      Temporary Address Extensions for Stateless Address Autoconfiguration in
                                  IPv6
                     draft-ietf-6man-rfc4941bis-12

Abstract

   This document describes an extension to IPv6 Stateless Address
   Autoconfiguration that causes hosts to generate global scope
   addresses with randomized interface identifiers that change over
   time. Changing global scope addresses over time limits the window of
   time during which eavesdroppers and other information collectors may
   trivially perform address-based network activity correlation when the
   same address is employed for multiple transactions by the same host.
   Additionally, it reduces the window of exposure of a host as being
   accessible via an address that becomes revealed as a result of active
   communication. This document obsoletes RFC4941.

Table of Contents

1. Introduction

   [RFC4862] specifies "Stateless Address Autoconfiguration (SLAAC) for
   IPv6", which typically results in hosts configuring one or more
   "stable" IPv6 addresses composed of a network prefix advertised by a
   local router and a locally-generated Interface Identifier (IID). The
   security and privacy implications of such addresses have been
   discussed in detail in [RFC7721], [RFC7217], and [RFC7707]. This
   document specifies an extension for SLAAC to generate temporary
   addresses, that can help mitigate some of the aforementioned issues.
   This is a revision of RFC4941, and formally obsoletes RFC4941.
   Section 5 describes the changes from [RFC4941].

   The default address selection for IPv6 has been specified in
   [RFC6724]. The determination as to whether to use stable versus
   temporary addresses can in some cases only be made by an application.
   For example, some applications may always want to use temporary
   addresses, while others may want to use them only in some
   circumstances or not at all. An Application Programming Interface
   (API) such as that specified in [RFC5014] can enable individual
   applications to indicate a preference for the use of temporary
   addresses.

   Section 2 provides background information. Section 3 describes a
   procedure for generating temporary addresses. Section 4 discusses
   implications of changing interface identifiers (IIDs). Section 5
   describes the changes from [RFC4941].

1.1. Terminology

   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.

   The terms "public address", "stable address", "temporary address",
   "constant IID", "stable IID", and "temporary IID" are to be
   interpreted as specified in [RFC7721].

   The term "global scope addresses" is used in this document to
   collectively refer to "Global unicast addresses" as defined in
   [RFC4291] and "Unique local addresses" as defined in [RFC4193], and
   not to "globally reachable" addresses, as defined in [RFC8190].

1.2. **Problem Statement**

   Addresses generated using stateless address autoconfiguration
   [RFC4862] contain an embedded interface identifier, which may remain
   stable over time. Anytime a fixed identifier is used in multiple
   contexts, it becomes possible to correlate seemingly unrelated
   activity using this identifier.

   The correlation can be performed by

   o An attacker who is in the path between the host in question and
      the peer(s) to which it is communicating, and who can view the
      IPv6 addresses present in the datagrams.

   o An attacker who can access the communication logs of the peers
      with which the host has communicated.

   Since the identifier is embedded within the IPv6 address, it cannot
   be hidden. This document proposes a solution to this issue by
   generating interface identifiers that vary over time.

   Note that an attacker, who is on path, may be able to perform
   significant correlation based on:

   o The payload contents of unencrypted packets on the wire

   o The characteristics of the packets such as packet size and timing

   Use of temporary addresses will not prevent such correlation, nor
   will it prevent an on-link observer (e.g. the host's default router)
   from tracking all the host's addresses.

2. **Background**

   This section discusses the problem in more detail, provides context
   for evaluating the significance of the concerns in specific
   environments, and makes comparisons with existing practices.

2.1. **Extended Use of the Same Identifier**

   The use of a non-changing interface identifier to form addresses is a
   specific instance of the more general case where a constant
   identifier is reused over an extended period of time and in multiple
   independent activities. Any time the same identifier is used in
   multiple contexts, it becomes possible for that identifier to be used
   to correlate seemingly unrelated activity. For example, a network
   sniffer placed strategically on a link across which all traffic to/
   from a particular host crosses could keep track of which destinations

a host communicated with and at what times. Such information can in
some cases be used to infer things, such as what hours an employee
was active, when someone is at home, etc. Although it might appear
that changing an address regularly in such environments would be
desirable to lessen privacy concerns, it should be noted that the
network prefix portion of an address also serves as a constant
identifier. All hosts at, say, a home, would have the same network
prefix, which identifies the topological location of those hosts.
This has implications for privacy, though not at the same granularity
as the concern that this document addresses. Specifically, all hosts
within a home could be grouped together for the purposes of
collecting information. If the network contains a very small number
of hosts, say, just one, changing just the interface identifier will
not enhance privacy, since the prefix serves as a constant
identifier.

One of the requirements for correlating seemingly unrelated
activities is the use (and reuse) of an identifier that is
recognizable over time within different contexts. IP addresses
provide one obvious example, but there are more. For example,

o Many hosts also have DNS names associated with their addresses, in
   which case the DNS name serves as a similar identifier. Although
   the DNS name associated with an address is more work to obtain (it
   may require a DNS query), the information is often readily
   available. In such cases, changing the address on a host over
   time would do little to address the concerns raised in this
   document, unless the DNS name is changed at the same time as well
   (see Section 4).

o Web browsers and servers typically exchange "cookies" with each
   other [RFC6265]. Cookies allow web servers to correlate a current
   activity with a previous activity. One common usage is to send
   back targeted advertising to a user by using the cookie supplied
   by the browser to identify what earlier queries had been made
   (e.g., for what type of information). Based on the earlier
   queries, advertisements can be targeted to match the (assumed)
   interests of the end-user.

