Erik Nordmark
Sun Microsystems

Strong Identity Multihoming using 128 bit Identifiers (SIM/CBID128)

<draft-nordmark-multi6-sim-01.txt>

Status of this Memo

Abstract

This document contains a rough outline of a potential solution to IPv6 multihoming in order to stimulate discussion.

This proposed solution uses 126 bit identifiers which are hashes of public keys (know as Cryptographically-based Identifiers) which are created in an autonomous fashion by every host. When there is a need to verify whether a new locator should be used with an identifier than a public-key based challenge-response is used.

The proposal allows locator rewriting by (border) routers, and ensures that the upper layer protocols can operate unmodified in a multihomed setting using the stable identifiers.

Contents

1. INTRODUCTION

The goal of the IPv6 multihoming work is to allow a site to take advantage of multiple attachments to the global Internet without having a specific entry for the site visible in the global routing table. Specifically, a solution should allow users to use multiple attachments in parallel, or to switch between these attachment points dynamically in the case of failures, without an impact on the upper layer protocols.

This proposed solution uses crypto-based identifiers [CBID] properties to perform enough validation to prevent redirection attacks.

The goals for this proposed solution is to:

- Have no impact on upper layer protocols in general and on transport protocols in particular.
- Address the security threats in [M6SEC].
- Allow routers rewriting the (source) locators as a means of quickly detecting which locator is likely to work for return traffic.
- Minimal per-packet overhead.
- No extra roundtrip for setup through optional piggybacking.
- Take advantage of multiple locators for load spreading.

1.1. Non-Goals

The assumption is that the problem we are trying to solve is site multihoming, with the ability to have the set of site locator prefixes change over time due to site renumbering. Further, we assume that such changes to the set of locator prefixes can be relatively slow and managed; slow enough to allow updates to the DNS to propagate. This proposal does not attempt to solve, perhaps related, problems such as host multihoming or host mobility.

This proposal introduces an IP layer identifier, but it does not make this identifier a first class object. In particular, there is no method for taking a identifier and using it to lookup other information (FQDN, set of locators, etc) about the identified entity. Even with this limitation the introduction of the identifier is quite

useful in identifying the a host. See discussion in the section on future work how it might be possible to add a lookup function over time.

1.2. Assumptions

The main technical assumptions this proposal makes is that using public key signatures during the more uncommon operations would provide acceptable performance. The proposal doesn't require such operations during normal communication; only when a locator changes for a host will it need to be verified before return traffic will be sent to that locator, or when two hosts claim to use the same identifier.

Another assumption is that where DNS is already used (normally at the initiating end of communication) it is acceptable to lookup the identifier of the peer in the DNS in addition to the current AAAA lookup (of the addresses/locators).

2. TERMINOLOGY

upper layer protocol (ULP)
- a protocol layer immediately above IP. Examples are transport protocols such as TCP and UDP, control protocols such as ICMP, routing protocols such as OSPF, and internet or lower-layer protocols being "tunneled" over (i.e., encapsulated in) IP such as IPX, AppleTalk, or IP itself.

interface - a node's attachment to a link.

address - an IP layer name that contains both topological significance and acts as a unique identifier for an interface. 128 bits.

locator - an IP layer topological name for an interface or a set of interfaces. 128 bits. The locators are carried in the IP address fields as the packets traverse the network.

identifier - an IP layer identifier for an IP layer endpoint (stack name in [NSRG]). The transport endpoint is a function of the transport protocol and would typically include the IP identifier plus a port number.

Application identifier (AID)
- The 128 bit quantity used by upper layer protocols for identifying a peer. In this proposal the AID=ID i.e. an IP layer identifier. This is used for pseudo-header checksum computation and connection identification in the ULP.

address field
- the source and destination address fields in the IPv6 header. As IPv6 is currently specified this fields carry "addresses". If identifiers and locators are separated these fields will contain locators.

FQDN - Fully Qualified Domain Name

2.1. Notational Conventions

A, B, and C are hosts. X is a potentially malicious host.

FQDN(A) is the domain name for A.

Ls(A) is the locator set for A, which consists of L1(A), L2(A), ... Ln(A).

ID(A) is an application ID for A. ID(A) is a 128 bit number consisting of two fixed bits (e.g., 10) followed by 126 bits of a truncated SHA1 hash of a public key that the host has generated.

CT(A) is a 64 bit "context tag" allocated by A and used when B sends packets to A. The packets contain the low-order 32 bits of the tag, named CT32(A). The full tag is used for DoS-attack prevention during the PK challenge/response.

3. PROTOCOL OVERVIEW

In order to prevent redirection attacks this protocol relies on the ability to verify (using public key crypto as in [CBID]) that the entity requesting redirection indeed holds the private key where the hash of the corresponding public key hashes to the ID itself.

The initiator of communication, where it uses DNS today to lookup FQDN->addresses, will instead lookup both FQDN->identifier (probably using some new DNS RR type) and FQDN->locator set (using AAAA resource records). The existence of the identifier RR for the name is an indication that the node supports multihoming.

