TRANS
Internet-Draft
Intended status: Experimental
Expires: April 30, 2015

L. Nordberg
NORDUnet

October 27, 2014

Transparency Gossip
draft-linus-trans-gossip-00

Abstract

This document describes Transparency Gossip, a gossip service for Certificate Transparency and other transparency systems.

Gossip peers connect to a gossip service and start sending and receiving gossip messages. Gossip transports register with a gossip service and transfer messages between other gossip services and local peers.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 30, 2015.

Table of Contents

1. Introduction

Regarding scope, gossiping in any Transparency system, such as Certificate Transparency [RFC6962], can be divided in two distinct pieces; a general gossip protocol and the specifics for the Transparency system at hand.

A general gossip protocol defines a message format, validation rules for messages and a mechanism for plugging in different transport protocols.

The specifics for a Transparency system include which types of data to gossip about, with whom to gossip, what particular log data to

gossip about and how to act on gossip. The last two, what particular data objects to send and how to deal with data objects received fall into the category of strategy or policy and should probably not be standardised.

The scope for this document is a general gossip protocol.

Terminology used in this document:

peer - Actors gossiping about log data. They are the endpoints in a global gossiping system, deciding what to send and how to treat gossip from other peers.

transport - Network programs connecting the local gossiping system to other gossiping systems on the internet.

gossip service - A server connecting local peers and transports.

2. Problem

Public append-only untrusted logs have to be monitored for consistency, i.e. that they don't break their promise of not rewriting history. Monitors and other log clients need to exchange information about monitored logs in order to be able to detect a partitioning attack.

A partitioning attack is when a log serves different views of the log to different clients. Each client would be able to verify the append-only attribute of the log while being the only client seeing this particular view.

The problem of disseminating information about log data that (locally) has been confirmed to be illegitimate is hard to deal with because of privacy implications for log clients and is not addressed by this document specifically. The gossip message format specified can still be useful for this task though.

3. Overview

This document defines a gossip service and a gossip message format used by peers and transports to exchange data about logs over the

internet.

Separating the gossiping actors, called peers, from the actual sending and receiving of gossip messages makes it possible to make various transport mechanisms available without having to add knowledge about transport protocols to peers.

The gossip service connects peers to one or more transports responsible for communicating with other gossip services with the help of specific internet protocols such as HTTP.

4. Transports

A gossip transport is responsible for sending and receiving gossip messages over a specific protocol, like HTTP or XMPP.

The way a transport uses its external protocol to convey gossip messages is specified by the transport itself and out of scope for this document.

A transport registers with a gossip service, either by being compiled into the service, being dynamically loaded into it or by connecting to it over a socket endpoint, local or remote. In the case of a connecting transport, the transport has to authenticate with the service by sending a shared secret in an Section 6.1 message.

4.1. Defined transports

- gossip-transport-https = "gossip-transport-https"
- gossip-transport-tls = TBD
- gossip-transport-xmpp = TBD
- gossip-transport-dns = TBD
- gossip-transport-tor = TBD
- gossip-transport-webrtc = TBD

5. Message format and processing

The message format is described in Section 6.5.

5.1. Validation of received messages

Peers MUST validate all gossip messages, incoming and outgoing, by verifying GOSSIP-MSG 'gossip-data' according to the rules of the log indicated by 'log-id'. Messages with an unknown log id or which signature don't check out correctly MUST be silently discarded.

Transports MAY validate gossip messages before forwarding them.

Peers MAY respond to an incoming message by sending one or more messages back to the transport it was received from.

[FIXME should this document stick to specifying what the gossip service does and leave peers and transports alone?]

6. Gossip service protocol

All messages are ASCII messages in S-expression format sent over a reliable data stream.

TODO: change to JSON object [RFC7159] encoding since that's used in CT

6.1. AUTHENTICATE-REQUEST

AUTHENTICATE-REQUEST messages are sent from transports to the gossip service.

- command = "gossip-authenticate-request-0"
- shared-secret = base64-encoded string

6.2. AUTHENTICATE-RESPONSE

AUTHENTICATE-RESPONSE messages are sent from the gossip service to a transport as a response to an AUTHENTICATE-REQUEST message.

- command = "gossip-authenticate-response-0"

- response = "ok" | "bad-auth"

6.3. ENUMERATE-TRANSPORTS-REQUEST

ENUMERATE-TRANSPORTS-REQUEST messages are sent from peers to the gossip service.

