Network Working Group
Request for Comments: 3706
Category: Informational

G. Huang
S. Beaulieu
D. Rochefort
Cisco Systems, Inc.

February 2004

A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers

Status of this Memo

Copyright Notice

Abstract

This document describes the method detecting a dead Internet Key Exchange (IKE) peer that is presently in use by a number of vendors. The method, called Dead Peer Detection (DPD) uses IPSec traffic patterns to minimize the number of IKE messages that are needed to confirm liveness. DPD, like other keepalive mechanisms, is needed to determine when to perform IKE peer failover, and to reclaim lost resources.

Table of Contents

1. Introduction

When two peers communicate with IKE [2] and IPSec [3], the situation may arise in which connectivity between the two goes down unexpectedly. This situation can arise because of routing problems, one host rebooting, etc., and in such cases, there is often no way for IKE and IPSec to identify the loss of peer connectivity. As such, the SAs can remain until their lifetimes naturally expire, resulting in a "black hole" situation where packets are tunneled to oblivion. It is often desirable to recognize black holes as soon as possible so that an entity can failover to a different peer quickly. Likewise, it is sometimes necessary to detect black holes to recover lost resources.

This problem of detecting a dead IKE peer has been addressed by proposals that require sending periodic HELLO/ACK messages to prove liveliness. These schemes tend to be unidirectional (a HELLO only) or bidirectional (a HELLO/ACK pair). For the purpose of this document, the term "heartbeat" will refer to a unidirectional message to prove liveliness. Likewise, the term "keepalive" will refer to a bidirectional message.

The problem with current heartbeat and keepalive proposals is their reliance upon their messages to be sent at regular intervals. In the implementation, this translates into managing some timer to service these message intervals. Similarly, because rapid detection of the dead peer is often desired, these messages must be sent with some frequency, again translating into considerable overhead for message processing. In implementations and installations where managing large numbers of simultaneous IKE sessions is of concern, these regular heartbeats/keepalives prove to be infeasible.

To this end, a number of vendors have implemented their own approach to detect peer liveliness without needing to send messages at regular intervals. This informational document describes the current practice of those implementations. This scheme, called Dead Peer Detection (DPD), relies on IKE Notify messages to query the liveliness of an IKE peer.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

2. Document Roadmap

As mentioned above, there are already proposed solutions to the problem of detecting dead peers. Section 3 elaborates the rationale for using an IKE message exchange to query a peer's liveliness. Section 4 examines a keepalives-based approach as well as a heartbeats-based approach. Section 5 presents the DPD proposal fully, highlighting differences between DPD and the schemes presented in Section 4 and emphasizing scalability issues. Section 6 examines security issues surrounding replayed messages and false liveliness.

3. Rationale for Periodic Message Exchange for Proof of Liveliness

As the introduction mentioned, it is often necessary to detect that a peer is unreachable as soon as possible. IKE provides no way for this to occur -- aside from waiting until the rekey period, then attempting (and failing the rekey). This would result in a period of loss connectivity lasting the remainder of the lifetime of the security association (SA), and in most deployments, this is unacceptable. As such, a method is needed for checking up on a peer's state at will. Different methods have arisen, usually using an IKE Notify to query the peer's liveliness. These methods rely on either a bidirectional "keepalive" message exchange (a HELLO followed by an ACK), or a unidirectional "heartbeat" message exchange (a HELLO only). The next section considers both of these schemes.

4. Keepalives vs. Heartbeats

4.1. Keepalives:

Consider a keepalives scheme in which peer A and peer B require regular acknowledgements of each other's liveliness. The messages are exchanged by means of an authenticated notify payload. The two peers must agree upon the interval at which keepalives are sent, meaning that some negotiation is required during Phase 1. For any prompt failover to be possible, the keepalives must also be sent at rather frequent intervals -- around 10 seconds or so. In this hypothetical keepalives scenario, peers A and B agree to exchange keepalives every 10 seconds. Essentially, every 10 seconds, one peer must send a HELLO to the other. This HELLO serves as proof of liveliness for the sending entity. In turn, the other peer must acknowledge each keepalive HELLO. If the 10 seconds elapse, and one side has not received a HELLO, it will send the HELLO message itself, using the peer's ACK as proof of liveliness. Receipt of either a

HELLO or ACK causes an entity's keepalive timer to reset. Failure to receive an ACK in a certain period of time signals an error. A clarification is presented below:

Scenario 1:
Peer A's 10-second timer elapses first, and it sends a HELLO to B. B responds with an ACK.

```
Peer A:                                Peer B:
10 second timer fires;  ------>
wants to know that B is alive;
sends HELLO.
                                       Receives HELLO; acknowledges
                                       A's liveliness;
                         <------       resets keepalive timer, sends
                                       ACK.
Receives ACK as proof of
B's liveliness; resets timer.
```

Scenario 2:
Peer A's 10-second timer elapses first, and it sends a HELLO to B. B fails to respond. A can retransmit, in case its initial HELLO is lost. This situation describes how peer A detects its peer is dead.

```
Peer A:                                Peer B (dead):

10 second timer fires;  ------X
wants to know that B is
alive; sends HELLO.

Retransmission timer    ------X
expires; initial message
could have been lost in
transit; A increments
error counter and
sends another HELLO.
```

After some number of errors, A assumes B is dead; deletes SAs and possibly initiates failover.

An advantage of this scheme is that the party interested in the other peer's liveliness begins the message exchange. In Scenario 1, peer A is interested in peer B's liveliness, and peer A consequently sends

the HELLO. It is conceivable in such a scheme that peer B would never be interested in peer A's liveliness. In such a case, the onus would always lie on peer A to initiate the exchange.

4.2. Heartbeats:

By contrast, consider a proof-of-liveliness scheme involving unidirectional (unacknowledged) messages. An entity interested in its peer's liveliness would rely on the peer itself to send periodic messages demonstrating liveliness. In such a scheme, the message exchange might look like this:

Scenario 3: Peer A and Peer B are interested in each other's liveliness. Each peer depends on the other to send periodic HELLOs.

Peer A:		Peer B:
10 second timer fires; sends HELLO. Timer also signals expectation of B's HELLO.	------>	
		Receives HELLO as proof of A's liveliness.
	<------	10 second timer fires; sends HELLO.
Receives HELLO as proof of B's liveliness.		

Scenario 4:
Peer A fails to receive HELLO from B and marks the peer dead. This is how an entity detects its peer is dead.

Peer A:		Peer B (dead):
10 second timer fires; sends HELLO. Timer also signals expectation of B's HELLO.	------X	

Some time passes and A assumes B is dead.

The disadvantage of this scheme is the reliance upon the peer to demonstrate liveliness. To this end, peer B might never be interested in peer A's liveliness. Nonetheless, if A is interested B's liveliness, B must be aware of this, and maintain the necessary state information to send periodic HELLOs to A. The disadvantage of

such a scheme becomes clear in the remote-access scenario. Consider a VPN aggregator that terminates a large number of sessions (on the order of 50,000 peers or so). Each peer requires fairly rapid failover, therefore requiring the aggregator to send HELLO packets every 10 seconds or so. Such a scheme simply lacks scalability, as the aggregator must send 50,000 messages every few seconds.

In both of these schemes (keepalives and heartbeats), some negotiation of message interval must occur, so that each entity can know how often its peer expects a HELLO. This immediately adds a degree of complexity. Similarly, the need to send periodic messages (regardless of other IPSec/IKE activity), also increases computational overhead to the system.

5. DPD Protocol

DPD addresses the shortcomings of IKE keepalives- and heartbeats-schemes by introducing a more reasonable logic governing message exchange. Essentially, keepalives and heartbeats mandate exchange of HELLOs at regular intervals. By contrast, with DPD, each peer's DPD state is largely independent of the other's. A peer is free to request proof of liveliness when it needs it -- not at mandated intervals. This asynchronous property of DPD exchanges allows fewer messages to be sent, and this is how DPD achieves greater scalability.

As an elaboration, consider two DPD peers A and B. If there is ongoing valid IPSec traffic between the two, there is little need for proof of liveliness. The IPSec traffic itself serves as the proof of liveliness. If, on the other hand, a period of time lapses during which no packet exchange occurs, the liveliness of each peer is questionable. Knowledge of the peer's liveliness, however, is only urgently necessary if there is traffic to be sent. For example, if peer A has some IPSec packets to send after the period of idleness, it will need to know if peer B is still alive. At this point, peer A can initiate the DPD exchange.