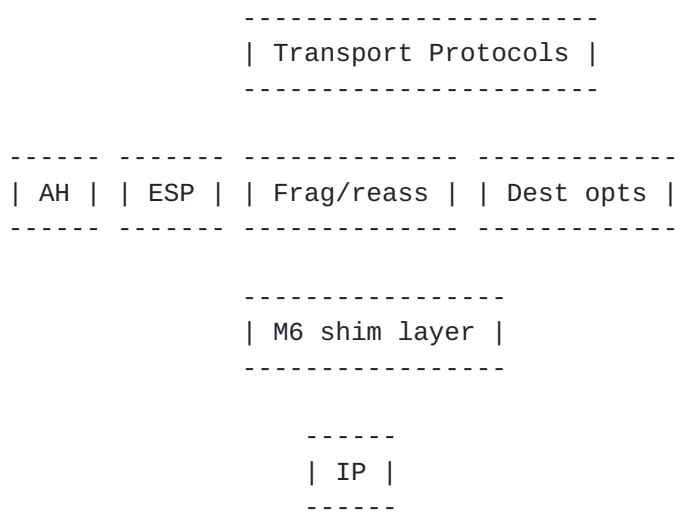


Figure 1: Protocol stack

The proposal uses an M6 shim layer between IP and the ULPs as shown in figure 1, in order to provide ULP independence. The M6 layer corresponds to an extension header which is the minimum 8 octets for data packets but larger for the messages used to establish the state at the two ends and perform the public key challenge response exchanges. In addition to carrying data packets, the M6 protocol has three message types to perform a 3-way handshake to establish state at the two ends, two message types to perform a challenge request/response exchange when a new locator is introduced, and finally a single message type to signal that state has been lost.

Layering AH and ESP above the M6 shim means that IPsec can be made to be unaware of locator changes the same way that transport protocols can be unaware. Thus the IPsec security associations remain stable even though the locators are changing. Layering the fragmentation header above the M6 shim makes reassembly robust in the case that there is broken multi-path routing which results in using different

paths, hence potentially different source locators, for different fragments.

The proposal uses router rewriting of (source) locators as one way to determine which is the preferred (or only working) locator to use for return traffic. But not all packets can have their locators rewritten. Thus a simple mechanism is needed to indicate to the routers on the path whether or not it is ok to rewrite the locators in the packet. Conceptually this is a single bit in the IPv6 header (we call it the "rewrite ok" bit) but there is no spare bit available. Instead we allocate two next header values for M6; one which means "rewrite ok" and one which means the rewrite should not be performed by routers.

Applications and upper layer protocols use IDs which the M6 layer will map to/from different locators. The M6 layer maintains state, called host-pair context, in order to perform this mapping. The mapping is performed consistently at the sender and the receiver, thus from the perspective of the upper layer protocols packets appear to be sent using IDs from end to end, even though the packets travel through the network containing locators in the IP address fields, and even though those locators might be rewritten in flight.

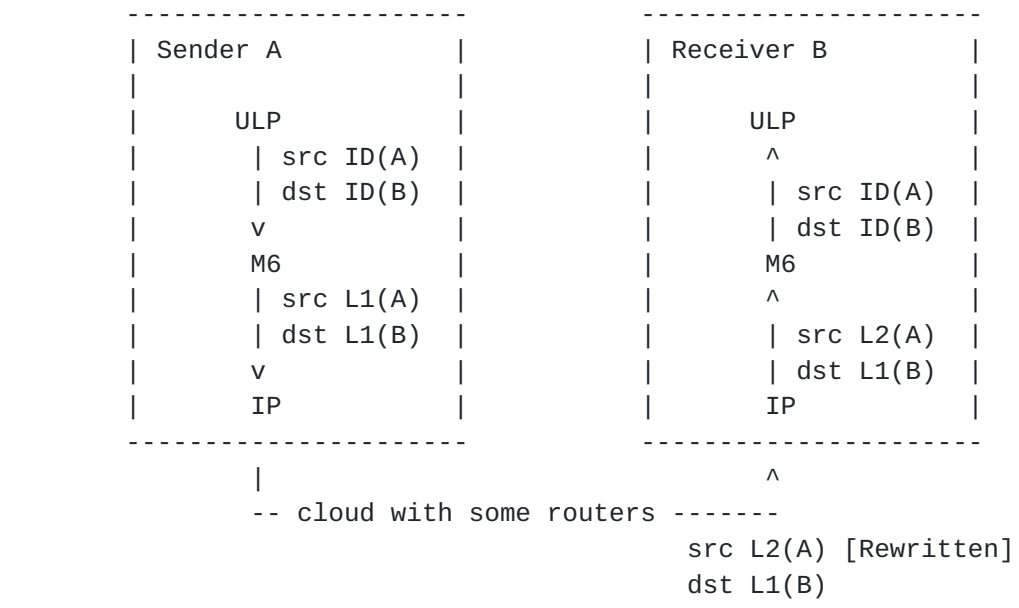


Figure 2: Mapping with router rewriting of locators.

The result of this consistent mapping is that there is no impact on the ULPs. In particular, there is no impact on pseudo-header checksums and connection identification.

