Stream: Internet Engineering Task Force (IETF)
RFC: 8968
Category: Standards Track
Published: January 2021
ISSN: 2070-1721
Authors: A. Décimo, *IRIF, University of Paris-Diderot*; D. Schinazi, *Google LLC*; J. Chroboczek, *IRIF, University of Paris-Diderot*

RFC 8968
Babel Routing Protocol over Datagram Transport Layer Security

Abstract

The Babel Routing Protocol does not contain any means to authenticate neighbours or provide integrity or confidentiality for messages sent between them. This document specifies a mechanism to ensure these properties using Datagram Transport Layer Security (DTLS).

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8968.

Table of Contents

1. Introduction

The Babel routing protocol [RFC8966] does not contain any means to authenticate neighbours or protect messages sent between them. Because of this, an attacker is able to send maliciously crafted Babel messages that could lead a network to route traffic to an attacker or to an under-resourced target, causing denial of service. This document specifies a mechanism to prevent such attacks using Datagram Transport Layer Security (DTLS) [RFC6347].

1.1. Specification of Requirements

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Applicability

The protocol described in this document protects Babel packets with DTLS. As such, it inherits the features offered by DTLS, notably authentication, integrity, optional replay protection, confidentiality, and asymmetric keying. It is therefore expected to be applicable in a wide range of environments.

There exists another mechanism for securing Babel, namely Message Authentication Code (MAC) authentication for Babel (Babel-MAC) [RFC8967]. Babel-MAC only offers basic features, namely authentication, integrity, and replay protection with a small number of symmetric keys. A comparison of Babel security mechanisms and their applicability can be found in [RFC8966].

Note that Babel over DTLS provides a single authentication domain, meaning that all nodes that have the right credentials can convey any and all routing information.

DTLS supports several mechanisms by which nodes can identify themselves and prove possession of secrets tied to these identities. This document does not prescribe which of these mechanisms to use; details of identity management are left to deployment profiles of Babel over DTLS.

2. Operation of the Protocol

Babel over DTLS requires some changes to how Babel operates. First, DTLS is a client-server protocol, while Babel is a peer-to-peer protocol. Second, DTLS can only protect unicast communication, while Babel packets can be sent to both unicast and multicast destinations.

2.1. DTLS Connection Initiation

Babel over DTLS operates on a different port than unencrypted Babel. All Babel over DTLS nodes **MUST** act as DTLS servers on a given UDP port and **MUST** listen for unencrypted Babel traffic on another UDP port, which **MUST** be distinct from the first one. The default port for Babel over DTLS is registered with IANA as the "babel-dtls" port (UDP port 6699, see Section 4), and the port exchanging unencrypted Babel traffic is registered as the "babel" port (UDP port 6696, see Section 5 of [RFC8966]).

When a Babel node discovers a new neighbour (generally by receiving an unencrypted multicast Babel packet), it compares the neighbour's IP address with its own, using network byte ordering. If a node's address is lower than the recently discovered neighbour's address, it acts as a client and connects to the neighbour. In other words, the node with the lowest address is the DTLS client for this pairwise relationship. As an example, fe80::1:2 is considered lower than fe80::2:1.

The node acting as DTLS client initiates its DTLS connection from an ephemeral UDP port. Nodes **SHOULD** ensure that new client DTLS connections use different ephemeral ports from recently used connections to allow servers to differentiate between the new and old DTLS connections. Alternatively, nodes could use DTLS connection identifiers [DTLS-CID] as a higher-entropy mechanism to distinguish between connections.

When a node receives a new DTLS connection, it **MUST** verify that the source IP address is either an IPv6 link-local address or an IPv4 address belonging to the local network; if it is neither, it **MUST** reject the connection. Nodes use mutual authentication (authenticating both client and server); clients **MUST** authenticate servers and servers **MUST** authenticate clients. Implementations **MUST** support authenticating peers against a local store of credentials. If either node fails to authenticate its peer against its local policy, it **MUST** abort the DTLS handshake. The guidance given in [BCP195] **MUST** be followed to avoid attacks on DTLS. Additionally, nodes **MUST** only negotiate DTLS version 1.2 or higher. Nodes **MUST** use DTLS replay protection to prevent attackers from replaying stale information. Nodes **SHOULD** drop packets that have been reordered by more than two IHU (I Heard You) intervals, to avoid letting attackers make stale information last longer. If a node receives a new DTLS connection from a neighbour to whom it already has a connection, the node **MUST NOT** discard the older connection until it has completed the handshake of the new one and validated the identity of the peer.

