Privacy Extensions for Stateless Address Autoconfiguration in IPv6
draft-fgont-6man-rfc4941bis-00

Abstract

   Nodes use IPv6 stateless address autoconfiguration to generate
   addresses using a combination of locally available information and
   information advertised by routers. Addresses are formed by combining
   network prefixes with an interface identifier. On an interface that
   contains an embedded IEEE Identifier, the interface identifier is
   typically derived from it. On other interface types, the interface
   identifier is generated through other means, for example, via random
   number generation. This document describes an extension to IPv6
   stateless address autoconfiguration for interfaces whose interface
   identifier is derived from an IEEE identifier. Use of the extension
   causes nodes to generate global scope addresses from interface
   identifiers that change over time, even in cases where the interface
   contains an embedded IEEE identifier. Changing the interface
   identifier (and the global scope addresses generated from it) over
   time makes it more difficult for eavesdroppers and other information
   collectors to identify when different addresses used in different
   transactions actually correspond to the same node.

Status of This Memo

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

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

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

   This Internet-Draft will expire on September 26, 2018.

Copyright Notice

Table of Contents

1. Introduction

   Stateless address autoconfiguration [ADDRCONF] defines how an IPv6
   node generates addresses without the need for a Dynamic Host
   Configuration Protocol for IPv6 (DHCPv6) server. Some types of
   network interfaces come with an embedded IEEE Identifier (i.e., a
   link-layer MAC address), and in those cases, stateless address
   autoconfiguration uses the IEEE identifier to generate a 64-bit
   interface identifier [ADDRARCH]. By design, the interface identifier
   is likely to be globally unique when generated in this fashion. The
   interface identifier is in turn appended to a prefix to form a
   128-bit IPv6 address. Note that an IPv6 identifier does not
   necessarily have to be 64 bits in length, but the algorithm specified
   in this document is targeted towards 64-bit interface identifiers.

   All nodes combine interface identifiers (whether derived from an IEEE
   identifier or generated through some other technique) with the
   reserved link-local prefix to generate link-local addresses for their
   attached interfaces. Additional addresses can then be created by
   combining prefixes advertised in Router Advertisements via Neighbor
   Discovery [DISCOVERY] with the interface identifier.

   Not all nodes and interfaces contain IEEE identifiers. In such
   cases, an interface identifier is generated through some other means
   (e.g., at random), and the resultant interface identifier may not be
   globally unique and may also change over time. The focus of this
   document is on addresses derived from IEEE identifiers because
   tracking of individual devices, the concern being addressed here, is
   possible only in those cases where the interface identifier is
   globally unique and non-changing. The rest of this document assumes
   that IEEE identifiers are being used, but the techniques described
   may also apply to interfaces with other types of globally unique and/
   or persistent identifiers.

   This document discusses concerns associated with the embedding of
   non-changing interface identifiers within IPv6 addresses and
   describes extensions to stateless address autoconfiguration that can
   help mitigate those concerns for individual users and in environments
   where such concerns are significant. Section 2 provides background
   information on the issue. Section 3 describes a procedure for
   generating alternate interface identifiers and global scope
   addresses. Section 4 discusses implications of changing interface
   identifiers. The term "global scope addresses" is used in this
   document to collectively refer to "Global unicast addresses" as
   defined in [ADDRARCH] and "Unique local addresses" as defined in
   [ULA].

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

1.2. Problem Statement

Addresses generated using stateless address autoconfiguration
[ADDRCONF] contain an embedded interface identifier, which remains
constant 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 node 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 node has communicated.

Since the identifier is embedded within the IPv6 address, which is a
fundamental requirement of communication, it cannot be easily 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 the packets on the wire

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

Use of temporary addresses will not prevent such payload-based
correlation.

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 node 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 nodes at, say, a home, would have the same network
   prefix, which identifies the topological location of those nodes.
   This has implications for privacy, though not at the same granularity
   as the concern that this document addresses. Specifically, all nodes
   within a home could be grouped together for the purposes of
   collecting information. If the network contains a very small number
   of nodes, say, just one, changing just the interface identifier will
   not enhance privacy at all, 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. Many nodes 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 machine over time would do little to
   address the concerns raised in this document, unless the DNS name is
   changed as well (see Section 4).

   Web browsers and servers typically exchange "cookies" with each other
   [COOKIES]. 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.

2.2. Address Usage in IPv4 Today

Addresses used in today's Internet are often non-changing in practice
for extended periods of time. In an increasing number of sites,
addresses are assigned statically and typically change infrequently.
Over the last few years, sites have begun moving away from static
allocation to dynamic allocation via DHCP [DHCP]. In theory, the
address a client gets via DHCP can change over time, but in practice
servers often return the same address to the same client (unless
addresses are in such short supply that they are reused immediately
by a different node when they become free). Thus, even within sites
using DHCP, clients frequently end up using the same address for
weeks to months at a time.