Conceptually one could view this approach as if both IDs and locators being present in every packet, but with a header compression

mechanism applied that removes the need for the IDs once the state has been established. As we will see below the context tag will be used akin to a "compression tag" i.e., to indicate the correct context to use for decompression.

The use of the context tag allows the receiver to find the correct context without relying on the locators in the packet.

3.1. Host-Pair Context

The host-pair context is established on the initiator of communication based on information learned from the DNS (either by starting with a FQDN or with an IP address -> FQDN lookup). The responder will establish some initial state using the context creation 3-way handshake. Both hosts later update the peer locators based on the source locator in received packets after having verified the new locator using a challenge exchange.

The context state contains the following information:

- the peer ID; ID(peer)
- the local ID; ID(local)
- the set of peer locators; Ls(peer)
- for each peer locator, a bit whether it has been verified for return traffic using a PK challenge.
- the preferred peer locator - used as destination; Lp(peer)
- the set of local locators; Ls(local)
- the preferred local locator - used as source; Lp(local)
- the context tag used to transmit packets; CT(local)
- the context to expect in receive packets; CT(peer)
- State about peer locators that are in the process of being verified in using challenge/response.

This state is accessed differently in the transmit and receive paths. In the transmit path when the ULP passes down a packet the key to the context state is the tuple <ID(local), ID(peer)>; this key must identify at most one state record. In the receive path it is the

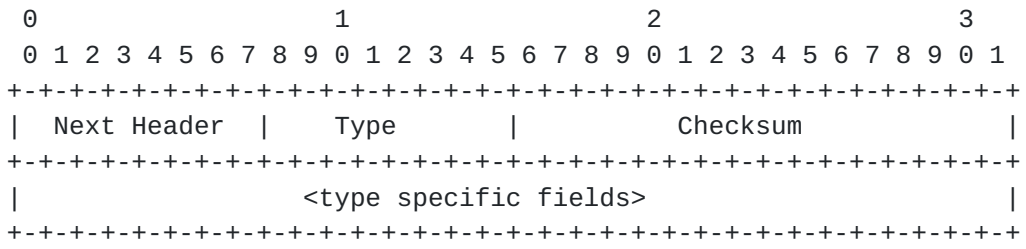
CT32(local) that is used to identify at most one state record.

At the initiating end it is possible that the DNS lookup of different FQDNs return the same ID and different locator sets. This is discussed in section 4.1 and 12.1.

The receiving end allocates the context tags thus it can trivially ensure that the CT32(local) is unique.

3.2. Message Formats

The base M6 header is as follows:



M6 Fields:

Next Header

8-bit selector. Identifies the type of header immediately following the M6 header. Uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].

Type

8-bit field. The type of M6 message. The M6 header carries 7 different message types:

- Data message; to be passed to the ULP after replacing the locators with the identifiers.
- Context request message; first message of the 3-way context establishment. An ULP packet can be piggybacked on this message.
- Context response message; second message of the 3-way context establishment. An ULP packet can be piggybacked on this message.
- Context confirm message; third message of the 3-way context establishment. An ULP packet can

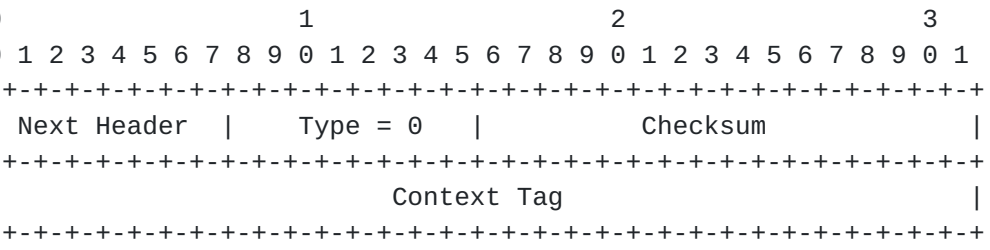
be piggybacked on this message.

- o Challenge request message; first message of the 2-way challenge.
- o Challenge response message; second message of the 2-way challenge.
- o Unknown context message; error which is sent when no state for context tag.

Checksum The Internet checksum applied to the IPv6 address fields and the M6 header. When computing the checksum the checksum field is set to zero.

Future versions of this protocol may define message types. Receivers MUST silently ignore? Reject? [TBD] any message type they do not recognize.

The M6 data message is as follows:



M6 Fields:

Context Tag
32-bit field. Identifies the context at the receiver.

This drafts doesn't contain actual message layout for the other M6 message types. However, the content of these messages is specified below.

The Context request message contains:

- Sender Nonce
- Sender ID
- Receiver ID
- Sender context tag (64 bits)

The Context response message contains:

- Receiver Nonce (copied from Sender Nonce in request)

- Context state consisting of: the two IDs, the two context tags, and the initial locators

- A timestamp or nonce (for sender's benefit)

- A hash over the context state and timestamp (to prevent modification)

The Context confirm message contains:

- The context state, timestamp/nonce, and hash copied from the context response.

The Challenge request message contains:

- Sender generated nonce/timestamp

- The two IDs

- The 32-bit context tag from the received data message

- The source locator from the received data message

The Challenge response message contains:

- The nonce/timestamp from the challenge request

- The 32-bit context tag (from the challenge request)

- The above locator

- A hash value (H2) which proves that the sender knows the full 64 bits of both the context tags. This allows the receiver of the response to avoid verifying the PK signature generated by a host which can't be the original peer.