- command = "gossip-enumerate-transports-request-0"

6.4. ENUMERATE-TRANSPORTS-RESPONSE

ENUMERATE-TRANSPORTS-RESPONSE messages are sent from the gossip service to a peer in response to an ENUMERATE-TRANSPORTS-REQUEST message.

- command = "gossip-enumerate-transports-response-0"

- transports = list of registered transports

6.5. GOSSIP-MSG

Gossip messages are sent by peers and transports to the gossip service.

The gossip service MUST forward messages from peers to all registered transports given in 'destination' of the message.

The gossip service MUST forward messages from transports to all connected peers.

An outgoing message is sent by a peer and received by a transport.

An incoming message is sent by a transport and received by one or more peers.

Example of an outgoing message, i.e. sent from a peer and received by a transport:

(gossip-msg

```
  (command gossip-msg-0)
  (destination (gossip-transport-https gossip-transport-tbd))
  (timestamp 1414396810000)
  (log-id
    467a28a27c206a26cdf7b36cc93e8c598e93592ef49ad3a8dc523a35e1f4bc0c)
  (gossip-data b3V0Z29pbmcgZ29zc2lwIGRhdGEK))
```

Example of an incoming message, i.e. sent from a transport and received by one or more peers:

```
(gossip-msg
  (command gossip-msg-0)
  (source gossip-transport-https)
  (timestamp 1414396811000)
  (log-id
    467a28a27c206a26cdf7b36cc93e8c598e93592ef49ad3a8dc523a35e1f4bc0c)
  (gossip-data aW5jb21pbmcgZ29zc2lwIGRhdGEK))
```

6.5.1. Components of a message

- command = "gossip-msg-0"

- timestamp = 64bit integer in decimal

  NTP time when the message was received by the gossip service, in milliseconds since the epoch

- destination = transports

  destination specifies which transport(s) to send an outgoing message over

  destination is meaningful only in outgoing messages and MUST be ignored by peers

  transports = list of 'transport'

  transport = string | "all"

  transport is a string containing the name of a registered transport

or the special string "all"

transport "all" means that the message should be sent to all registered transports

o source = transport | ""

an incoming message has source set to the transport it came in over

source is meaningful only in incoming messages and MUST be ignored by transports

o log-id = SHA-256 hash of a log's public key, 32 octets

the log id of the transparency log gossiped about

o gossip-data = opaque string, base64-encoded

gossip-data exact contents depend on what kind of data is being gossiped but MUST include a signature made by the log indicated by 'log-id'.

7. Examples

This section describes how a gossiping system could be implemented for Certificate Transparency.

7.1. CT clients

web-browser ct-monitor <-- gossip peer

```
           \       /
         gossip-daemon                    <-- gossip service
           /       \
 socket-transport   https-transport       <-- gossip transport
         |            |
   web-browser[0]   tls-lib
```

[0] same instance as the gossip peer "web-browser"

```
          web-server                      <-- gossip peer
              |
         gossip-daemon                    <-- gossip service
              |
         socket-transport                 <-- gossip transport
              |
          web-server[1]
```

[1] same instance as the gossip peer "web-server"

The daemon listens to two separate sockets, one for peers and one for transports.

Peers (top row) connect to the daemon client socket and send and receive gossip messages. A message contains one or more transport ids and gossip data. The transport ids in outgoing messages are used to select which transport(s) the gossip data is allowed to be sent over. Transport ids in incoming messages reflect which transport they were received over.

Transports are either built into the daemon or registered with the daemon by connecting to the daemon transport socket and identifying itself with a string (the transport id). This registering process will require authentication.

(A less complex alternative to such a registering scheme would be to have built-in transports only, one of which is "echo". Adding a tag (think IMAP) to messages would make it possible for peers to match sent and received messages, making an echo transport useful for programs like web browsers which already have the transport ready for use.)

8. Security considerations

- TODO: sending of sensitive data to the gossip service
- TODO: protection against bad actors

    * flooding attacks flushing gossip pools [belongs in *T-specific specs?]

  - TODO: gossiping messages being blocked

  - TODO: transports authenticating with gossip services

9. Open questions

  - remove transports lacking a draft?

10. IANA considerations

TBD

11. Contributors

The author would like to thank Ben Laurie for the idea with a gossip daemon.

12. References

12.1. Normative References

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

12.2. Informative References

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, June 2013.

Author's Address

Linus Nordberg
NORDUnet

Email: linus@nordu.net