The use of a constant identifier within an address is of special
concern because addresses are a fundamental requirement of
communication and cannot easily be hidden from eavesdroppers and
other parties. Even when higher layers encrypt their payloads,
addresses in packet headers appear in the clear. Consequently, if a
mobile host (e.g., laptop) accessed the network from several
different locations, an eavesdropper might be able to track the
movement of that mobile host from place to place, even if the upper
layer payloads were encrypted.

Changing global scope addresses over time limits the time window over which eavesdroppers and other information collectors may trivially correlate network activity when the same address is employed for multiple transactions by the same host. Additionally, it reduces the window of exposure of a host as being accessible via an address that becomes revealed as a result of active communication.

The security and privacy implications of IPv6 addresses are discussed in detail in [RFC7721], [RFC7707], and [RFC7217].

2.2. Possible Approaches

One approach, compatible with the stateless address autoconfiguration architecture, would be to change the interface identifier portion of an address over time. Changing the interface identifier can make it more difficult to look at the IP addresses in independent transactions and identify which ones actually correspond to the same host, both in the case where the routing prefix portion of an address changes and when it does not.

Many hosts function as both clients and servers. In such cases, the host would need a name (e.g. a DNS domain name) for its use as a server. Whether the address stays fixed or changes has little privacy implication since the name remains constant and serves as a constant identifier. When acting as a client (e.g., initiating communication), however, such a host may want to vary the addresses it uses. In such environments, one may need multiple addresses: a stable address associated with the name, that is used to accept incoming connection requests from other hosts, and a temporary address used to shield the identity of the client when it initiates communication.

On the other hand, a host that functions only as a client may want to employ only temporary addresses for public communication.

To make it difficult to make educated guesses as to whether two different interface identifiers belong to the same host, the algorithm for generating alternate identifiers must include input that has an unpredictable component from the perspective of the outside entities that are collecting information.

3. Protocol Description

The following subsections define the procedures for the generation of IPv6 temporary addresses.

3.1. Design Guidelines

Temporary addresses observe the following properties:

1. Temporary addresses are typically employed for initiating
    outgoing sessions.

2. Temporary addresses are used for a short period of time
    (typically hours to days) and are subsequently deprecated.
    Deprecated addresses can continue to be used for established
    connections, but are not used to initiate new connections.

3. New temporary addresses are generated over time to replace
    temporary addresses that expire.

4. Temporary addresses must have a limited lifetime (limited "valid
    lifetime" and "preferred lifetime" from [RFC4862]). The lifetime
    of an address should be further reduced when privacy-meaningful
    events (such as a host attaching to a different network, or the
    regeneration of a new randomized MAC address) takes place. The
    lifetime of temporary addresses must be statistically different
    for different addresses, such that it is hard to predict or infer
    when a new temporary address is generated, or correlate a newly-
    generated address with an existing one.

5. By default, one address is generated for each prefix advertised
    by stateless address autoconfiguration. The resulting Interface
    Identifiers must be statistically different when addresses are
    configured for different prefixes or different network
    interfaces. This means that, given two addresses, it must be
    difficult for an outside entity to infer whether the addresses
    correspond to the same host or network interface.

6. It must be difficult for an outside entity to predict the
    Interface Identifiers that will be employed for temporary
    addresses, even with knowledge of the algorithm/method employed
    to generate them and/or knowledge of the Interface Identifiers
    previously employed for other temporary addresses. These
    Interface Identifiers must be semantically opaque [RFC7136] and
    must not follow any specific patterns.

3.2. Assumptions

The following algorithm assumes that for a given temporary address,
an implementation can determine the prefix from which it was
generated. When a temporary address is deprecated, a new temporary
address is generated. The specific valid and preferred lifetimes for

the new address are dependent on the corresponding lifetime values
set for the prefix from which it was generated.

Finally, this document assumes that when a host initiates outgoing
communication, temporary addresses can be given preference over
stable addresses (if available), when the device is configured to do
so. [RFC6724] mandates implementations to provide a mechanism, which
allows an application to configure its preference for temporary
addresses over stable addresses. It also allows for an
implementation to prefer temporary addresses by default, so that the
connections initiated by the host can use temporary addresses without
requiring application-specific enablement. This document also
assumes that an API will exist that allows individual applications to
indicate whether they prefer to use temporary or stable addresses and
override the system defaults (see e.g. [RFC5014]).

3.3. Generation of Randomized Interface Identifiers

The following subsections specify example algorithms for generating
temporary interface identifiers that follow the guidelines in
Section 3.1 of this document. The algorithm specified in
Section 3.3.1 benefits from a Pseudo-Random Number Generator (PRNG)
available on the system. The algorithm specified in Section 3.3.2
allows for code reuse by hosts that implement [RFC7217].

3.3.1. Simple Randomized Interface Identifiers

One approach is to select a pseudorandom number of the appropriate
length. A host employing this algorithm should generate IIDs as
follows:

1. Obtain a random number from a pseudo-random number generator
    (PRNG) that can produce random numbers of at least as many bits
    as required for the Interface Identifier (please see the next
    step). [RFC4086] specifies randomness requirements for security.