- The public key of the sender.

- The public key signature of all of the above fields.

The Unknown context message contains:

- The 32 bit context tag in the triggering data message.

4. PROTOCOL WALKTHROUGH

4.1. Initial Context Establishment

Here is the sequence of events when A starts talking to B:

1. A looks up FQDN(B) in the DNS. The lookup is both for the new ID records and the AAAA records. If no ID record then the peer is a standard IPv6 host and the set of AAAA records is returned to the application as today. If an ID record is found then only that record is returned to the application. The ID and AAAA content is passed to the M6 layer on the host as ID(B) and Ls(B). To make sure that the lookup from ID(B) returns a single state record in the M6 layer there has to be a check if there is already a record for that ID with a different Ls. One could envision sending a PK challenge to the locators to resolve such a conflict. See section 12.1 for more discussion.

2. The ULP creates "connection" state between ID(A) and ID(B) and sends the first packet. ID(A) was picked using regular source address selection mechanisms.

3. The leading two bits of the destination address (10) makes the transmit side go through M6 processing. The M6 layer matches on ID(B) and finds the proto-context state containing Ls(B) created in step 1. The fact that the context state isn't complete triggers context creation.

4. A picks a random Nonce to include in the Context Request message. A picks pseudo-random 64-bit context tags (CT(A)) and checks for duplicates against existing context state records until it finds a unique one. The nonce and CT(A) are saved in the context state. Then A sends the context request message to one of B's locators. The data packet (TCP SYN or whatever) can be piggybacked on the context request message. The "rewrite ok" bit is set in the header.

5. The M6 layer on B receives context request message. It does not find any context state for the pair of IDs. It forms the 64-bit pseudo-random CT(B) and checks it against duplicates for existing contexts. (It can't check against other contexts in progress of being created since it doesn't create any state until the context confirm is received. Hence the duplicate check is redone on the context confirm.) It forms a keyed hash (using K(B) which is never shares with anybody) of the context state and a timestamp. Then puts the state, timestamp and hash

in the context response message.

If there was a piggybacked packet on the context request message then B passes packet to ULP after putting the identifiers in the IP address fields. The ULP sees a packet identified by ID(A), ID(B).

6. The M6 layer on A receives the context response message. It looks up the state using CT32(A). It verifies that the stored nonce matches the echoed nonce in the message. It records the source address field in as Lp(peer). It sends back a context confirm message containing what was in the response message.

   If there was a piggybacked packet on the context response message then A passes packet to ULP after putting the identifiers in the IP address fields. The ULP sees a packet identified by ID(A), ID(B).

7. The M6 layer on B receives context confirm message. It looks up context state using CT32(B). If state is found and the identifiers are different than in the context confirm this must have been a duplicate caused by concurrent context creations in progress. Requires restarting by responding with a new context response with a different CT(B).

   B verifies that the timestamp is recent and then verifies that the hash over the state is correct. Then it can create the context state.

   If there was a piggybacked packet on the context confirm message then B passes packet to ULP after putting the identifiers in the IP address fields. The ULP sees a packet identified by ID(A), ID(B).

4.2. Locator Change

This is the sequence of events when B receives a packet with a previously unknown source locator for A.

1. Host B receives a data message. Finds the context using the CT32(B) that is in the message. B passes the packet to the ULP after replacing the locators with the IDs (whether or not the source locator is known).

   If the source locator is in Ls(peer) and already verified then the preferred return locator (Lp(peer)) is updated to use it for

return packets.

If the source locator is previously unknown then it is added to the context state as a Ls(peer) awaiting verification and a Challenge Request packet is generated. The challenge request includes a nonce generated by B, CT32(B) (that was received in the packet from the unknown locator), the identifiers and the previously unknown peer locator.

2. Host A receives the challenge request packet. Verifies that it has state for those identifiers with the CT32(peer) it received on the request.

   It computes the hash H2 to show to B that it knows both full 64 bit context tags as H2 = SHA1(nonce from request, CT(A), CT(B), ID(A), ID(B))

   It includes its public key (the one whose hash is ID(A)) and signs the whole challenge response using its private key.

3. Host B receives the challenge response packet. It finds the context state using CT32(B). It verifies the nonce against what it used for sending the challenge request. It verifies H2. (Only devices on the path between A and B during the context establishment knows CT(A) and CT(B), thus this check limits DoS attacks based on forcing PK signature verification to attackers on the path.) Then it verifies that the hash of the public key equals ID(A), and finally the public key signature using that public key.

4.3. Handling Locator Failures

The M6 layer is responsible for retransmitting context request messages using different locators until a contest response is received.

After context setup the sender can use retransmit hints from the ULP to get the M6 layer to try a different verified locator.

If one outbound path from the site fails and the border routers rewrite source locators then the peer in another site will see packets with the working source locators. Once that locator has been verified, the return path will switch to use the working locator. As long as both ends are transmitting packets this will relatively quickly switch to working locators except when both hosts experience a failing locator at the same time.

