Network Working Group
Internet-Draft
Expires: August, 2003

L. Dusseault
Xythos

February 23, 2003

Requirements for Server-to-Server Event Pipeline Protocols
draft-dusseault-s2s-event-reqs

Status of this Memo

This Internet-Draft will expire on May 28, 2003.

Copyright Notice

Abstract

Server-to-server event pipelines are required in situations where an event source generates events for a number of end-users (or simply generates events without an idea of who the end recipients might be), and sends these events to an event or notification aggregator service. These kinds of event source servers are common in messaging (for example, voice messaging servers and calendar servers might send events to a user's main messaging server or to a single-purpose notification server). A general set of requirements is outlined here for the server-to-server aspect of these kinds of scenarios.

1. Introduction

The need for this requirements document arose in the VPIM and Lemonade working groups in 2002, where a protocol for server-to-server messaging events was being considered. Such a protocol cannot be considered in isolation: it must be considered together with other protocols and applications with which the protocol interacts. This includes the nature of the application server generating events, the kind of events that may be generated, and other protocols that may be used later to deliver notifications. It is also important to have a basic idea how this protocol will be used and extended beyond its original description.

This document considers security requirements, internationalization requirements, and architectural requirements. Also, it considers notification info-set requirements. The notification info-set refers to the set of information that must be present or can be generated for an event notification so that it can be delivered using another protocol without having to make inaccurate guesses at any important information. For example, it must be possible to tell what host an event arises on so that when the event is delivered the end application knows what host to go to for more information about the event. Since the source machine may be host multiple site names which all use the same event pipeline, this information must be in the event notification. This and other possible examples will be discussed later.

The security requirements section includes some discussion of the threat model.

Examples from unified messaging are typically used (voice mail, email, calendaring) but occasionally examples from more speculative areas are used to illustrate the need for a certain requirement.

Here is a representative list of the kinds of application problems IETF groups are would like to solve related to notifications.

- SNAP proposal in lemonade and vpim working groups, intended for email/voice mailbox event delivery from source server to aggregation server

- RFC 3265 [1] specifies SIP notifications

- WebDAV WG has seen a specific proposal for events (GENA) relating to resources. For example, authors may be notified when their document becomes unlocked.

- CalSched WG has discussed notifications for appointment reminders.

- HTTP/OPES/WEBI WG has discussed notifications of web cache timeout events.

- XMPP involves notifications of presence changes. Instant messages can almost be seen as events.

- LDAP-EXT WG has discussed a notification protocol

- IPP WG has discussed requirements for a printing event notification

2. Architecture Considerations

2.1 Connection Characteristics

A server-to-server messaging event pipeline can be used to aggregate notifications for eventual delivery to end-user. It can also be used to update information on an application server. For example, notifications of new voice mail events could arrive at a regular email server, and the regular email server could store a count of new voice mail events until the user logs on and asks for all new messages. Multiple event source servers, of the same type or different types, could all set up a pipeline to the source sink server.

If the messaging servers all have accounts for the same users, then the pipeline is likely to carry a large number of events. Modern email servers can handle 10,000 concurrent users, and people can recieve 300 new email a day, thus 3 million new email events can be generated daily, before even counting other types of email events, or other types of events (voice messages, calendaring, instant messaging and presence). This kind of load must be assumed for a server-to-server messaging event protocol.

The server-to-server pipeline could be efficiently maintained as an open TCP connection, to avoid recurring connection startup costs. Privacy and authentication could be ensured with a transport layer encryption protocol like TLS [REF]. A TCP connection also allows for somewhat simpler acknowledgement of received events, assuming acks are required.

UDP could be considered for efficient delivery of small messages, but in the absence of IPSEC, some kind of encryption would have to be used to protect privacy. A per-UDP-packet encryption mechanism like S/MIME is probably unusable due to length limitations, but it would be possible to define an encryption mechanism that encrypted each packet using the same shared secret. However, this might not provide authorization/authentication, unless provisioning work was used to configure the servers to share the secret and not connect to servers without such a secret. Note that with UDP acks are somewhat more complicated because the event source server needs to correlate acks to events that may have been sent a while ago even if more recent events have already been acked.

TODO: think more about TCP/UDP and per-packet encryption.

Whether TCP or UDP is used, a request/response model protocol is likely inappropriate. Request/response protocols rely very little on state between requests, so instead state-like information is repeated

on each request. In this application, some state information is likely to change only rarely if ever. For example, HTTP requires headers like content-length and content-type on every single request. In an event pipeline, content-type (of the notification itself) will likely be unnecessary or rarely necessary.

2.2 ACKs

Are acknowledgements required?

