Network Working Group
Request for Comments: 3857
Category: Standards Track

J. Rosenberg
dynamicsoft

August 2004

A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)

Status of this Memo

Copyright Notice

Abstract

This document defines the watcher information template-package for the Session Initiation Protocol (SIP) event framework. Watcher information refers to the set of users subscribed to a particular resource within a particular event package. Watcher information changes dynamically as users subscribe, unsubscribe, are approved, or are rejected. A user can subscribe to this information, and therefore learn about changes to it. This event package is a template-package because it can be applied to any event package, including itself.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) event framework is described in RFC 3265 [1]. It defines a generic framework for subscription to, and notification of, events related to SIP systems. The framework defines the methods SUBSCRIBE and NOTIFY, and introduces the notion of a package. A package is a concrete application of the event framework to a particular class of events. Packages have been defined for user presence [5], for example.

This document defines a "template-package" within the SIP event framework. A template-package has all the properties of a regular SIP event package. However, it is always associated with some other event package, and can always be applied to any event package, including the template-package itself.

The template-package defined here is for watcher information, and is denoted with the token "winfo". For any event package, such as presence, there exists a set (perhaps an empty set) of subscriptions that have been created or requested by users trying to ascertain the state of a resource in that package. This set of subscriptions changes over time as new subscriptions are requested by users, old subscriptions expire, and subscriptions are approved or rejected by the owners of that resource. The set of users subscribed to a particular resource for a specific event package, and the state of their subscriptions, is referred to as watcher information. Since this state is itself dynamic, it is reasonable to subscribe to it in order to learn about changes to it. The watcher information event template-package is meant to facilitate exactly that - tracking the state of subscriptions to a resource in another package.

To denote this template-package, the name is constructed by appending ".winfo" to the name of whatever package is being tracked. For example, the set of people subscribed to presence is defined by the "presence.winfo" package.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP14, RFC 2119 [2] and indicate requirement levels for compliant implementations.

This document fundamentally deals with recursion - subscriptions to subscriptions. Therefore, the term "subscription" itself can be confusing in this document. To reduce confusion, the term "watcherinfo subscription" refers to a subscription to watcher information, and the term "watcherinfo subscriber" refers to a user that has subscribed to watcher information. The term "watcherinfo notification" refers to a NOTIFY request sent as part of a watcherinfo subscription. When the terms "subscription", "subscriber", and "notification" are used unqualified, they refer to the "inner" subscriptions, subscribers and notifications - those that are being monitored through the watcherinfo subscriptions. We also use the term "watcher" to refer to a subscriber to the "inner" resource. Information on watchers is reported through watcherinfo subscriptions.

3. Usage Scenarios

There are many useful applications for the watcher information template-package.

3.1. Presence Authorization

The motivating application for this template-package is presence authorization. When user A subscribes to the presence of user B, the subscription needs to be authorized. Frequently, that authorization needs to occur through direct user intervention. For that to happen, B's software needs to become aware that a presence subscription has been requested. This is supported through watcher information. B's client software would SUBSCRIBE to the watcher information for the presence of B:

```
SUBSCRIBE sip:B@example.com SIP/2.0
Via: SIP/2.0/UDP pc34.example.com;branch=z9hG4bKnashds7
From: sip:B@example.com;tag=123s8a
To: sip:B@example.com
Call-ID: 9987@pc34.example.com
Max-Forwards: 70
CSeq: 9887 SUBSCRIBE
Contact: sip:B@pc34.example.com
Event: presence.winfo
```

The policy of the server is such that it allows B to subscribe to its own watcher information. So, when A subscribes to B's presence, B gets a notification of the change in watcher information state:

```
NOTIFY sip:B@pc34.example.com SIP/2.0
Via: SIP/2.0/UDP server.example.com;branch=z9hG4bKna66g
From: sip:B@example.com;tag=xyz887
To: sip:B@example.com;tag=123s8a
Call-ID: 9987@pc34.example.com
Max-Forwards: 70
CSeq: 1288 NOTIFY
Contact: sip:B@server.example.com
Event: presence.winfo
Content-Type: application/watcherinfo+xml
Content-Length: ...

<?xml version="1.0"?>
<watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
             version="0" state="full">
   <watcher-list resource="sip:B@example.com" package="presence">
     <watcher id="7768a77s" event="subscribe"
              status="pending">sip:A@example.com</watcher>
   </watcher-list>
</watcherinfo>
```

This indicates to B that A has subscribed, and that the subscription is pending (meaning, it is awaiting authorization). B's software can alert B that this subscription is awaiting authorization. B can then set policy for that subscription.