Without locator rewriting one would need to add some notification e.g., by defining a new bit in the router advertisement prefixes (IMHO this is semantically different than the preferred vs. deprecated stuff), but we also need some mechanism to carry this info from the border routers to the routers on each subnet.

4.4. Locator Set Changes

Should the set of locators change after the context has been established the ability to learn and verify new peer locators will handle this fine.

The DNS only needs to be updated with new locators in order to allow new communication to be established.

When a host sees (based on router advertisements [DISCOVERY]) that one of its locators has become deprecated and it has additional locators that are still preferred, it is recommended that the host start using the preferred locator(s) with the contexts that have already been established. This ensures that should the deprecated locator become invalid the peers have already verified other locator(s) for the host.

4.5. Preventing Premeditated Redirection Attacks

The threats document [M6SEC] talks of premeditated redirection attacks that is where an attacker claims to be a host before the real host appears.

This proposal is potentially subject to this threat because for performance reasons there is no public-key challenge when the context state is initially established.

The following sequence shows how such a redirection attack is detected when X pretends to be A:

1. Host X with locator L1(X) sends a content request message to B. In the message it claims to have ID(A) and includes CT(X).

2. The context response and context confirm messages are exchanged resulting on B selecting CT(B) for communicating with X (which B believes has identifier ID(A)).

3. X and B happily communicate without B performing any higher-

level, such as IKE/IPsec, identity check on its peer.

4. Host A tries to communicate with B. It sends a context request message to B where the message claims to have ID(A) and includes CT(A).

5. Host B receives the context request and discovers it already has context state for ID(A). B doesn't do anything different than if there was no context state - the difference in processing happens when the context confirm is received - except that any piggybacked ULP packet is not passed to the ULP. Thus, as in section 4.1, a context tag is selected and the context reply is sent, which makes A send back a context confirm.

6. Host B receives the context confirm and verifies it the same way as in section 4.1. Then it looks if there is already a context for ID(A) and finds the context which contains CT(X) and L1(X).

   The existence of this "old" context could be due to multiple reasons:

   - The peer lost state while B retained the context state. In this case one would expect that the old context has not been used to receive packets for some time. (Having a protocol constant denoting the minimum time after sending a packet that state can be lost and later recreated would be helpful here.) In this case it would also be common that the source address of the packet would fall in the locator set for the old context. But it isn't impossible that a peer state loss and using a different locator happens at the same time.

   - The old host was performing a premediated redirection attack.

   - The new host is attempting a redirection attack.

   In all cases the approach consists of sending a challenge to both the "new" A and the "old" A. But depending on the time since packets where last received from the "old" A the order can be different. The first peer locator which responds with a valid challenge response will "win" and the other context state will be deleted.

TBD: The above has DoS concerns in terms of verifying the challenge response. Having both ends remove the context state at about the same time would be beneficial since it would reduce the frequency of this happening in the absence of attacks, thus it would be more realistic to apply resource limits for this type of challenges.

5. HANDLING STATE LOSS

The protocol needs to handle two forms of state loss:

- a peer loosing all state,

- the M6 layer garbage collecting state too early due to not being aware of what all ULPs do.

The first case is the already existing case of a host crashing and "rebooting" and as a result loosing transport and application state. In this case there are some added complications from the M6 layer since a peer will continue to send packets assuming the context still exists and due to the loss of state on the receiver it isn't even able to pass the correct packet up to the ULP (e.g., to be able to get TCP to generate a reset packet) since it doesn't know what IDs to use when replacing the locators.

The second case is a bit more subtle. Ideally an implementation shouldn't discard the context state when there is some ULP that still depends on this state. While this might be possible for some implementations with a fixed set of applications, it doesn't appear to be possible for implementations which provide the socket API; there can be things like user-level "connections" on top of UDP as well as application layer "session" above TCP which retain the identifiers from some previous communication and expect to use those identifiers at a later date. But the M6 layer has no ability to be aware of this.

Thus an implementation shouldn't discard context state when it knows it has ULP connection state (which can be checked in e.g., Unix for TCP), or when there is active communication (UDP packets being sent to ID(A) recently), but when there is an infrequently communicating user-level "connection" over UDP or "session" over TCP the context state might be garbage collected even though it shouldn't.

Independently whether the state loss was for the whole host or for just the M6 layer things can be recovered when the loss is detected as a result of receiving a packet from the peer. Should B receive a packet from some locator without being able to find any context state for the CT32(B) that is in the packet, it will send back an Unknown context message to the source. The unknown context message includes the receive tag (and it can't receive much other useful information since B has no state). The Unknown context message is sent without setting the "rewrite ok" bit since locator rewriting would make it harder for A to perform sanity checks on this error message.

When A receives the Unknown context message it verifies that the

CT32(B) value is used by a context with the locators that are in the IP address fields, and that the source address field is Lp(peer) for the context (the preferred peer locator - to which an earlier data message would have been sent). Once this verification has been done A needs to restart the 3-way context establishment; A can reuse CT(A) for this if it so desires.