2.2. Protocol Encoding

Babel over DTLS sends all unicast Babel packets protected by DTLS. The entire Babel packet, from the Magic byte at the start of the Babel header to the last byte of the Babel packet trailer, is sent protected by DTLS.

2.3. Transmission

When sending packets, Babel over DTLS nodes **MUST NOT** send any TLVs over the unprotected "babel" port, with the exception of Hello TLVs without the Unicast flag set. Babel over DTLS nodes **MUST NOT** send any unprotected unicast packets. This ensures the confidentiality of the information sent in Babel packets (e.g., the network topology) by only sending it encrypted by DTLS. Unless some out-of-band neighbour discovery mechanism is available, nodes **SHOULD** periodically send unprotected Multicast Hellos to ensure discovery of new neighbours. In order to maintain bidirectional reachability, nodes can either rely entirely on unprotected Multicast Hellos, or send protected Unicast Hellos in addition to the Multicast Hellos.

Since Babel over DTLS only protects unicast packets, implementors may implement Babel over DTLS by modifying an implementation of Babel without DTLS support and replacing any TLV previously sent over multicast with a separate TLV sent over unicast for each neighbour. TLVs

previously sent over multicast can be replaced with the same contents over unicast, with the exception of Hellos as described above. Some implementations could also change the contents of IHU TLVs when converting to unicast in order to remove redundant information.

2.4. Reception

Babel over DTLS nodes can receive Babel packets either protected over a DTLS connection or unprotected directly over the "babel" port. To ensure the security properties of this mechanism, unprotected packets are treated differently. Nodes **MUST** silently ignore any unprotected packet sent over unicast. When parsing an unprotected packet, a node **MUST** silently ignore all TLVs that are not of type Hello. Nodes **MUST** also silently ignore any unprotected Hello with the Unicast flag set. Note that receiving an unprotected packet can still be used to discover new neighbours, even when all TLVs in that packet are silently ignored.

2.5. Neighbour Table Entry

It is **RECOMMENDED** for nodes to associate the state of their DTLS connection with their neighbour table. When a neighbour entry is flushed from the neighbour table (Appendix A of [RFC8966]), its associated DTLS state **SHOULD** be discarded. The node **SHOULD** send a DTLS close_notify alert to the neighbour if it believes the link is still viable.

2.6. Simultaneous Operation of Babel over DTLS and Unprotected Babel on a Node

Implementations **MAY** implement both Babel over DTLS and unprotected Babel. Additionally, a node **MAY** simultaneously run both Babel over DTLS and unprotected Babel. However, a node running both **MUST** ensure that it runs them on separate interfaces, as the security properties of Babel over DTLS rely on ignoring unprotected Babel packets (other than Multicast Hellos). An implementation **MAY** offer configuration options to allow unprotected Babel on some interfaces but not others, which effectively gives nodes on that interface the same access as authenticated nodes; however, this **SHOULD NOT** be done unless that interface has a mechanism to authenticate nodes at a lower layer (e.g., IPsec).

2.7. Simultaneous Operation of Babel over DTLS and Unprotected Babel on a Network

If Babel over DTLS and unprotected Babel are both operated on the same network, the Babel over DTLS implementation will receive unprotected Multicast Hellos and attempt to initiate a DTLS connection. These connection attempts can be sent to nodes that only run unprotected Babel, who will not respond. Babel over DTLS implementations **SHOULD** therefore rate-limit their DTLS connection attempts to avoid causing undue load on the network.

3. Interface Maximum Transmission Unit Issues

Compared to unprotected Babel, DTLS adds header, authentication tag, and possibly block-size padding overhead to every packet. This reduces the size of the Babel payload that can be carried. This document does not relax the packet size requirements in Section 4 of [RFC8966] but recommends that DTLS overhead be taken into account when computing maximum packet size.

More precisely, nodes **SHOULD** compute the overhead of DTLS depending on the ciphersuites in use and **SHOULD NOT** send Babel packets larger than the interface maximum transmission unit (MTU) minus the overhead of IP, UDP, and DTLS. Nodes **MUST NOT** send Babel packets larger than the attached interface's MTU adjusted for known lower-layer headers (at least UDP and IP) or 512 octets, whichever is larger, but not exceeding 2^{16} - 1 adjusted for lower-layer headers. Every Babel speaker **MUST** be able to receive packets that are as large as any attached interface's MTU adjusted for UDP and IP headers or 512 octets, whichever is larger. Note that this requirement on reception does not take into account the overhead of DTLS because the peer may not have the ability to compute the overhead of DTLS, and the packet may be fragmented by lower layers.