3.2. Blacklist Alerts

Applications can subscribe to watcher information in order to provide value-added features. An example application is "blacklist alerts". In this scenario, an application server maintains a list of known "bad guys". A user, Joe, signs up for service with the application provider, presumably by going to a web page and entering in his presence URI. The application server subscribes to the watcher information for Joe's presence. When someone attempts to SUBSCRIBE to Joe's user presence, the application learns of this subscription as a result of its watcher info subscription. It checks the watcher's URI against the database of known bad guys. If there is a match, it sends email to Joe letting him know about this.

For this application to work, Joe needs to make sure that the application is allowed to subscribe to his presence.winfo.

4. Package Definition

This section fills in the details needed to specify an event package as defined in Section 4.4 of RFC 3265 [1].

4.1. Event Package Name

RFC 3265 [1] requires package definitions to specify the name of their package or template-package.

The name of this template-package is "winfo". It can be applied to any other package. Watcher information for any package foo is denoted by the name "foo.winfo". Recursive template-packaging is explicitly allowed (and useful), so that "foo.winfo.winfo" is a valid package name.

4.2. Event Package Parameters

RFC 3265 [1] requires package and template-package definitions to specify any package specific parameters of the Event header field.

No package specific Event header field parameters are defined for this event template-package.

4.3. SUBSCRIBE Bodies

RFC 3265 [1] requires package or template-package definitions to define the usage, if any, of bodies in SUBSCRIBE requests.

A SUBSCRIBE request for watcher information MAY contain a body. This body would serve the purpose of filtering the watcherinfo subscription. The definition of such a body is outside the scope of this specification. For example, in the case of presence, the body might indicate that notifications should contain full state every time something changes, and that the time the subscription was first made should not be included in the watcherinfo notifications.

A SUBSCRIBE request for a watcher information package MAY be sent without a body. This implies the default watcherinfo subscription filtering policy has been requested. The default policy is:

- Watcherinfo notifications are generated every time there is any change in the state of the watcher information.

- Watcherinfo notifications triggered from a SUBSCRIBE contain full state (the list of all watchers that the watcherinfo subscriber is permitted to know about). Watcherinfo notifications triggered from a change in watcher state only contain information on the watcher whose state has changed.

Of course, the server can apply any policy it likes to the subscription.

4.4. Subscription Duration

RFC 3265 [1] requires package definitions to define a default value for subscription durations, and to discuss reasonable choices for durations when they are explicitly specified.

Watcher information changes as users subscribe to a particular resource for some package, or their subscriptions time out. As a result, the state of watcher information can change very dynamically, depending on the number of subscribers for a particular resource in a given package. The rate at which subscriptions time out depends on how long a user maintains its subscription. Typically, watcherinfo subscriptions will be timed to span the lifetime of the subscriptions being watched, and therefore range from minutes to days.

As a result of these factors, it is difficult to define a broadly useful default value for the lifetime of a watcherinfo subscription. We arbitrarily choose one hour. However, clients SHOULD use an Expires header field to specify their preferred duration.

4.5. NOTIFY Bodies

RFC 3265 [1] requires package definitions to describe the allowed set of body types in NOTIFY requests, and to specify the default value to be used when there is no Accept header field in the SUBSCRIBE request.

The body of the watcherinfo notification contains a watcher information document. This document describes some or all of the watchers for a resource within a given package, and the state of their subscriptions. All watcherinfo subscribers and notifiers MUST support the application/watcherinfo+xml format described in [3], and MUST list its MIME type, application/watcherinfo+xml, in any Accept header field present in the SUBSCRIBE request.

Other watcher information formats might be defined in the future. In that case, the watcherinfo subscriptions MAY indicate support for other formats. However, they MUST always support and list application/watcherinfo+xml as an allowed format.

Of course, the watcherinfo notifications generated by the server MUST be in one of the formats specified in the Accept header field in the SUBSCRIBE request. If no Accept header field was present, the notifications MUST use the application/watcherinfo+xml format described in [3].

4.6. Notifier Processing of SUBSCRIBE Requests

RFC 3265 [1] specifies that packages should define any package-specific processing of SUBSCRIBE requests at a notifier, specifically with regards to authentication and authorization.