As a result of this error handling mechanism an attacker on the path between A and B can send frequent Unknown context messages (but an off-path attacker cannot since it wouldn't know CT32(B)). Since state loss is expected to be infrequent it is reasonable to rate limit the handling of Unknown context messages per context to one per minute.

In the case of M6 layer state loss (due to too early garbage collection) the above provides recovery when the peer transmits. But there is also the possibility that the lack of state will be detected when the ULP is passing down a packet to transmit. In this case M6 would see a packet which needs state (because the leading to bits of the address field is 10) but there is no state. Unless we invent a method (see section 14) to lookup identifiers there is nothing that can be done in that case (except perhaps wait for a while in case the peer will send a packet triggering recovery using the Unknown context message). Thus it is quite important that the context state is not discarded prematurely.

TBD: If we decide to explore the approach in this proposal further, would be it useful to add APIs that allow applications to advise when it isn't and when it is ok to discard the context state for a particular id?

6. APPLICATION USAGE OF IDENTIFIERS

In this proposal the upper layer protocols will operate on the identifiers. As long as the M6 layer doesn't garbage collect context state too early then those identifiers can be used by other applications on the same host; they will only become useless if none of the locators stored in Ls(peer) stop working for instance due to renumbering of the peer.

But the identifier is rather useless for referrals; should B pass ID(A) to C there is no mechanism for C to find A's locators. It is only if A somehow contacts C that it can tell that it is indeed A.

In order to be able to use an identifier for establishing communication a host would also need a set of locators where at least one locator is still current and working.

Thus with this approach, unless we defer until we have explored the future work in section 14, it would make sense to perform referrals by either passing FQDNs or by passing an ID plus the list of locators know by the referring host. One could envision a getpeerlocators() API which, given an ID, would extract the locator set from the context on the host itself.

Applications which use to map the peer's IP address to a domain name today perform a reverse lookup in the DNS (e.g., using the getnameinfo() API). Since the flat identifier space can't be effectively added to the ip6.arpa tree, the getnameinfo() can instead extract the locators from the local context state. For example, this operation by B on ID(A) would find Ls(A) in the context and do a reverse lookup (presumably only on one of them); L1(A). This would be likely to return FQDN(A). Applications which today perform a forward+reverse lookup would need then lookup FQDN(A) - to find the identifier. Note that the above doesn't show that A actually is the "owner" of ID(A).

One could envision providing an optional stronger verification to the applications using the CBID properties; a verification of ID(A) would extract Ls(A) and then send a challenge request to the locator. That proof of ownership of ID(A) coupled with the locator->FQDN->ID DNS looks gives stronger assurance of the identity of the peer IP layer than today. But as pointed out in [M6SEC], applications which require strong identity authentication typically also want integrity with or without confidentiality for the communication. Thus identity checks unrelated to payload cryptography might be unnecessary as a specific service to the application layer.

7. COMPATIBILITY WITH STANDARD IPv6

A host can easily implement M6 in a way that interoperates with current IPv6 as follows.

When the DNS lookup routines do not find an ID record for the peer they will return the AAAA resource record set to the application; those would be the IPv6 addresses. When the ULP passes down these addresses the M6 layer will see that the leading two bits are not

(10) thus it will pass things through.

It is an open issue whether or not it makes sense to check in an implementation that the content of the returned ID records start with binary 10 and that the content of the returned AAAA records not start with binary 10.

8. CHECKSUM ISSUES

The IPv6 header does not have a checksum field; the IPv6 address fields are assumed to be protected by the ULP pseudo-header checksum. The general approach of an M6 shim which replaces locators with identifiers (where only the identifiers are covered by the ULP checksum) raises the potential issue of robustly handling bit errors in the address fields.

This proposal as currently written has an M6 checksum field to cover the locators and the M6 header, but it isn't obvious that such a checksum is strictly required for the data packets.

If there is an undetected bit error in the source address field, this would look like an unverified source locator to the receiver. Thus the packet would (after replacing locators with identifiers based on the context) be passed to the ULP and a challenge response exchange be triggered. In the case of a bit error in the locator this challenge isn't likely to receive a response; and if there is a response by someone it wouldn't be from the actual peer thus the verification would fail. Thus such an undetected bit error is harmless.

Except for the obscure case when Ls(A) contains multiple verified locators, one or more of those are not working, and the bit error causes L1(A) to be replaced by L2(A). That would make the return traffic go to L2(A), but that might be a non-functioning locator. In this case the mistake will be corrected when a subsequent packet is received from A.

An undetected bit error in the destination address field is also harmless; it might cause misdelivery of the packet to a host which has no context but the reception of the resulting Unknown context error message will show that it arrives from the incorrect locator thus it will be ignored.

An undetected bit error in the M6 next header field isn't any different than for the base IPv6 next header field.

An undetected bit error in the context tag in a data message could have two possible effects: not finding any context state, or finding the incorrect context state. In the first case the Unknown context error message would be dropped by the peer since the tag doesn't match the tag that was originally sent. In the second case this will result in a packet with incorrect identifiers being delivered to the ULP which most like will drop it due to ULP checksums not matching.

NOTE: If one thinks that new peer locators could be used for return traffic without any challenge/response (e.g., relying on the hard to guess context tags), then clearly a checksum must protect against undetected bit errors causing the return packets to be sent to a bogus locator.