For home users accessing the Internet over dial-up lines, the
situation is generally different. Such users do not have permanent
connections and are often assigned temporary addresses each time they
connect to their ISP. Consequently, the addresses they use change
frequently over time and are shared among a number of different
users. Thus, an address does not reliably identify a particular
device over time spans of more than a few minutes.

A more interesting case concerns always-on connections (e.g., cable
modems, ISDN, DSL, etc.) that result in a home site using the same
address for extended periods of time. This is a scenario that is
just starting to become common in IPv4 and promises to become more of
a concern as always-on Internet connectivity becomes widely
available.

Finally, it should be noted that nodes that need a (non-changing) DNS
name generally have static addresses assigned to them to simplify the
configuration of DNS servers. Although Dynamic DNS [DDNS] can be
used to update the DNS dynamically, it may not always be available
depending on the administrative policy. In addition, changing an
address but keeping the same DNS name does not really address the
underlying concern, since the DNS name becomes a non-changing
identifier. Servers generally require a DNS name (so clients can
connect to them), and clients often do as well (e.g., some servers
refuse to speak to a client whose address cannot be mapped into a DNS

name that also maps back into the same address). Section 4 describes
one approach to this issue.

2.3. The Concern with IPv6 Addresses

The division of IPv6 addresses into distinct topology and interface
identifier portions raises an issue new to IPv6 in that a fixed
portion of an IPv6 address (i.e., the interface identifier) can
contain an identifier that remains constant even when the topology
portion of an address changes (e.g., as the result of connecting to a
different part of the Internet). In IPv4, when an address changes,
the entire address (including the local part of the address) usually
changes. It is this new issue that this document addresses.

If addresses are generated from an interface identifier, a home
user's address could contain an interface identifier that remains the
same from one dial-up session to the next, even if the rest of the
address changes. The way PPP is used today, however, PPP servers
typically unilaterally inform the client what address they are to use
(i.e., the client doesn't generate one on its own). This practice,
if continued in IPv6, would avoid the concerns that are the focus of
this document.

A more troubling case concerns mobile devices (e.g., laptops, PDAs,
etc.) that move topologically within the Internet. Whenever they
move, they form new addresses for their current topological point of
attachment. This is typified today by the "road warrior" who has
Internet connectivity both at home and at the office. While the
node's address changes as it moves, the interface identifier
contained within the address remains the same (when derived from an
IEEE Identifier). In such cases, the interface identifier can be
used to track the movement and usage of a particular machine. For
example, a server that logs usage information together with source
addresses, is also recording the interface identifier since it is
embedded within an address. Consequently, any data-mining technique
that correlates activity based on addresses could easily be extended
to do the same using the interface identifier. This is of particular
concern with the expected proliferation of next-generation network-
connected devices (e.g., PDAs, cell phones, etc.) in which large
numbers of devices are, in practice, associated with individual users
(i.e., not shared). Thus, the interface identifier embedded within
an address could be used to track activities of an individual, even
as they move topologically within the Internet.

In summary, IPv6 addresses on a given interface generated via
Stateless Autoconfiguration contain the same interface identifier,
regardless of where within the Internet the device connects. This
facilitates the tracking of individual devices (and thus,

potentially, users). The purpose of this document is to define
mechanisms that eliminate this issue in those situations where it is
a concern.

2.4. Possible Approaches

One way to avoid having a static non-changing address is to use
DHCPv6 [DHCPV6] for obtaining addresses. Section 12 of [DHCPV6]
discusses the use of DHCPv6 for the assignment and management of
"temporary addresses", which are never renewed and provide the same
property of temporary addresses described in this document with
regards to the privacy concern.

Another approach, compatible with the stateless address
autoconfiguration architecture, would be to change the interface
identifier portion of an address over time and generate new addresses
from the interface identifier for some address scopes. 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 node, both in the case where the
routing prefix portion of an address changes and when it does not.

Many machines function as both clients and servers. In such cases,
the machine would need a DNS name for its use as a server. Whether
the address stays fixed or changes has little privacy implication
since the DNS name remains constant and serves as a constant
identifier. When acting as a client (e.g., initiating
communication), however, such a machine may want to vary the
addresses it uses. In such environments, one may need multiple
addresses: a "public" (i.e., non-secret) server address, registered
in the DNS, that is used to accept incoming connection requests from
other machines, and a "temporary" address used to shield the identity
of the client when it initiates communication. These two cases are
roughly analogous to telephone numbers and caller ID, where a user
may list their telephone number in the public phone book, but disable
the display of its number via caller ID when initiating calls.