The watcher information for a particular package contains sensitive information. Therefore, all watcherinfo subscriptions SHOULD be authenticated and then authorized before approval. Authentication MAY be performed using any of the techniques available through SIP, including digest, S/MIME, TLS or other transport specific mechanisms [4]. Authorization policy is at the discretion of the administrator, as always. However, a few recommendations can be made.

It is RECOMMENDED that user A be allowed to subscribe to their own watcher information for any package. This is true recursively, so that it is RECOMMENDED that a user be able to subscribe to the watcher information for their watcher information for any package.

It is RECOMMENDED that watcherinfo subscriptions for some package foo for user A be allowed from some other user B, if B is an authorized subscriber to A within the package foo. However, it is RECOMMENDED

that the watcherinfo notifications sent to B only contain the state of B's own subscription. In other words, it is RECOMMENDED that a user be allowed to monitor the state of their own subscription.

To avoid infinite recursion of authorization policy, it is RECOMMENDED that only user A be allowed to subscribe to foo.winfo.winfo for user A, for any foo. It is also RECOMMENDED that by default, a server does not authorize any subscriptions to foo.winfo.winfo.winfo or any other deeper recursions.

4.7. Notifier Generation of NOTIFY Requests

The SIP Event framework requests that packages specify the conditions under which notifications are sent for that package, and how such notifications are constructed.

Each watcherinfo subscription is associated with a set of "inner" subscriptions being watched. This set is defined by the URI in the Request URI of the watcherinfo SUBSCRIBE request, along with the parent event package of the watcherinfo subscription. The parent event package is obtained by removing the trailing ".winfo" from the value of the Event header field from the watcherinfo SUBSCRIBE request. If the Event header field in the watcherinfo subscription has a value of "presence.winfo", the parent event package is "presence". If the Event header field has a value of "presence.winfo.winfo", the parent event package is "presence.winfo". Normally, the URI in the Request URI of the watcherinfo SUBSCRIBE identifies an address-of-record within the domain. In that case, the set of subscriptions to be watched are all of the subscriptions for the parent event package that have been made to the resource in the Request URI of the watcherinfo SUBSCRIBE. However, the Request URI can contain a URI that identifies any set of subscriptions, including the subscriptions to a larger collection of resources. For example, sip:all-resources@example.com might be defined within example.com to refer to all resources. In that case, a watcherinfo subscription for "presence.winfo" to sip:all-resources@example.com is requesting notifications any time the state of any presence subscription for any resource within example.com changes. A watcherinfo notifier MAY generate a notification any time the state of any of the watched subscriptions changes.

Because a watcherinfo subscription is made to a collection of subscriptions, the watcher information package needs a model of subscription state. This is accomplished by specifying a subscription Fine State Machine (FSM), described below, which governs the subscription state of a user in any package. Watcherinfo notifications MAY be generated on transitions in this state machine. It's important to note that this FSM is just a model of the

subscription state machinery maintained by a server. An implementation would map its own state machines to this one in an implementation-specific manner.

4.7.1. The Subscription State Machine

The underlying state machine for a subscription is shown in Figure 1. It derives almost entirely from the descriptions in RFC 3265 [1], but adds the notion of a waiting state.

When a SUBSCRIBE request arrives, the subscription FSM is created in the init state. This state is transient. The next state depends on whether policy exists for the subscription. If there is an existing policy that determines that the subscription is forbidden, it moves into the terminated state immediately, where the FSM can be destroyed. If there is existing policy that determines that the subscription is authorized, the FSM moves into the active state. This state indicates that the subscriber will receive notifications.

If, when a subscription arrives, there is no authorization policy in existence, the subscription moves into the pending state. In this state, the server is awaiting an authorization decision. No notifications are generated on changes in presence state (an initial NOTIFY will have been delivered as per RFC 3265 [1]), but the subscription FSM is maintained. If the authorization decision comes back positive, the subscription is approved, and moves into the active state. If the authorization is negative, the subscription is rejected, and the FSM goes into the terminated state. It is possible that the authorization decision can take a very long time. In fact, no authorization decision may arrive until after the subscription itself expires. If a pending subscription suffers a timeout, it moves into the waiting state. At any time, the server can decide to end a pending or waiting subscription because it is concerned about allocating memory and CPU resources to unauthorized subscription state. If this happens, a "giveup" event is generated by the server, moving the subscription to terminated.