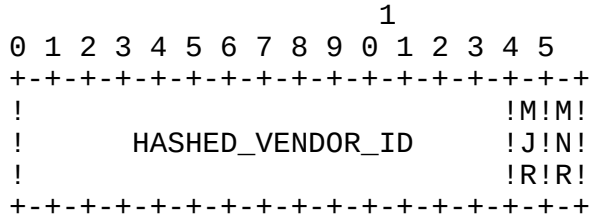
To this end, each peer may have different requirements for detecting proof of liveliness. Peer A, for example, may require rapid failover, whereas peer B's requirements for resource cleanup are less urgent. In DPD, each peer can define its own "worry metric" - an interval that defines the urgency of the DPD exchange. Continuing the example, peer A might define its DPD interval to be 10 seconds. Then, if peer A sends outbound IPSec traffic, but fails to receive any inbound traffic for 10 seconds, it can initiate a DPD exchange.

Peer B, on the other hand, defines its less urgent DPD interval to be 5 minutes. If the IPSec session is idle for 5 minutes, peer B can initiate a DPD exchange the next time it sends IPSec packets to A.

It is important to note that the decision about when to initiate a DPD exchange is implementation specific. An implementation might even define the DPD messages to be at regular intervals following idle periods. See section 5.5 for more implementation suggestions.

5.1. DPD Vendor ID

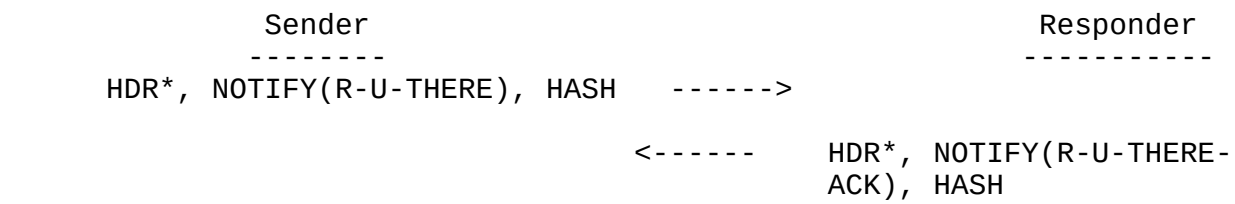
To demonstrate DPD capability, an entity must send the DPD vendor ID. Both peers of an IKE session MUST send the DPD vendor ID before DPD exchanges can begin. The format of the DPD Vendor ID is:



where HASHED_VENDOR_ID = {0xAF, 0xCA, 0xD7, 0x13, 0x68, 0xA1, 0xF1, 0xC9, 0x6B, 0x86, 0x96, 0xFC, 0x77, 0x57}, and MJR and MNR correspond to the current major and minor version of this protocol (1 and 0 respectively). An IKE peer MUST send the Vendor ID if it wishes to take part in DPD exchanges.

5.2. Message Exchanges

The DPD exchange is a bidirectional (HELLO/ACK) Notify message. The exchange is defined as:



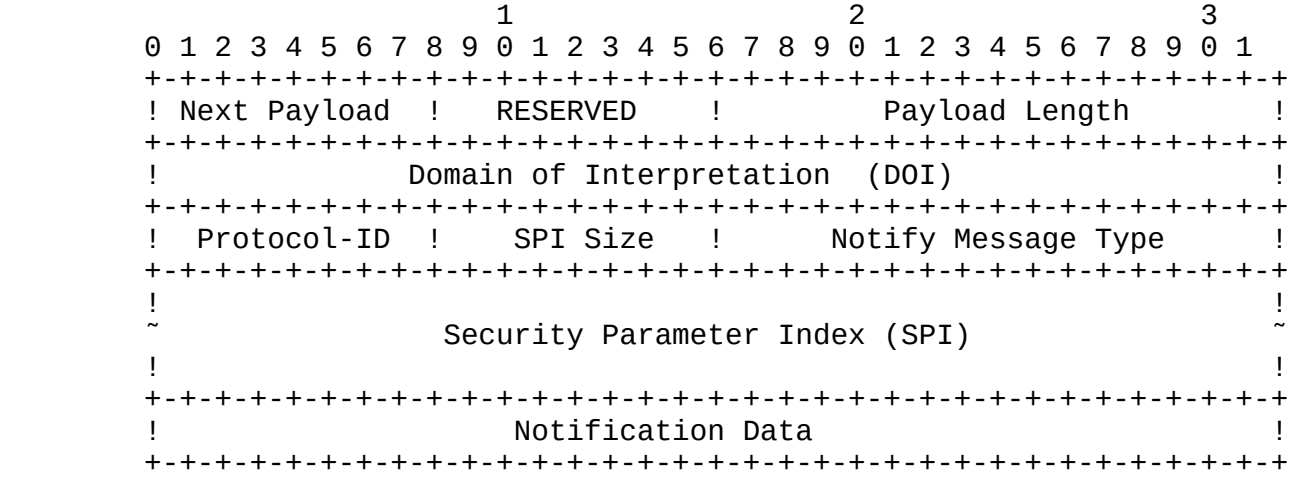
The R-U-THERE message corresponds to a "HELLO" and the R-U-THERE-ACK corresponds to an "ACK." Both messages are simply ISAKMP Notify payloads, and as such, this document defines these two new ISAKMP Notify message types:

Notify	Message Value
R-U-THERE	36136
R-U-THERE-ACK	36137

An entity that has sent the DPD Vendor ID MUST respond to an R-U-THERE query. Furthermore, an entity MUST reject unencrypted R-U-THERE and R-U-THERE-ACK messages.

5.3. NOTIFY(R-U-THERE/R-U-THERE-ACK) Message Format

When sent, the R-U-THERE message MUST take the following form:



As this message is an ISAKMP NOTIFY, the Next Payload, RESERVED, and Payload Length fields should be set accordingly. The remaining fields are set as:

- Domain of Interpretation (4 octets) - SHOULD be set to IPSEC-DOI.

- Protocol ID (1 octet) - MUST be set to the protocol ID for ISAKMP.

- SPI Size (1 octet) - SHOULD be set to sixteen (16), the length of two octet-sized ISAKMP cookies.

- Notify Message Type (2 octets) - MUST be set to R-U-THERE

- Security Parameter Index (16 octets) - SHOULD be set to the cookies of the Initiator and Responder of the IKE SA (in that order)

- Notification Data (4 octets) - MUST be set to the sequence number corresponding to this message

The format of the R-U-THERE-ACK message is the same, with the exception that the Notify Message Type MUST be set to R-U-THERE-ACK. Again, the Notification Data MUST be sent to the sequence number corresponding to the received R-U-THERE message.

5.4. Impetus for DPD Exchange

Again, rather than relying on some negotiated time interval to force the exchange of messages, DPD does not mandate the exchange of R-U-THERE messages at any time. Instead, an IKE peer SHOULD send an R-U-THERE query to its peer only if it is interested in the liveliness of this peer. To this end, if traffic is regularly exchanged between two peers, either peer SHOULD use this traffic as proof of liveliness, and both peers SHOULD NOT initiate a DPD exchange.

A peer MUST keep track of the state of a given DPD exchange. That is, once it has sent an R-U-THERE query, it expects an ACK in response within some implementation-defined period of time. An implementation SHOULD retransmit R-U-THERE queries when it fails to receive an ACK. After some number of retransmitted messages, an implementation SHOULD assume its peer to be unreachable and delete IPSec and IKE SAs to the peer.

5.5. Implementation Suggestion

Since the liveliness of a peer is only questionable when no traffic is exchanged, a viable implementation might begin by monitoring idleness. Along these lines, a peer’s liveliness is only important when there is outbound traffic to be sent. To this end, an implementation can initiate a DPD exchange (i.e., send an R-U-THERE message) when there has been some period of idleness, followed by the desire to send outbound traffic. Likewise, an entity can initiate a DPD exchange if it has sent outbound IPSec traffic, but not received any inbound IPSec packets in response. A complete DPD exchange (i.e., transmission of R-U-THERE and receipt of corresponding R-U-THERE-ACK) will serve as proof of liveliness until the next idle period.

Again, since DPD does not mandate any interval, this "idle period" (or "worry metric") is left as an implementation decision. It is not a negotiated value.