9. IMPLICATIONS FOR PACKET FILTERING

Ingress filtering should be replaced by locator rewrite when the "rewrite ok" bit is set.

Locator rewriting (when the bit is set) can be applied at places where ingress filtering isn't currently performed (e.g., due to multihoming issues).

Firewall filtering potentially require modifications to be aware of M6. All the packets contain locator thus a firewall would need to be aware of the context state to let the correct packets through. Such firewalls could optionally perform their own verification by issuing challenge request messages (the protocol doesn't explicitly allow for this; they would have to pretend being the actual endpoint sending the challenge).

10. IPSEC INTERACTIONS

As specified all of ESP, AH, and key management is layered above the M6 layer. Thus they benefit from the stable identifiers provided above the M6 layer. This means the IPsec security associations are unaffected by switching locators.

The alternative would be to layer M6 above IPsec, but that doesn't seem to provide any benefits. Since we want to allow routers performing locator rewriting it wouldn't possible to take advantage of for instance AH to protect the integrity of the IP headers.

11. SECURITY CONSIDERATIONS

This analysis is far from complete. Early analysis indicates this addresses the issues in [M6SEC].

Just as in today's Internet hosts on the path can inject bogus packets; in this proposal they need to extract the context tags from the packets in order to do this which wouldn't be hard. Packet injection from off-path places becomes harder since it requires guessing the 32 bit context tag.

Hosts on the path can also launch PK signature verification DoS attacks against either end since they can observe the context tags from the establishment and therefor compute the H2 hash in the challenge response packet. This would force the endpoint to run the signature verification algorithm which is expensive. If we don't expect the locator sets to be very dynamic one could restrict the rate at which such verification takes place, at least after the first few locators have been verified for a peer.

The initial setup of a host-pair context does not perform any verification using public key crypto, but this does not seem to make the result less secure than today's Internet. Applications which do not perform access control based on it's notion of the peer wouldn't care about the strength of the peer's identifier. And applications which perform strict access control hopefully do this using strong crypto (IPsec, TLS, etc) today and would continue to do so. That leaves applications which perform the questionable practise of merely verifying the forward plus reverse lookups in the DNS and then using the IP address (or resulting FQDN) for access control discussions. As discussed in section 6 the application's lookup of locator->FQDN-

>ID and verifying that the identifier matches provides about the same strength. [TBD are we really sure?]

The CBIDs are only statistically unique. But 126 bit identifiers seems large enough to make collisions unlikely enough to keep the protocol simple. (If not one could envision complications like making the protocol capable of detecting collisions by storing the public key in the context state and seeing if a host claims to use the same ID but has a different public key.) While at about 8*10^18 hosts in the Internet there is approximately a 50% probability that there exists 2 hosts with the same 126-bit identifier this has no effect on the protocol per see. It is not until a single host has that order of magnitude of context state records that it could get confused due to collisions.

12. OPEN ISSUES

Some protocol complexity is added by not performing a mutual public-key challenge immediately when a context is created. At the expensive of a performance hit one could simplify the protocol to always to these challenges.

Is it possible to facilitate transition to M6 using some "M6 proxy" at site boundaries until all important hosts in a site have been upgraded to support M6? Would would be the properties of such a proxy? Would it place any additional requirements on the protocol itself?

One of the issues with FQDNs mapping to AAAA records is that in some cases multiple AAAA records mean a multihomed host and in other cases it means multiple hosts providing the same service. If we need to introduce a new "ID" resource record type for multihoming, would it be useful to try to make this host/service distinction more clear at the same time? An example solution would be that the ID record in addition to returning the identifier return the FQDN under which to lookup the locators.

Would destination locator rewriting be a useful way for the routing system to pass some information to the host? Or is source locator rewriting sufficient?

With a context tag sufficiently large, what would happen to the residual threats if redirection was allowed without PK checks; either

by just verifying the H2 hash result (which prevents off-path attackers from redirecting) or by only verifying the correct context tag in the received packets? In that case the public key verification would only occur when there is a conflict i.e. trying to create state for an ID when context state already exists for that ID perhaps with different peer locators.

The mechanisms allow adding locators to a locator set but there is currently no mechanism for removing a locator (e.g., when a host renumbers). Does it make sense to add such a mechanism?

The responder only discovers the peer's locators once they are used as sources in packets. Would it make sense to allow the initiator to pass a list of its locators to the responder? (They would still need to be verified before use to prevent 3rd party DoS attacks [M6SEC]).

12.1. Initiator Confusion vs. "Virtual Hosting"

When A wants to communicate with host B and host X at the same time there can be some confusion since the DNS could return the same identifier for B and X while returning different locator sets. For example,

The lookup of FQDN(B) returns ID(B), Ls(B)

The lookup of FQDN(X) returns ID(B), Ls(X)

The result is that connections that could be intended to go to B and to X would both end up with the same ID at the ULP, but the multihoming shim layer would have two separate locator sets associated with ID(B). Thus at a minimum when the second of the two communications starts there has to be some way to resolve this conflict.