The waiting state is similar to pending, in that no notifications are generated. However, if the subscription is approved or denied, the FSM enters the terminated state, and is destroyed. Furthermore, if another subscription is received to the same resource, from the same watcher, for the same event package, event package parameters and filter in the body of the SUBSCRIBE request (if one was present initially), the FSM enters the terminated state with a "giveup" event, and is destroyed. This transition occurs because, on arrival of a new subscription with identical parameters, it will enter the pending state, making the waiting state for the prior subscription redundant. The purpose of the waiting state is so that a user can

fetch watcherinfo state at any time, and learn of any subscriptions that arrived previously (and which may arrive again) which require an authorization decision. Consider an example. A subscribes to B. B has not defined policy about this subscription, so it moves into the pending state. B is not "online", so that B's software agent cannot be contacted to approve the subscription. The subscription expires. Let's say it were destroyed. B logs in, and fetches its watcherinfo state. There is no record of the subscription from A, so no policy decision is made about subscriptions from A. B logs off. A refreshes its subscription. Once more, the subscription is pending since no policy is defined for it. This process could continue indefinitely. The waiting state ensures that B can find out about this subscription attempt.

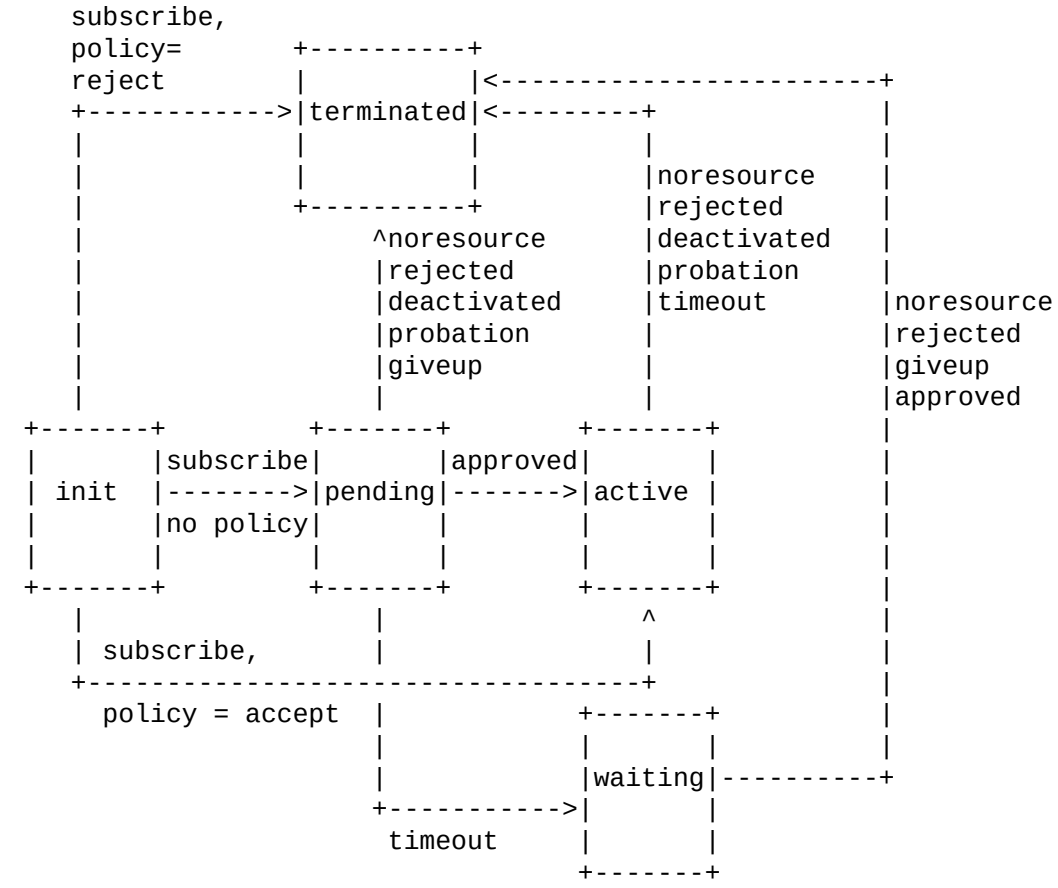

Figure 1: Subscription State Machine

The waiting state is also needed to allow for authorization of fetch attempts, which are subscriptions that expire immediately.

Of course, policy may never be specified for the subscription. As a result, the server can generate a giveup event to move the waiting subscription to the terminated state. The amount of time to wait before issuing a giveup event is system dependent.