2.3 Interoperation with other Notification systems

A server receiving a large number of notifications relating to many users mailboxes may be able to deal with those notifications itself, but may also need to pass those notifications on to the end-user's client at some point. This may result in some need for consistent or consistently transformable addresses. Email addresses can probably be used in some messaging scenarios, but what about other addresses? What if the server receiving the notifications is not an email server but another kind of notification aggregator? Can a SIP event server receive a "new voice message" notification and know what address to use to set up a replay session with a SIP client?

2.4 Subscriptions

Is it a requirement that the recipient of notifications be able to choose which kinds are required? This would allow the server to filter out notifications for accounts that are inactive, and notifications of uninteresting types. It might greatly reduce the quantity of notifications used in the system.

Another possible advantage of subscriptions is that if a subscription ID is assigned, the sender of the notification can more efficiently identify the context of the notification for the recipient.

3. Security Requirements

3.1 Privacy

Messaging events, and any other event having to do with user account activity, and in fact many events of many kinds, may have sensitive or private information. The event may only contain the information that an email arrived from a certain address at a certain time with a certain subject, but even this much information can be considered sensitive.

A protocol for event notification MUST require minimum-to-implement privacy protection mechanisms. It is not sufficient to leave these mechanisms up to implementors, because two independent implementations must have at least one shared privacy mechanism in order to be able to interoperate and still protect privacy.

3.2 Integrity

Event integrity considerations include the following threats

- An attacker might want to cause certain events not to get delivered, and for this lack of delivery to go unnoticed.

- An attacker might generate false event notifications which are received and treated as legitimate events.

- An attacker might cause duplicate delivery of otherwise legitimate event notifications. The danger is if the duplicate delivery goes unnoticed. For example, a legitimate event signaling the order of expensive equipment might get delivered several times appearing as separate events.

- Event notifications might be changed en route.

A protocol for event notification MUST protect from event notification integrity attacks by requiring a minimum-to-implement integrity protection mechanism. Notice also that the architecture MUST protect the notification aggregator from impersonation attacks, which allow streams of false events to be inserted into the system by pretending to be a legitimate event source.

It's probably not possible to protect completely from an attacker causing events to be lost (the attacker might be able to cause the connection between the servers to go down). However, it should not be possible for events to be removed from an otherwise successfully maintained connection.

3.3 Authentication

Authentication must be considered in order to ensure privacy and protect from integrity attacks. It may be necessary for a notification aggregation server to provide authentication for the end user in order to prove that access to certain kinds of event information is allowed.

3.4 Authorization

There are a number of potential authorization issues.

- Permission to subscribe to certain information: e.g. is the notification aggregation server allowed to subscribe to a certain user's calendar events

- Permission to send events: e.g. is the event source server allowed to open a pipeline and send events to the event sink server

- Permission to route events to a particular user: e.g. is the event source server allowed to send events to a particular user via a notification aggregation server

Some of these authorization issues may be mitigated by opening a connection in a certain direction. For example, if a notification aggregation server opens a connection to the event source server, it can be assumed the event source server is allowed to send events over this connection.

Sometimes authorization issues are dealt with as a matter of provisioning. It is assumed that the administrator configures the source to send events to the sink, and configures the sink to receive events from the source. However, if this is the approach used, the protocol specification MUST discuss how an implementation should behave such that provisioning does solve the problem.

4. Internationalization Requirements

4.1 Character Set

An event notification protocol must support Unicode (must require implementations to support Unicode) in order to convey information about events. For example, the name of a WebDAV resource that becomes unlocked (and generates an unlock event) might be Unicode).

4.2 Language Negotiation

An event notification may not need language negotiation. The events occur with given text, with or without language negotiation. The event sink server may not even be capable of determining the language preference for an event if the event is going to be multiplexed to multiple users.

Because some languages share some unicode characters, an event notification protocol will sometimes need to convey the language of human-readable strings. The language information allows proper selection of font for display.

In order to minimize language issues, error codes should be machine parsable.

5. Extensibility Requirements

5.1 Event Type Names

The most likely path for extensibility is simply to define new event types, new event names, and the information that goes along with events. By "event name" I mean the name given to a type of event so that the event sink server can tell what type of event it is. Thus the event name could be something like "new-email" in the namespace "http://www.example.com/namespaces/email-application" (a XML QName or Qualified Name [REF]).

Event names must be machine readable so that applications can dispatch based on event type.

Event names should be extensible by any number of independently operating groups without serious risk of collision. For example, the WebDAV group should be able to define a resource unlock event, and not have to coordinate with other groups that might wish to define an event for unlocking other kinds of objects. This problem could be solved with use of XML namespaces, because each group can independently define its own namespace with certain rules that make duplicate namespaces unlikely, then the group can define event names that are qualified with that namespace.

References

[1] Roach, A., "SIP-Specific Event Notification", RFC 3265, June 2002.

Author's Address

Lisa Dusseault
Xythos Software, Inc.
2064 Edgewood Dr.
Palo Alto, CA 94303
US

EMail: lisa@xythos.com

Full Copyright Statement

Acknowledgement

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