5.6. Comparisons

The performance benefit that DPD offers over traditional keepalives- and heartbeats-schemes comes from the fact that regular messages do not need to be sent. Returning to the examples presented in section 4.1, a keepalive implementation such as the one presented would require one timer to signal when to send a HELLO message and another timer to "timeout" the ACK from the peer (this could also be the retransmit timer). Similarly, a heartbeats scheme such as the one presented in section 4.2 would need to keep one timer to signal when to send a HELLO, as well as another timer to signal the expectation of a HELLO from the peer. By contrast a DPD scheme needs to keep a timestamp to keep track of the last received traffic from the peer (thus marking beginning of the "idle period"). Once a DPD R-U-THERE message has been sent, an implementation need only maintain a timer to signal retransmission. Thus, the need to maintain active timer state is reduced, resulting in a scalability improvement (assuming maintaining a timestamp is less costly than an active timer). Furthermore, since a DPD exchange only occurs if an entity has not received traffic recently from its peer, the number of IKE messages to be sent and processed is also reduced. As a consequence, the scalability of DPD is much better than keepalives and heartbeats.

DPD maintains the HELLO/ACK model presented by keepalives, as it follows that an exchange is initiated only by an entity interested in the liveliness of its peer.

6. Resistance to Replay Attack and False Proof of Liveliness

6.1. Sequence Number in DPD Messages

To guard against message replay attacks and false proof of liveliness, a 32-bit sequence number MUST be presented with each R-U-THERE message. A responder to an R-U-THERE message MUST send an R-U-THERE-ACK with the same sequence number. Upon receipt of the R-U-THERE-ACK message, the initial sender SHOULD check the validity of the sequence number. The initial sender SHOULD reject the R-U-THERE-ACK if the sequence number fails to match the one sent with the R-U-THERE message.

Additionally, both the receiver of the R-U-THERE and the R-U-THERE-ACK message SHOULD check the validity of the Initiator and Responder cookies presented in the SPI field of the payload.

6.2. Selection and Maintenance of Sequence Numbers

As both DPD peers can initiate a DPD exchange (i.e., both peers can send R-U-THERE messages), each peer MUST maintain its own sequence number for R-U-THERE messages. The first R-U-THERE message sent in a session MUST be a randomly chosen number. To prevent rolling past overflowing the 32-bit boundary, the high-bit of the sequence number initially SHOULD be set to zero. Subsequent R-U-THERE messages MUST increment the sequence number by one. Sequence numbers MAY reset at the expiry of the IKE SA, moving to a newly chosen random number. Each entity SHOULD also maintain its peer's R-U-THERE sequence number, and an entity SHOULD reject the R-U-THERE message if it fails to match the expected sequence number.

Implementations MAY maintain a window of acceptable sequence numbers, but this specification makes no assumptions about how this is done. Again, it is an implementation specific detail.

7. Security Considerations

As the previous section highlighted, DPD uses sequence numbers to ensure liveliness. This section describes the advantages of using sequence numbers over random nonces to ensure liveliness.

While sequence numbers do require entities to keep per-peer state, they also provide an added method of protection in certain replay attacks. Consider a case where peer A sends peer B a valid DPD R-U-THERE message. An attacker C can intercept this message and flood B with multiple copies of the messages. B will have to decrypt and process each packet (regardless of whether sequence numbers or nonces are in use). With sequence numbers B can detect that the packets are replayed: the sequence numbers in these replayed packets will not match the incremented sequence number that B expects to receive from A. This prevents B from needing to build, encrypt, and send ACKs. By contrast, if the DPD protocol used nonces, it would provide no way for B to detect that the messages are replayed (unless B maintained a list of recently received nonces).

Another benefit of sequence numbers is that it adds an extra assurance of the peer's liveliness. As long as a receiver verifies the validity of a DPD R-U-THERE message (by verifying its incremented sequence number), then the receiver can be assured of the peer's liveliness by the very fact that the sender initiated the query. Nonces, by contrast, cannot provide this assurance.

8. IANA Considerations

There is no IANA action required for this document. DPD uses notify numbers from the private range.

9. References

9.1. Normative Reference

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

9.2. Informative References

[2] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[3] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

10. Editors' Addresses

Geoffrey Huang
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134

Phone: (408) 525-5354
EMail: ghuang@cisco.com

Stephane Beaulieu
Cisco Systems, Inc.
2000 Innovation Drive
Kanata, ON
Canada, K2K 3E8

Phone: (613) 254-3678
EMail: stephane@cisco.com

Dany Rochefort
Cisco Systems, Inc.
124 Grove Street, Suite 205
Franklin, MA 02038

Phone: (508) 553-8644
EMail: danyr@cisco.com

11. Full Copyright Statement

Intellectual Property

Acknowledgement

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