The giveup event is generated in either the waiting or pending states to destroy resources associated with unauthorized subscriptions. This event is generated when a giveup timer fires. This timer is set to a timeout value when entering either the pending or waiting states. Servers need to exercise care in selecting this value. It needs to be large in order to provide a useful user experience; a user should be able to log in days later and see that someone tried to subscribe to them. However, allocating state to unauthorized subscriptions can be used as a source of DoS attacks. Therefore, it is RECOMMENDED that servers that retain state for unauthorized subscriptions add policies which prohibit a particular subscriber from having more than some number of pending or waiting subscriptions.

At any time, the server can deactivate a subscription. Deactivation implies that the subscription is discarded without a change in authorization policy. This may be done in order to trigger refreshes of subscriptions for a graceful shutdown or subscription migration operation. A related event is probation, where a subscription is terminated, and the subscriber is requested to wait some amount of time before trying again. The meaning of these events is described in more detail in Section 3.2.4 of RFC 3265 [1].

A subscription can be terminated at any time because the resource associated with that subscription no longer exists. This corresponds to the noresource event.

4.7.2. Applying the State Machine

The server MAY generate a notification to watcherinfo subscribers on a transition of the state machine. Whether it does or not is policy dependent. However, several guidelines are defined.

Consider some event package foo. A subscribes to B for events within that package. A also subscribes to foo.winfo for B. In this scenario (where the subscriber to foo.winfo is also a subscriber to foo for the same resource), it is RECOMMENDED that A receive watcherinfo notifications only about the changes in its own subscription. Normally, A will receive notifications about changes in its subscription to foo through the Subscription-State header field. This will frequently obviate the need for a separate subscription to foo.winfo. However, if such a subscription is performed by A, the foo.winfo notifications SHOULD NOT report any

state changes which would not be reported (because of authorization policy) in the Subscription-State header field in notifications on foo.

As a general rule, when a watcherinfo subscriber is authorized to receive watcherinfo notifications about more than one watcher, it is RECOMMENDED that watcherinfo notifications contain information about those watchers which have changed state (and thus triggered a notification), instead of delivering the current state of every watcher in every watcherinfo notification. However, watcherinfo notifications triggered as a result of a fetch operation (a SUBSCRIBE with Expires of 0) SHOULD result in the full state of all watchers (of course, only those watchers that have been authorized to be divulged to the watcherinfo subscriber) to be present in the NOTIFY.

Frequently, states in the subscription state machine will be transient. For example, if an authorized watcher performs a fetch operation, this will cause the state machine to be created, transition from init to active, and then from active to terminated, followed by a destruction of the FSM. In such cases, watcherinfo notifications SHOULD NOT be sent for any transient states. In the prior example, the server wouldn't send any notifications, since all of the states are transient.

4.8. Subscriber Processing of NOTIFY Requests

RFC 3265 [1] expects packages to specify how a subscriber processes NOTIFY requests in any package specific ways, and in particular, how it uses the NOTIFY requests to construct a coherent view of the state of the subscribed resource. Typically, the watcherinfo NOTIFY will only contain information about those watchers whose state has changed. To construct a coherent view of the total state of all watchers, a watcherinfo subscriber will need to combine NOTIFYs received over time. This details of this process depend on the document format. See [3] for details on the application/watcherinfo+xml format.

4.9. Handling of Forked Requests

The SIP Events framework mandates that packages indicate whether or not forked SUBSCRIBE requests can install multiple subscriptions.

When a user wishes to obtain watcher information for some resource for package foo, the SUBSCRIBE to the watcher information will need to reach a collection of servers that have, unioned together, complete information about all watchers on that resource for package foo. If there are a multiplicity of servers handling subscriptions for that resource for package foo (for load balancing reasons,

typically), it is very likely that no single server will have the complete set of watcher information. There are several solutions in this case. This specification does not mandate a particular one, nor does it rule out others. It merely ensures that a broad range of solutions can be built.

One solution is to use forking. The system can be designed so that a SUBSCRIBE for watcher information arrives at a special proxy which is aware of the requirements for watcher information. This proxy would fork the SUBSCRIBE request to all of the servers which could possibly maintain subscriptions for that resource for that package. Each of these servers, whether or not they have any current subscribers for that resource, would accept the watcherinfo subscription. Each needs to accept because they may all eventually receive a subscription for that resource. The watcherinfo subscriber would receive some number of watcherinfo NOTIFY requests, each of which establishes a separate dialog. By aggregating the information across each dialog, the watcherinfo subscriber can compute full watcherinfo state. In many cases, a particular dialog might never generate any watcherinfo notifications; this would happen if the servers never receive any subscriptions for the resource.