Note that distinct DTLS connections can use different ciphers, which can have different amounts of per-packet overhead. Therefore, the MTU to one neighbour can be different from the MTU to another neighbour on the same link.

4. IANA Considerations

IANA has registered a UDP port number, called "babel-dtls", for use by Babel over DTLS:

Service Name: babel-dtls

Port Number: 6699

Transport Protocols: UDP only

Description: Babel Routing Protocol over DTLS

Assignee: IESG, iesg@ietf.org

Contact: IETF Chair, chair@ietf.org

Reference: RFC 8968

Service Code: None

5. Security Considerations

A malicious client might attempt to perform a high number of DTLS handshakes with a server. As the clients are not uniquely identified by the protocol until the handshake completes and can be obfuscated with IPv6 temporary addresses, a server needs to mitigate the impact of such an attack. Note that attackers might attempt to keep in-progress handshakes open for as long as

possible by using variants on the attack commonly known as Slowloris [SLOWLORIS]. Mitigating these attacks might involve limiting the rate of handshakes from a given subnet or more advanced denial of service avoidance techniques beyond the scope of this document.

Babel over DTLS allows sending Multicast Hellos unprotected; attackers can therefore tamper with them. For example, an attacker could send erroneous values for the Seqno and Interval fields, causing bidirectional reachability detection to fail. While implementations **MAY** use Multicast Hellos for link quality estimation, they **SHOULD** also emit protected Unicast Hellos to prevent this class of denial-of-service attack.

While DTLS provides protection against an attacker that replays valid packets, DTLS is not able to detect when an active on-path attacker intercepts valid packets and resends them at a later time. This attack could be used to make a node believe it has bidirectional reachability to a neighbour even though that neighbour has disconnected from the network. To prevent this attack, nodes **MUST** discard the DTLS state associated with a neighbour after a finite time of not receiving valid DTLS packets. This can be implemented by, for example, discarding a neighbour's DTLS state when its associated IHU timer fires. Note that relying solely on the receipt of Hellos is not sufficient as Multicast Hellos are sent unprotected. Additionally, an attacker could save some packets and replay them later in hopes of propagating stale routing information at a later time. This can be mitigated by discarding received packets that have been reordered by more than two IHU intervals.

6. References

6.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <https://www.rfc-editor.org/info/bcp195>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8966] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", RFC 8966, DOI 10.17487/RFC8966, January 2021, <https://www.rfc-editor.org/info/rfc8966>.

6.2. Informative References

[DTLS-CID] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", Work in Progress, Internet-Draft, draft-ietf-tls-dtls-connection-id-08, 2 November 2020, <https://tools.ietf.org/html/draft-ietf-tls-dtls-connection-id-08>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <https://www.rfc-editor.org/info/rfc7918>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/info/rfc7924>.

[RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <https://www.rfc-editor.org/info/rfc8094>.

[RFC8967] Dô, C., Kolodziejak, W., and J. Chroboczek, "MAC Authentication for the Babel Routing Protocol", RFC 8967, DOI 10.17487/RFC8967, January 2021, <https://www.rfc-editor.org/info/rfc8967>.

[SLOWLORIS] Hansen, R., "Slowloris HTTP DoS", June 2009, <https://web.archive.org/web/20150315054838/http://ha.ckers.org/slowloris/>.

Appendix A. Performance Considerations

To reduce the number of octets taken by the DTLS handshake, especially the size of the certificate in the ServerHello (which can be several kilobytes), Babel peers can use raw public keys [RFC7250] or the Cached Information Extension [RFC7924]. The Cached Information Extension avoids transmitting the server's certificate and certificate chain if the client has cached that information from a previous TLS handshake. TLS False Start [RFC7918] can reduce round trips by allowing the TLS second flight of messages (ChangeCipherSpec) to also contain the (encrypted) Babel packet.

Acknowledgments

The authors would like to thank Roman Danyliw, Donald Eastlake, Thomas Fossati, Benjamin Kaduk, Gabriel Kerneis, Mirja Kühlewind, Antoni Przygienda, Henning Rogge, Dan Romascanu, Barbara Stark, Markus Stenberg, Dave Taht, Martin Thomson, Sean Turner, and Martin Vigoureux for their input and contributions. The performance considerations in this document were inspired from the ones for DNS over DTLS [RFC8094].

Authors' Addresses

Antonin Décimo
IRIF, University of Paris-Diderot
Paris
France
Email: antonin.decimo@gmail.com

David Schinazi
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

Juliusz Chroboczek
IRIF, University of Paris-Diderot
Case 7014
75205 Paris CEDEX 13
France
Email: jch@irif.fr