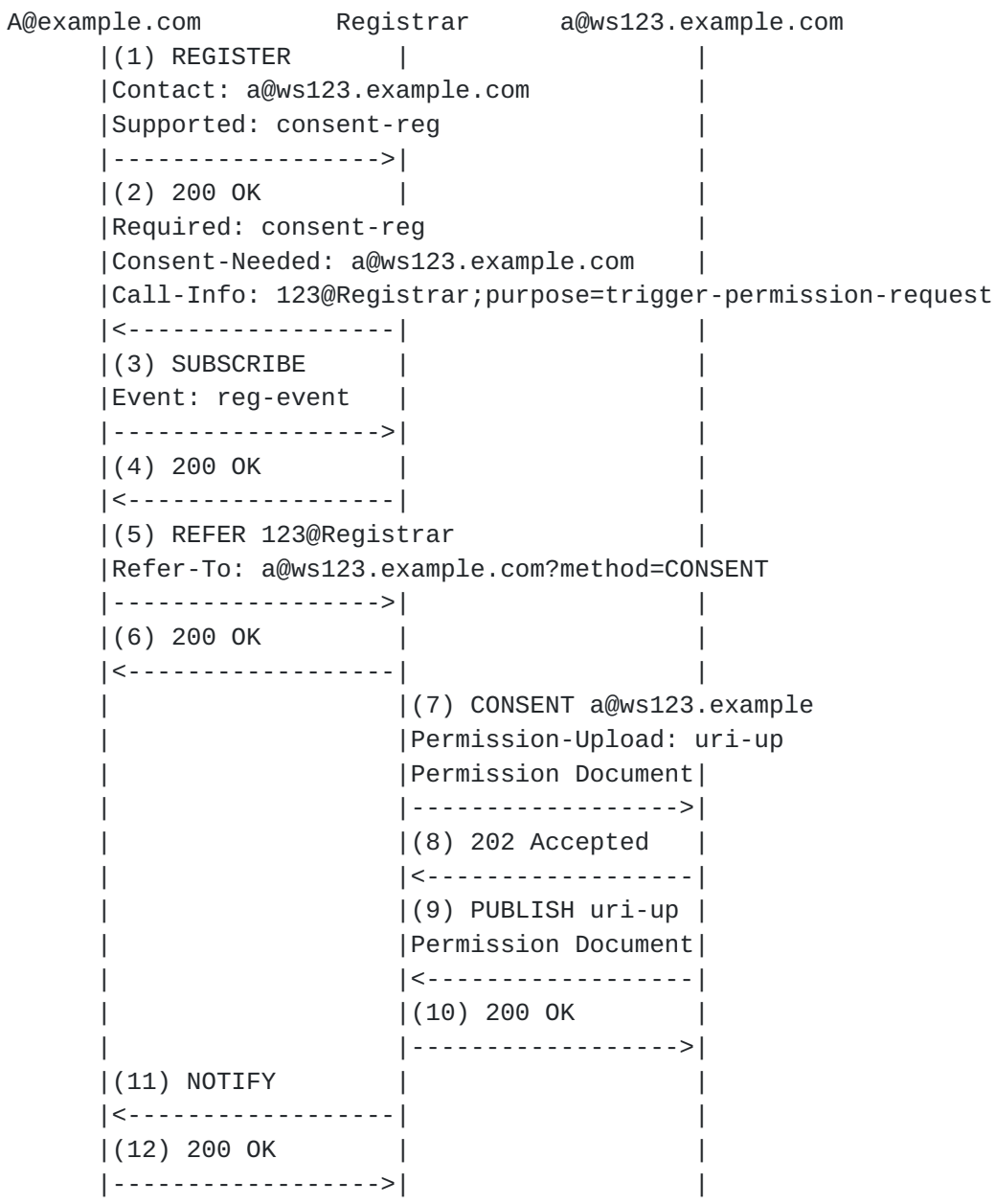

Figure 6: Registration

Permission documents used to authorize registrations are very general. For example, one such document may authorize the registrar to forward any request from any sender to a particular recipient 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.

9. IANA Considerations

This document does not require the IANA to take any actions.

10. Security Considerations

TBD.

Editor's note: we have to avoid that attackers provide permissions for translations that apply to other users (e.g., allow everyone to send traffic to a victim) and that attackers provide permissions for a translation that apply to them but routes to a victim (e.g., 3rd party registration that binds attacker@relay to victim@somewhere). For the former we need authentication (e.g., SIP identity) and for the latter we relay on the routing infrastructure to route CONSENTs to the same place the traffic will be sent to once permissions are obtained (i.e., a return routability test).

11. References

11.1 Normative References

[1] 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.

[2] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

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

[4] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[5] Levin, O., "Suppression of Session Initiation Protocol REFER Method Implicit Subscription", draft-ietf-sip-refer-with-norefersub-02 (work in progress), July 2005.

11.2 Informative References

[6] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

[7] Rosenberg, J., "The Extensible Markup Language (XML)

Configuration Access Protocol (XCAP)", draft-ietf-simple-xcap-07 (work in progress), June 2005.

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

[9] Schulzrinne, H., "A Document Format for Expressing Privacy Preferences", draft-ietf-geopriv-common-policy-05 (work in progress), July 2005.

[10] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[11] Rosenberg, J., "Requirements for Consent-Based Communications in the Session Initiation Protocol (SIP)", draft-ietf-sipping-consent-reqs-01 (work in progress), July 2005.

[12] Rosenberg, J., "Presence Authorization Rules", draft-ietf-simple-presence-rules-03 (work in progress), July 2005.

[13] Camarillo, G. and A. Roach, "Requirements and Framework for Session Initiation Protocol (SIP)Uniform Resource Identifier (URI)-List Services", draft-ietf-sipping-uri-services-03 (work in progress), April 2005.

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.