2. The Interface Identifier is obtained by taking as many bits from
    the random number obtained in the previous step as necessary.
    See [RFC7136] for the necessary number of bits, that is, the
    length of the IID. See also [RFC7421] for a discussion of the
    privacy implications of the IID length. Note: there are no
    special bits in an Interface Identifier [RFC7136].

3. The resulting Interface Identifier MUST be compared against the
    reserved IPv6 Interface Identifiers [RFC5453] [IANA-RESERVED-IID]
    and against those Interface Identifiers already employed in an
    address of the same network interface and the same network
    prefix. In the event that an unacceptable identifier has been

generated, a new interface identifier should be generated, by
repeating the algorithm from the first step.

3.3.2. Hash-based Generation of Randomized Interface Identifiers

The algorithm in [RFC7217] can be augmented for the generation of
temporary addresses. The benefit of this would be that a host could
employ a single algorithm for generating stable and temporary
addresses, by employing appropriate parameters.

Hosts would employ the following algorithm for generating the
temporary IID:

1. Compute a random identifier with the expression:

    RID = F(Prefix, Net_Iface, Network_ID, Time, DAD_Counter,
    secret_key)

    Where:

    RID:
       Random Identifier

    F():
       A pseudorandom function (PRF) that MUST NOT be computable from
       the outside (without knowledge of the secret key). F() MUST
       also be difficult to reverse, such that it resists attempts to
       obtain the secret_key, even when given samples of the output
       of F() and knowledge or control of the other input parameters.
       F() SHOULD produce an output of at least as many bits as
       required for the Interface Identifier. F() could be the
       result of applying a cryptographic hash over an encoded
       version of the function parameters. While this document does
       not recommend a specific mechanism for encoding the function
       parameters (or a specific cryptographic hash function), a
       cryptographically robust construction will ensure that the
       mapping from parameters to the hash function input is an
       injective map, as might be attained by using fixed-width
       encodings and/or length-prefixing variable-length parameters.
       SHA-256 [FIPS-SHS] is one possible option for F(). Note: MD5
       [RFC1321] is considered unacceptable for F() [RFC6151].

    Prefix:
       The prefix to be used for SLAAC, as learned from an ICMPv6
       Router Advertisement message.

    Net_Iface:

The MAC address corresponding to the underlying network
interface card, in the case the link uses IEEE802 link-layer
identifiers. Employing the MAC address for this parameter
(over the other suggested options in RFC7217) means that the
re-generation of a randomized MAC address will result in a
different temporary address.

Network_ID:
   Some network-specific data that identifies the subnet to which
   this interface is attached -- for example, the IEEE 802.11
   Service Set Identifier (SSID) corresponding to the network to
   which this interface is associated. Additionally, "Simple
   Procedures for Detecting Network Attachment in IPv6" ("Simple
   DNA") [RFC6059] describes ideas that could be leveraged to
   generate a Network_ID parameter. This parameter SHOULD be
   employed if some form of "Network_ID" is available.

Time:
   An implementation-dependent representation of time. One
   possible example is the representation in UNIX-like systems
   [OPEN-GROUP], that measure time in terms of the number of
   seconds elapsed since the Epoch (00:00:00 Coordinated
   Universal Time (UTC), 1 January 1970). The addition of the
   "Time" argument results in (statistically) different interface
   identifiers over time.

DAD_Counter:
   A counter that is employed to resolve Duplicate Address
   Detection (DAD) conflicts.

secret_key:
   A secret key that is not known by the attacker. The secret
   key SHOULD be of at least 128 bits. It MUST be initialized to
   a pseudo-random number (see [RFC4086] for randomness
   requirements for security) when the operating system is
   "bootstrapped". The secret_key MUST NOT be employed for any
   other purpose than the one discussed in this section. For
   example, implementations MUST NOT employ the same secret_key
   for the generation of stable addresses [RFC7217] and the
   generation of temporary addresses via this algorithm.

2. The Interface Identifier is finally obtained by taking as many
    bits from the RID value (computed in the previous step) as
    necessary, starting from the least significant bit. See
    [RFC7136] for the necessary number of bits, that is, the length
    of the IID. See also [RFC7421] for a discussion of the privacy
    implications of the IID length. Note: there are no special bits
    in an Interface Identifier [RFC7136].

   3. The resulting Interface Identifier MUST be compared against the
       reserved IPv6 Interface Identifiers [RFC5453] [IANA-RESERVED-IID]
       and against those Interface Identifiers already employed in an
       address of the same network interface and the same network
       prefix. In the event that an unacceptable identifier has been
       generated, the value DAD_Counter should be incremented by 1, and
       the algorithm should be restarted from the first step.

3.4. Generating Temporary Addresses

   [RFC4862] describes the steps for generating a link-local address
   when an interface becomes enabled as well as the steps for generating
   addresses for other scopes. This document extends [RFC4862] as
   follows. When processing a Router Advertisement with a Prefix
   Information option carrying a prefix for the purposes of address
   autoconfiguration (i.e., the A bit is set), the host MUST perform the
   following steps:

   1. Process the Prefix Information Option as defined in [RFC4862],
       adjusting the lifetimes of existing temporary addresses, with the
       overall constraint that no temporary addresses should ever remain
       "valid" or "preferred" for a time longer than
       (TEMP_VALID_LIFETIME) or (TEMP_PREFERRED_LIFETIME -
       DESYNC_FACTOR) respectively. The configuration variables
       TEMP_VALID_LIFETIME and TEMP_PREFERRED_LIFETIME correspond to
       maximum target lifetimes for temporary addresses.