In order for such a system to be built in an interoperable fashion, all watcherinfo subscribers MUST be prepared to install multiple subscriptions as a result of a multiplicity of NOTIFY messages in response to a single SUBSCRIBE.

Another approach for handling the server multiplicity problem is to use state agents. See Section 4.11 for details.

4.10. Rate of Notifications

RFC 3265 [1] mandates that packages define a maximum rate of notifications for their package.

For reasons of congestion control, it is important that the rate of notifications not become excessive. As a result, it is RECOMMENDED that the server not generate watcherinfo notifications for a single watcherinfo subscriber at a rate faster than once every 5 seconds.

4.11. State Agents

RFC 3265 [1] asks packages to consider the role of state agents in their design.

State agents play an important role in this package. As discussed in Section 4.9, there may be a multiplicity of servers sharing the load of subscriptions for a particular package. A watcherinfo

subscription might require subscription state spread across all of those servers. To handle that, a farm of state agents can be used. Each of these state agents would know the entire watcherinfo state for some set of resources. The means by which the state agents would determine the full watcherinfo state is outside the scope of this specification. When a watcherinfo subscription is received, it would be routed to a state agent that has the full watcherinfo state for the requested resource. This server would accept the watcherinfo subscription (assuming it was authorized, of course), and generate watcherinfo notifications as the watcherinfo state changed. The watcherinfo subscriber would only have a single dialog in this case.

5. Example Usage

The following section discusses an example application and call flows using the watcherinfo package.

In this example, a user Joe, sip:joe@example.com provides presence through the example.com presence server. Joe subscribes to his own watcher information, in order to learn about people who subscribe to his presence, so that he can approve or reject their subscriptions. Joe sends the following SUBSCRIBE request:

```
SUBSCRIBE sip:joe@example.com SIP/2.0
Via: SIP/2.0/UDP pc34.example.com;branch=z9hG4bKnashds7
From: sip:joe@example.com;tag=123aa9
To: sip:joe@example.com
Call-ID: 9987@pc34.example.com
CSeq: 9887 SUBSCRIBE
Contact: sip:joe@pc34.example.com
Event: presence.winfo
Max-Forwards: 70
```

The server responds with a 401 to authenticate, and Joe resubmits the SUBSCRIBE with credentials (message not shown). The server then authorizes the subscription, since it allows Joe to subscribe to his own watcher information for presence. It responds with a 200 OK:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc34.example.com;branch=z9hG4bKnashds8
  ;received=192.0.2.8
From: sip:joe@example.com;tag=123aa9
To: sip:joe@example.com;tag=xyzygg
Call-ID: 9987@pc34.example.com
CSeq: 9988 SUBSCRIBE
Contact: sip:server19.example.com
Expires: 3600
Event: presence.winfo
```

The server then sends a NOTIFY with the current state of presence.winfo for joe@example.com:

```
NOTIFY sip:joe@pc34.example.com SIP/2.0
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaii
From: sip:joe@example.com;tag=xyzygg
To: sip:joe@example.com;tag=123aa9
Call-ID: 9987@pc34.example.com
CSeq: 1288 NOTIFY
Contact: sip:server19.example.com
Event: presence.winfo
Subscription-State: active
Max-Forwards: 70
Content-Type: application/watcherinfo+xml
Content-Length: ...

<?xml version="1.0"?>
<watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
             version="0" state="full">
  <watcher-list resource="sip:joe@example.com" package="presence">
    <watcher id="77ajsyy76" event="subscribe"
             status="pending">sip:A@example.com</watcher>
  </watcher-list>
</watcherinfo>
```

Joe then responds with a 200 OK to the NOTIFY:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaii
  ;received=192.0.2.7
From: sip:joe@example.com;tag=xyzygg
To: sip:joe@example.com;tag=123aa9
Call-ID: 9987@pc34.example.com
CSeq: 1288 NOTIFY
```