If multiple FQDNs map to the same host, which is common in virtual hosting using IPv4 today, and the locator set is being modified for that host then this could be quite normal; looking up www.foo.com would provide the ID of the peer and a perhaps staler set of locators for the ID than looking up www.bar.com.

Is it reasonable to assume when there is some overlap between Ls(B) and Ls(X) above that this is a normal condition? Should one form the intersection of Ls(B) and Ls(X) and use that for the existing context state? Or at least purge unverified peer locators, those from which the host hasn't seen a challenge response, that are not in the intersection from the locator set

Section 4.1 suggests using a challenge request/response exchange when the second lookup takes place. Should the challenge be performed with the newer or older locator sets? What are the DoS issues in performing such a challenge?

13. FUTURE WORK

It would be desirable to explore making identifiers first class objects and having a lookup system, perhaps based on distributed hash tables, for identifiers. But there are significant scaling issues in this domain.

Instead of making the identifier a hash of a host public key it could be composed in two parts (about 60 bits each):

- A hash of a site public key

- A hash of a host-within site certificate

The certificate would be issued by using the site key.

Thus the verification of a challenge response would consist of verifying that

- the site public key hashes to top N bits of the ID

- the host certificate hashes to the bottom 126-N bits of the ID

- the host certificate is signed by the site public key

- the response is signed by the host public key

While this in itself isn't interesting, it would make it more feasible to use techniques like distributed hash tables to build a mapping system from IDs to locators; once (if?) we could do so the IP identifiers could become a first class object in the architecture.

Doing a DHT which scales to having one entry for every IP addressable device in the Internet is less likely to be feasible than designing an DHT which only needs to scale to one entry per site in the Internet.

There are several issues related to designing such a DHT such as ensuring that lookups for IDs within the same site don't depend on

external connectivity (and the above scheme can easily handle that), making the DHT robust against failures and malice, and making it so that those deploying hosts in the DHT benefit themselves somehow.

The use of CBIDs make authorizing insertion and modification straightforward; a PK challenge using the CBID property can take care of that.

14. ACKNOWLEDGEMENTS

This document has benefited from discussions with (in alphabetical order): Marcelo Bagnulo, Iljitsch van Beijnum, Brian Carpenter, Dave Crocker, Tony Li, Pekka Nikander, Mike O'Dell, and Pekka Savola.

The idea to allow locator rewriting by routers was first presented by Mike O'Dell [ODELL96]. The techniques for avoiding state DoS attacks on the first packet are patterned after [MIPv6]. There are certain similarities with HIP, but the SIM design factors things so that the strength of the identifier (for "rehoming") is separate from payload protection (which can use existing techniques like IPsec).

15. REFERENCES

15.1. Normative References

[M6SEC] Nordmark, E., and T. Li, "Threats relating to IPv6 multihoming solutions", draft-nordmark-multi6-threats-00.txt, October 2003.

[CBID] G. Montenegro and C. Castelluccia, "Statistically Unique and Cryptographically Verifiable Identifiers and Addresses", ISOC NDSS02, San Diego, February 2002.

[ADDR-ARCH] S. Deering, R. Hinden, Editors, "IP Version 6 Addressing Architecture", RFC 3513, April 2003.

15.2. Informative References

[NSRG] Lear, E., and R. Droms, "What's In A Name: Thoughts from the NSRG", draft-irtf-nsrg-report-09.txt (work in progress), March 2003.

[MIPv6] Johnson, D., C. Perkins, and J. Arkko, "Mobility Support in IPv6", draft-ietf-mobileip-ipv6-24.txt (work in progress), June 2003.

[AURA02] Aura, T. and J. Arkko, "MIPv6 BU Attacks and Defenses", draft-aura-mipv6-bu-attacks-01 (work in progress), March 2002.

[NIKANDER03] Nikander, P., T. Aura, J. Arkko, G. Montenegro, and E. Nordmark, "Mobile IP version 6 Route Optimization Security Design Background", draft-nikander-mobileip-v6-ro-sec-01 (work in progress), June 2003.

[PAXSON01] V. Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks", Computer Communication Review 31(3), July 2001.

[INGRESS] Ferguson P., and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", RFC 2827, May 2000.

[ODELL96] O'Dell M., "8+8 - An Alternate Addressing Architecture for IPv6", draft-odell-8+8-00.txt, October 1996,

[IPv6] S. Deering, R. Hinden, Editors, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2461.

[NOID] E. Nordmark, "Multihoming without IP Identifiers", draft-nordmark-multi6-noid-00.txt, October 2003.

[DISCOVERY] T. Narten, E. Nordmark, and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[IPv6-SA] R. Atkinson. "Security Architecture for the Internet Protocol". RFC 2401, November 1998.

[IPv6-AUTH] R. Atkinson. "IP Authentication Header", RFC 2402, November 1998.

[IPv6-ESP] R. Atkinson. "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

AUTHORS' ADDRESSES

Erik Nordmark
Sun Microsystems, Inc.
17 Network Circle
Mountain View, CA
USA

phone: +1 650 786 2921
fax: +1 650 786 5896
email: erik.nordmark@sun.com

Full Copyright Statement

Acknowledgement

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