   2. One way an implementation can satisfy the above constraints is to
       associate with each temporary address a creation time (called
       CREATION_TIME) that indicates the time at which the address was
       created. When updating the preferred lifetime of an existing
       temporary address, it would be set to expire at whichever time is
       earlier: the time indicated by the received lifetime or
       (CREATION_TIME + TEMP_PREFERRED_LIFETIME - DESYNC_FACTOR). A
       similar approach can be used with the valid lifetime.

   3. If the host has not configured any temporary address for the
       corresponding prefix, the host SHOULD create a new temporary
       address for such prefix.

       Note:
          For example, a host might implement prefix-specific policies
          such as not configuring temporary addresses for the Unique
          Local IPv6 Unicast Addresses (ULA) [RFC4193] prefix.

   4. When creating a temporary address, the DESYNC_FACTOR MUST be
       computed for this prefix, and the lifetime values MUST be derived
       from the corresponding prefix as follows:

     *   Its Valid Lifetime is the lower of the Valid Lifetime of the
         prefix and TEMP_VALID_LIFETIME.

     *   Its Preferred Lifetime is the lower of the Preferred Lifetime
         of the prefix and TEMP_PREFERRED_LIFETIME - DESYNC_FACTOR.

   5. A temporary address is created only if this calculated Preferred
       Lifetime is greater than REGEN_ADVANCE time units.   In
       particular, an implementation MUST NOT create a temporary address
       with a zero Preferred Lifetime.

   6. New temporary addresses MUST be created by appending a randomized
       interface identifier to the prefix that was received.
       Section 3.3 of this document specifies some sample algorithms for
       generating the randomized interface identifier.

   7. The host MUST perform duplicate address detection (DAD) on the
       generated temporary address. If DAD indicates the address is
       already in use, the host MUST generate a new randomized interface
       identifier, and repeat the previous steps as appropriate up to
       TEMP_IDGEN_RETRIES times. If after TEMP_IDGEN_RETRIES
       consecutive attempts no non-unique address was generated, the
       host MUST log a system error and SHOULD NOT attempt to generate a
       temporary address for the given prefix for the duration of the
       host's attachment to the network via this interface. This allows
       hosts to recover from occasional DAD failures, or otherwise log
       the recurrent address collisions.

3.5. Expiration of Temporary Addresses

   When a temporary address becomes deprecated, a new one MUST be
   generated. This is done by repeating the actions described in
   Section 3.4, starting at step 4). Note that, except for the
   transient period when a temporary address is being regenerated, in
   normal operation at most one temporary address per prefix should be
   in a non-deprecated state at any given time on a given interface.
   Note that if a temporary address becomes deprecated as result of
   processing a Prefix Information Option with a zero Preferred
   Lifetime, then a new temporary address MUST NOT be generated. To
   ensure that a preferred temporary address is always available, a new
   temporary address SHOULD be regenerated slightly before its
   predecessor is deprecated. This is to allow sufficient time to avoid
   race conditions in the case where generating a new temporary address
   is not instantaneous, such as when duplicate address detection must
   be run. The host SHOULD start the address regeneration process
   REGEN_ADVANCE time units before a temporary address would actually be
   deprecated.

As an optional optimization, an implementation MAY remove a
deprecated temporary address that is not in use by applications or
upper layers as detailed in Section 6.

3.6. Regeneration of Temporary Addresses

The frequency at which temporary addresses change depends on how a
device is being used (e.g., how frequently it initiates new
communication) and the concerns of the end user. The most egregious
privacy concerns appear to involve addresses used for long periods of
time (weeks to months to years). The more frequently an address
changes, the less feasible collecting or coordinating information
keyed on interface identifiers becomes. Moreover, the cost of
collecting information and attempting to correlate it based on
interface identifiers will only be justified if enough addresses
contain non-changing identifiers to make it worthwhile. Thus, having
large numbers of clients change their address on a daily or weekly
basis is likely to be sufficient to alleviate most privacy concerns.

There are also client costs associated with having a large number of
addresses associated with a host (e.g., in doing address lookups, the
need to join many multicast groups, etc.). Thus, changing addresses
frequently (e.g., every few minutes) may have performance
implications.

Hosts following this specification SHOULD generate new temporary
addresses on a periodic basis. This can be achieved by generating a
new temporary address at least once every (TEMP_PREFERRED_LIFETIME -
REGEN_ADVANCE - DESYNC_FACTOR) time units. As described above,
generating a new temporary address REGEN_ADVANCE time units before a
temporary address becomes deprecated produces addresses with a
preferred lifetime no larger than TEMP_PREFERRED_LIFETIME. The value
DESYNC_FACTOR is a random value computed for a prefix when a
temporary address is generated, that ensures that clients do not
generate new addresses with a fixed frequency, and that clients do
not synchronize with each other and generate new addresses at exactly
the same time. When the preferred lifetime expires, a new temporary
address MUST be generated using the algorithm specified in
Section 3.4.