The NOTIFY tells Joe that user A currently has a pending subscription. Joe then authorizes A's subscription through some means. This causes a change in the status of the subscription (which moves from pending to active), and the delivery of another notification:

```
NOTIFY sip:joe@pc34.example.com SIP/2.0
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaij
From: sip:joe@example.com;tag=xyzygg
To: sip:joe@example.com;tag=123aa9
Call-ID: 9987@pc34.example.com
CSeq: 1289 NOTIFY
Contact: sip:server19.example.com
Event: presence.winfo
Subscription-State: active
Max-Forwards: 70
Content-Type: application/watcherinfo+xml
Content-Length: ...

<?xml version="1.0"?>
<watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
             version="1" state="partial">
  <watcher-list resource="sip:joe@example.com" package="presence">
    <watcher id="77ajsyy76" event="approved"
             status="active">sip:A@example.com</watcher>
  </watcher-list>
</watcherinfo>
```

B then responds with a 200 OK to the NOTIFY:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server19.example.com;branch=z9hG4bKnasaij
  ;received=192.0.2.7
From: sip:joe@example.com;tag=xyzygg
To: sip:joe@example.com;tag=123aa9
Call-ID: 9987@pc34.example.com
CSeq: 1289 NOTIFY
```

6. Security Considerations

6.1. Denial of Service Attacks

Watcher information generates notifications about changes in the state of watchers for a particular resource. It is possible for a single resource to have many watchers, resulting in the possibility of a large volume of notifications. This makes watcherinfo subscription a potential tool for denial of service attacks. Preventing these can be done through a combination of sensible authorization policies and good operating principles.

First, when a resource has a lot of watchers, watcherinfo subscriptions to that resource should only be allowed from explicitly authorized entities, whose identity has been properly authenticated. That prevents a watcherinfo NOTIFY stream from being generated from subscriptions made by an attacker.

Even when watcherinfo subscriptions are properly authenticated, there are still potential attacks. For example, consider a valid user, T, who is to be the target of an attack. T has subscribed to their own watcher information. The attacker generates a large number of subscriptions (not watcherinfo subscriptions). If the server creates subscription state for unauthenticated subscriptions, and reports those changes in watcherinfo notifications, user T would receive a flood of watcherinfo notifications. In fact, if the server generates a watcherinfo notification when the subscription is created, and another when it is terminated, there will be an amplification by a factor of two. The amplification would actually be substantial if the server generates full state in each watcherinfo notification. Indeed, the amount of data sent to T would be the square of the data generated by the attacker! Each of the N subscriptions generated by the attacker would result in a watcherinfo NOTIFY being sent to T, each of which would report on up to N watchers. To avoid this, servers should never generate subscription state for unauthenticated SUBSCRIBE requests, and should never generate watcherinfo notifications for them either.

6.2. Divulging Sensitive Information

Watcher information indicates what users are interested in a particular resource. Depending on the package and the resource, this can be very sensitive information. For example, in the case of presence, the watcher information for some user represents the friends, family, and business relations of that person. This information can be used for a variety of malicious purposes.

One way in which this information can be revealed is eavesdropping. An attacker can observe watcherinfo notifications, and learn this information. To prevent that, watchers MAY use the sips URI scheme when subscribing to a watcherinfo resource. Notifiers for watcherinfo MUST support TLS and sips as if they were a proxy (see Section 26.3.1 of RFC 3261).

SIP encryption, using S/MIME, MAY be used end-to-end for the transmission of both SUBSCRIBE and NOTIFY requests.

Another way in which this information can be revealed is through spoofed subscriptions. These attacks can be prevented by authenticating and authorizing all watcherinfo subscriptions. In order for the notifier to authenticate the subscriber, it MAY use HTTP Digest (Section 22 of RFC 3261). As a result, all watchers MUST support HTTP Digest. This is a redundant requirement, however, since all SIP user agents are mandated to support it by RFC 3261.

7. IANA Considerations

This specification registers an event template package as specified in Section 6.2 of RFC 3265 [1].

Package Name: winfo

Template Package: yes

Published Specification: RFC 3857

Person to Contact: Jonathan Rosenberg, jdrosen@jdrosen.net.

8. Acknowledgements

The authors would like to thank Adam Roach, Allison Mankin and Brian Stucker for their detailed comments.

9. Normative References

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

[2] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Rosenberg, J., "An Extensible Markup Language (XML) Based Format for Watcher Information", RFC 3858, August 2004.

[4] 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.

10. Informative References

[5] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, July 2004.

11. Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054

EMail: jdrosen@dynamicsoft.com

12. Full Copyright Statement

Intellectual Property

Acknowledgement

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