Because the precise frequency at which it is appropriate to generate
new addresses varies from one environment to another, implementations
SHOULD provide end users with the ability to change the frequency at
which addresses are regenerated. The default value is given in
TEMP_PREFERRED_LIFETIME and is one day. In addition, the exact time
at which to invalidate a temporary address depends on how
applications are used by end users. Thus, the suggested default
value of two days (TEMP_VALID_LIFETIME) may not be appropriate in all

environments. Implementations SHOULD provide end users with the
ability to override both of these default values.

Finally, when an interface connects to a new (different) link,
existing temporary addresses for the corresponding interface MUST be
eliminated, and new temporary addresses MUST be generated immediately
for use on the new link. If a device moves from one link to another,
generating new temporary addresses ensures that the device uses
different randomized interface identifiers for the temporary
addresses associated with the two links, making it more difficult to
correlate addresses from the two different links as being from the
same hosts. The host MAY follow any process available to it, to
determine that the link change has occurred. One such process is
described by Simple DNA [RFC6059]. Detecting link changes would
prevent link down/up events from causing temporary addresses to be
(unnecessarily) regenerated.

3.7. Implementation Considerations

Devices implementing this specification MUST provide a way for the
end user to explicitly enable or disable the use of temporary
addresses. In addition, a site might wish to disable the use of
temporary addresses in order to simplify network debugging and
operations. Consequently, implementations SHOULD provide a way for
trusted system administrators to enable or disable the use of
temporary addresses.

Additionally, sites might wish to selectively enable or disable the
use of temporary addresses for some prefixes. For example, a site
might wish to disable temporary address generation for "Unique local"
[RFC4193] prefixes while still generating temporary addresses for all
other global prefixes. Another site might wish to enable temporary
address generation only for the prefixes 2001:db8:1::/48 and
2001:db8:2::/48 while disabling it for all other prefixes. To
support this behavior, implementations SHOULD provide a way to enable
and disable generation of temporary addresses for specific prefix
subranges. This per-prefix setting SHOULD override the global
settings on the host with respect to the specified prefix subranges.
Note that the per-prefix setting can be applied at any granularity,
and not necessarily on a per subnet basis.

3.8. Defined Constants and Configuration Variables

Constants and configuration variables defined in this document
include:

TEMP_VALID_LIFETIME -- Default value: 2 days. Users should be able
to override the default value.

TEMP_PREFERRED_LIFETIME -- Default value: 1 day. Users should be
able to override the default value.

Note:
   The TEMP_PREFERRED_LIFETIME value MUST be less than the
   TEMP_VALID_LIFETIME value, to avoid the pathological case where an
   address is employed for new communications, but becomes invalid in
   less than 1 second, disrupting those communications

REGEN_ADVANCE -- 2 + (TEMP_IDGEN_RETRIES * DupAddrDetectTransmits *
RetransTimer / 1000)

Notes:
   This parameter is specified as a function of other protocol
   parameters, to account for the time possibly spent in Duplicate
   Address Detection (DAD) in the worst-case scenario of
   TEMP_IDGEN_RETRIES. This prevents the pathological case where the
   generation of a new temporary address is not started with enough
   anticipation such that a new preferred address is generated before
   the currently-preferred temporary address becomes deprecated.

   RetransTimer is specified in [RFC4861], while
   DupAddrDetectTransmits is specified in [RFC4862]. Since
   RetransTimer is specified in units of milliseconds, this
   expression employs the constant "1000" such that REGEN_ADVANCE is
   expressed in seconds.

MAX_DESYNC_FACTOR -- 0.4 * TEMP_PREFERRED_LIFETIME. Upper bound on
DESYNC_FACTOR.

Note:
   Setting MAX_DESYNC_FACTOR to 0.4 TEMP_PREFERRED_LIFETIME results
   in addresses that have statistically different lifetimes, and a
   maximum of 3 concurrent temporary addresses when the default
   parameters specified in this section are employed.

DESYNC_FACTOR -- A random value within the range 0 -
MAX_DESYNC_FACTOR. It is computed for a prefix each time a temporary
address is generated, and must be smaller than
(TEMP_PREFERRED_LIFETIME - REGEN_ADVANCE).

TEMP_IDGEN_RETRIES -- Default value: 3

4. Implications of Changing Interface Identifiers

The desire to protect individual privacy can conflict with the desire
% to effectively maintain and debug a network. Having clients use
addresses that change over time will make it more difficult to track

down and isolate operational problems. For example, when looking at
packet traces, it could become more difficult to determine whether
one is seeing behavior caused by a single errant host, or by a number
of them.

Network deployments are currently recommended to provide multiple
IPv6 addresses from each prefix to general-purpose hosts [RFC7934].
However, in some scenarios, use of a large number of IPv6 addresses
may have negative implications on network devices that need to
maintain entries for each IPv6 address in some data structures (e.g.,
[RFC7039]). For example, concurrent active use of multiple IPv6
addresses will increase neighbor discovery traffic if Neighbor Caches
in network devices are not large enough to store all addresses on the
link. This can impact performance and energy efficiency on networks
on which multicast is expensive (e.g.
[I-D.ietf-mboned-ieee802-mcast-problems]). Additionally, some
network security devices might incorrectly infer IPv6 address forging
if temporary addresses are regenerated at a high rate.

The use of temporary addresses may cause unexpected difficulties with
some applications. For example, some servers refuse to accept
communications from clients for which they cannot map the IP address
into a DNS name. That is, they perform a DNS PTR query to determine
the DNS name, and may then also perform an AAAA query on the returned
name to verify that the returned DNS name maps back into the address
being used. Consequently, clients not properly registered in the DNS
may be unable to access some services. However, a host's DNS name
(if non-changing) would serve as a constant identifier. The wide
deployment of the extension described in this document could
challenge the practice of inverse-DNS-based "validation", which has
little validity, though it is widely implemented. In order to meet
server challenges, hosts could register temporary addresses in the
DNS using random names (for example, a string version of the random
address itself), albeit at the expense of increased complexity.

In addition, some applications may not behave robustly if temporary
addresses are used and an address expires before the application has
terminated, or if it opens multiple sessions, but expects them to all
use the same addresses.

[RFC4941] employed a randomized temporary Interface Identifier for
generating a set of temporary addresses, such that temporary
addresses configured at a given time for multiple SLAAC prefixes
would employ the same Interface Identifier. Sharing the same IID
among multiple address allowed host to join only one solicited-node
multicast group per temporary address set.

This document requires that the Interface Identifiers of all
temporary addresses on a host are statistically different from each
other. This means that when a network employs multiple prefixes,
each temporary address of a set will result in a different solicited-
node multicast address, and thus the number of multicast groups that
a host must join becomes a function of the number of SLAAC prefixes
employed for generating temporary addresses.

Thus, a network that employs multiple prefixes may require hosts to
join more multicast groups than for an RFC4941 implementation. If
the number of multicast groups were large enough, a node might need
to resort to setting the network interface card to promiscuous mode.
This could cause the node to process more packets than strictly
necessary, and might have a negative impact on battery-life, and on
system performance in general.

We note that since this document reduces the default
TEMP_VALID_LIFETIME from 7 days (in [RFC4941]) to 2 days, the number
of concurrent temporary addresses per SLAAC prefix will be smaller
than for RFC4941 implementations, and thus the number of multicast
groups for a network that employs, say, between 1 and three prefixes
will be similar than of RFC4941 implementations.

Implementations concerned with the maximum number of multicast groups
that would be required to join as a result of configured addresses,
or the overall number of configured addresses, should consider
enforcing implementation-specific limits on e.g. the maximum number
of configured addresses, the maximum number of SLAAC prefixes that
are employed for auto-configuration, and/or the maximum ratio for
TEMP_VALID_LIFETIME/TEMP_PREFERRED_LIFETIME (that ultimately controls
the approximate number of concurrent temporary addresses per SLAAC
prefix). Many of these configuration limits are readily available in
SLAAC and RFC4941 implementations. We note that these configurable
limits are meant to prevent pathological behaviors (as opposed to
simply limiting the usage of IPv6 addresses), since IPv6
implementations are expected to leverage the usage of multiple
addresses [RFC7934].

5. Significant Changes from RFC4941

This section summarizes the substantive changes in this document
relative to RFC 4941.

Broadly speaking, this document introduces the following changes:

o Addresses a number of flaws in the algorithm for generating
   temporary addresses: The aforementioned flaws include the use of
   MD5 for computing the temporary IIDs, and reusing the same IID for

      multiple prefixes (see [RAID2015] and [RFC7721] for further
      details).

   o Allows hosts to employ only temporary addresses:
      [RFC4941] assumed that temporary addresses were configured in
      addition to stable addresses. This document does not imply or
      require the configuration of stable addresses, and thus
      implementations can now configure both stable and temporary
      addresses, or temporary addresses only.

   o Removes the recommendation that temporary addresses be disabled by
      default:
      This is in line with BCP188 ([RFC7258]), and also with BCP204
      ([RFC7934]).

   o Reduces the default maximum Valid Lifetime for temporary
      addresses: The default Valid Lifetime for temporary addresses has
      been reduced from 1 week to 2 days, decreasing the typical number
      of concurrent temporary addresses from 7 to 3. This reduces the
      possible stress on network elements (see Section 4 for further
      details).

   o DESYNC_FACTOR is computed on a per-prefix basis each time a
      temporary address is generated, such that each temporary address
      has a statistically different preferred lifetime, and that
      temporary addresses are not generated at a constant frequency.

   o Changes the requirement to not try to regenerate temporary
      addresses upon DAD failures from "MUST NOT" to "SHOULD NOT".

   o The discussion about the security and privacy implications of
      different address generation techniques has been replaced with
      references to recent work in this area ([RFC7707], [RFC7721], and
      [RFC7217]).

   o Addresses all errata submitted for [RFC4941].

6. Future Work

   An implementation might want to keep track of which addresses are
   being used by upper layers so as to be able to remove a deprecated
   temporary address from internal data structures once no upper layer
   protocols are using it (but not before). This is in contrast to
   current approaches where addresses are removed from an interface when
   they become invalid [RFC4862], independent of whether or not upper
   layer protocols are still using them. For TCP connections, such
   information is available in control blocks. For UDP-based
   applications, it may be the case that only the applications have

   knowledge about what addresses are actually in use. Consequently, an
   implementation generally will need to use heuristics in deciding when
   an address is no longer in use.

7. Implementation Status

   [The RFC-Editor should remove this section before publishing this
   document as an RFC]

   The following are known implementations of this document:

   o FreeBSD kernel: There is a FreeBSD kernel implementation of this
      document, albeit not yet committed. The implementation has been
      done in April 2020 by Fernando Gont <fgont@si6networks.com>. The
      corresponding patch can be found at:
      <https://www.gont.com.ar/code/fgont-patch-freebsd-rfc4941bis.txt>

   o Linux kernel: A Linux kernel implementation of this document has
      been committed to the net-next tree. The implementation has been
      produced in April 2020 by Fernando Gont <fgont@si6networks.com>.
      The corresponding patch can be found at:
      <https://patchwork.ozlabs.org/project/netdev/
      patch/20200501035147.GA1587@archlinux-current.localdomain/>

   o slaacd(8): slaacd(8) has traditionally used different randomized
      interface identifiers for each prefix, and it has recently reduced
      the Valid Lifetime of temporary addresses as specified in
      Section 3.8, thus fully implementing this document. The
      implementation has been done by Florian Obser
      <florian@openbsd.org>, with the update to the temporary address
      Valid Lifetime applied in March 2020. The implementation can be
      found at: <https://github.com/openbsd/src/tree/master/sbin/slaacd>

8. IANA Considerations

   There are no IANA registries within this document. The RFC-Editor
   can remove this section before publication of this document as an
   RFC.

9. Security Considerations

   If a very small number of hosts (say, only one) use a given prefix
   for extended periods of time, just changing the interface identifier
   part of the address may not be sufficient to mitigate address-based
   network activity correlation, since the prefix acts as a constant
   identifier. The procedures described in this document are most
   effective when the prefix is reasonably non static or is used by a
   fairly large number of hosts. Additionally, if a temporary address

is used in a session where the user authenticates, any notion of
"privacy" for that address is compromised for the part(ies) that
receive the authentication information.

While this document discusses ways to limit the lifetime of Interface
Identifiers to reduce the ability of attackers to perform address-
based network activity correlation, the method described is believed
to be ineffective against sophisticated forms of traffic analysis.
To increase effectiveness, one may need to consider the use of more
advanced techniques, such as Onion Routing [ONION].

Ingress filtering has been and is being deployed as a means of
preventing the use of spoofed source addresses in Distributed Denial
of Service (DDoS) attacks. In a network with a large number of
hosts, new temporary addresses are created at a fairly high rate.
This might make it difficult for ingress filtering mechanisms to
distinguish between legitimately changing temporary addresses and
spoofed source addresses, which are "in-prefix" (using a
topologically correct prefix and non-existent interface ID). This
can be addressed by using access control mechanisms on a per-address
basis on the network egress point, though as noted in Section 4 there
are corresponding costs for doing so.

10. Acknowledgments

The authors would like to thank (in alphabetical order) Fred Baker,
Brian Carpenter, Tim Chown, Lorenzo Colitti, Roman Danyliw, David
Farmer, Tom Herbert, Bob Hinden, Christian Huitema, Benjamin Kaduk,
Erik Kline, Gyan Mishra, Dave Plonka, Alvaro Retana, Michael
Richardson, Mark Smith, Dave Thaler, Pascal Thubert, Ole Troan,
Johanna Ullrich, Eric Vyncke, and Timothy Winters, for providing
valuable comments on earlier versions of this document.

This document incorporates errata submitted for [RFC4941] by Jiri
Bohac and Alfred Hoenes.

This document is based on [RFC4941] (a revision of RFC3041). Suresh
Krishnan was the sole author of RFC4941. He would like to
acknowledge the contributions of the IPv6 working group and, in
particular, Jari Arkko, Pekka Nikander, Pekka Savola, Francis Dupont,
Brian Haberman, Tatuya Jinmei, and Margaret Wasserman for their
detailed comments.

Rich Draves and Thomas Narten were the authors of RFC 3041. They
would like to acknowledge the contributions of the IPv6 working group
and, in particular, Ran Atkinson, Matt Crawford, Steve Deering,
Allison Mankin, and Peter Bieringer.

11. References

11.1. Normative References

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker,
           "Randomness Requirements for Security", BCP 106, RFC 4086,
           DOI 10.17487/RFC4086, June 2005,
           <https://www.rfc-editor.org/info/rfc4086>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast
           Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005,
           <https://www.rfc-editor.org/info/rfc4193>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing
           Architecture", RFC 4291, DOI 10.17487/RFC4291, February
           2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman,
           "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861,
           DOI 10.17487/RFC4861, September 2007,
           <https://www.rfc-editor.org/info/rfc4861>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless
           Address Autoconfiguration", RFC 4862,
           DOI 10.17487/RFC4862, September 2007,
           <https://www.rfc-editor.org/info/rfc4862>.

[RFC5453] Krishnan, S., "Reserved IPv6 Interface Identifiers",
           RFC 5453, DOI 10.17487/RFC5453, February 2009,
           <https://www.rfc-editor.org/info/rfc5453>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown,
           "Default Address Selection for Internet Protocol Version 6
           (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012,
           <https://www.rfc-editor.org/info/rfc6724>.

[RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6
           Interface Identifiers", RFC 7136, DOI 10.17487/RFC7136,
           February 2014, <https://www.rfc-editor.org/info/rfc7136>.

[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>.

11.2. Informative References

   [FIPS-SHS]
              NIST, "Secure Hash Standard (SHS)", FIPS
              Publication 180-4, August 2015,
              <https://nvlpubs.nist.gov/nistpubs/FIPS/
              NIST.FIPS.180-4.pdf>.

   [I-D.ietf-mboned-ieee802-mcast-problems]
              Perkins, C., McBride, M., Stanley, D., Kumari, W., and J.
              Zuniga, "Multicast Considerations over IEEE 802 Wireless
              Media", draft-ietf-mboned-ieee802-mcast-problems-12 (work
              in progress), October 2020.

   [IANA-RESERVED-IID]
              IANA, "Reserved IPv6 Interface Identifiers",
              <http://www.iana.org/assignments/ipv6-interface-ids>.

   [ONION]   Reed, MGR., Syverson, PFS., and DMG. Goldschlag, "Proxies
              for Anonymous Routing", Proceedings of the 12th Annual
              Computer Security Applications Conference, San Diego, CA,
              December 1996.

   [OPEN-GROUP]
              The Open Group, "The Open Group Base Specifications Issue
              7 / IEEE Std 1003.1-2008, 2016 Edition",
              Section 4.16 Seconds Since the Epoch, 2016,
              <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/
              contents.html>.

   [RAID2015]
              Ullrich, J. and E. Weippl, "Privacy is Not an Option:
              Attacking the IPv6 Privacy Extension", International
              Symposium on Recent Advances in Intrusion Detection
              (RAID), 2015, <https://www.sba-research.org/wp-
              content/uploads/publications/Ullrich2015Privacy.pdf>.

   [RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321,
              DOI 10.17487/RFC1321, April 1992,
              <https://www.rfc-editor.org/info/rfc1321>.

   [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy
              Extensions for Stateless Address Autoconfiguration in
              IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007,
              <https://www.rfc-editor.org/info/rfc4941>.

   [RFC5014] Nordmark, E., Chakrabarti, S., and J. Laganier, "IPv6
              Socket API for Source Address Selection", RFC 5014,
              DOI 10.17487/RFC5014, September 2007,
              <https://www.rfc-editor.org/info/rfc5014>.

   [RFC6059] Krishnan, S. and G. Daley, "Simple Procedures for
              Detecting Network Attachment in IPv6", RFC 6059,
              DOI 10.17487/RFC6059, November 2010,
              <https://www.rfc-editor.org/info/rfc6059>.

   [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations
              for the MD5 Message-Digest and the HMAC-MD5 Algorithms",
              RFC 6151, DOI 10.17487/RFC6151, March 2011,
              <https://www.rfc-editor.org/info/rfc6151>.

   [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265,
              DOI 10.17487/RFC6265, April 2011,
              <https://www.rfc-editor.org/info/rfc6265>.

   [RFC7039] Wu, J., Bi, J., Bagnulo, M., Baker, F., and C. Vogt, Ed.,
              "Source Address Validation Improvement (SAVI) Framework",
              RFC 7039, DOI 10.17487/RFC7039, October 2013,
              <https://www.rfc-editor.org/info/rfc7039>.

   [RFC7217] Gont, F., "A Method for Generating Semantically Opaque
              Interface Identifiers with IPv6 Stateless Address
              Autoconfiguration (SLAAC)", RFC 7217,
              DOI 10.17487/RFC7217, April 2014,
              <https://www.rfc-editor.org/info/rfc7217>.

   [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an
              Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May
              2014, <https://www.rfc-editor.org/info/rfc7258>.

   [RFC7421] Carpenter, B., Ed., Chown, T., Gont, F., Jiang, S.,
              Petrescu, A., and A. Yourtchenko, "Analysis of the 64-bit
              Boundary in IPv6 Addressing", RFC 7421,
              DOI 10.17487/RFC7421, January 2015,
              <https://www.rfc-editor.org/info/rfc7421>.

   [RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6
              Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016,
              <https://www.rfc-editor.org/info/rfc7707>.

   [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy
              Considerations for IPv6 Address Generation Mechanisms",
              RFC 7721, DOI 10.17487/RFC7721, March 2016,
              <https://www.rfc-editor.org/info/rfc7721>.

   [RFC7934] Colitti, L., Cerf, V., Cheshire, S., and D. Schinazi,
              "Host Address Availability Recommendations", BCP 204,
              RFC 7934, DOI 10.17487/RFC7934, July 2016,
              <https://www.rfc-editor.org/info/rfc7934>.

   [RFC8190] Bonica, R., Cotton, M., Haberman, B., and L. Vegoda,
              "Updates to the Special-Purpose IP Address Registries",
              BCP 153, RFC 8190, DOI 10.17487/RFC8190, June 2017,
              <https://www.rfc-editor.org/info/rfc8190>.

Authors' Addresses

   Fernando Gont
   SI6 Networks
   Evaristo Carriego 2644
   Haedo, Provincia de Buenos Aires 1706
   Argentina

   Phone: +54 11 4650 8472
   Email: fgont@si6networks.com
   URI: https://www.si6networks.com

   Suresh Krishnan
   Kaloom

   Email: suresh@kaloom.com

   Thomas Narten

   Email: narten@cs.duke.edu

   Richard Draves
   Microsoft Research
   One Microsoft Way
   Redmond, WA
   USA

   Email: richdr@microsoft.com