To make it difficult to make educated guesses as to whether two
different interface identifiers belong to the same node, 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. Picking
identifiers from a pseudo-random sequence suffices, so long as the
specific sequence cannot be determined by an outsider examining
information that is readily available or easily determinable (e.g.,
by examining packet contents). This document proposes the generation
of a pseudo-random sequence of interface identifiers via an MD5 hash.
Periodically, the next interface identifier in the sequence is

generated, a new set of temporary addresses is created, and the
previous temporary addresses are deprecated to discourage their
further use. The precise pseudo-random sequence depends on both a
random component and the globally unique interface identifier (when
available), to increase the likelihood that different nodes generate
different sequences.

3. Protocol Description

The goal of this section is to define procedures that:

1. Do not result in any changes to the basic behavior of addresses
    generated via stateless address autoconfiguration [ADDRCONF].

2. Create additional addresses based on a random interface
    identifier for the purpose of initiating outgoing sessions.
    These "random" or temporary addresses would be used for a short
    period of time (hours to days) and would then be deprecated.
    Deprecated address can continue to be used for already
    established connections, but are not used to initiate new
    connections. New temporary addresses are generated periodically
    to replace temporary addresses that expire, with the exact time
    between address generation a matter of local policy.

3. Produce a sequence of temporary global scope addresses from a
    sequence of interface identifiers that appear to be random in the
    sense that it is difficult for an outside observer to predict a
    future address (or identifier) based on a current one, and it is
    difficult to determine previous addresses (or identifiers)
    knowing only the present one.

4. By default, generate a set of addresses from the same
    (randomized) interface identifier, one address for each prefix
    for which a global address has been generated via stateless
    address autoconfiguration. Using the same interface identifier
    to generate a set of temporary addresses reduces the number of IP
    multicast groups a host must join. Nodes join the solicited-node
    multicast address for each unicast address they support, and
    solicited-node addresses are dependent only on the low-order bits
    of the corresponding address. This default behavior was made to
    address the concern that a node that joins a large number of
    multicast groups may be required to put its interface into
    promiscuous mode, resulting in possible reduced performance.

    A node highly concerned about privacy MAY use different interface
    identifiers on different prefixes, resulting in a set of global
    addresses that cannot be easily tied to each other. For example
    a node MAY create different interface identifiers I1, I2, and I3

for use with different prefixes P1, P2, and P3 on the same
interface.

3.1. Assumptions

The following algorithm assumes that each interface maintains an
associated randomized interface identifier. When temporary addresses
are generated, the current value of the associated randomized
interface identifier is used. While the same identifier can be used
to create more than one temporary address, the value SHOULD change
over time as described in Section 3.5.

The algorithm also 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 node initiates outgoing
communication, temporary addresses can be given preference over
public addresses when the device is configured to do so.
[ADDR_SELECT] mandates implementations to provide a mechanism, which
allows an application to configure its preference for temporary
addresses over public addresses. It also allows for an
implementation to prefer temporary addresses by default, so that the
connections initiated by the node 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 public addresses and
override the system defaults.

3.2. Generation of Randomized Interface Identifiers

We describe two approaches for the generation and maintenance of the
randomized interface identifier. The first assumes the presence of
stable storage that can be used to record state history for use as
input into the next iteration of the algorithm across system
restarts. A second approach addresses the case where stable storage
is unavailable and there is a need to generate randomized interface
identifiers without previous state.

The random interface identifier generation algorithm, as described in
this document, uses MD5 as the hash algorithm. The node MAY use
another algorithm instead of MD5 to produce the random interface
identifier.

3.2.1. **When Stable Storage Is Present**

   The following algorithm assumes the presence of a 64-bit "history
   value" that is used as input in generating a randomized interface
   identifier. The very first time the system boots (i.e., out-of-the-
   box), a random value SHOULD be generated using techniques that help
   ensure the initial value is hard to guess [RANDOM]. Whenever a new
   interface identifier is generated, a value generated by the
   computation is saved in the history value for the next iteration of
   the algorithm.

   A randomized interface identifier is created as follows:

   1. Take the history value from the previous iteration of this
       algorithm (or a random value if there is no previous value) and
       append to it the interface identifier generated as described in
       [ADDRARCH].

   2. Compute the MD5 message digest [MD5] over the quantity created in
       the previous step.

   3. Take the leftmost 64-bits of the MD5 digest and set bit 6 (the
       leftmost bit is numbered 0) to zero. This creates an interface
       identifier with the universal/local bit indicating local
       significance only.

   4. Compare the generated identifier against a list of reserved
       interface identifiers and to those already assigned to an address
       on the local device. In the event that an unacceptable
       identifier has been generated, the node MUST restart the process
       at step 1 above, using the rightmost 64 bits of the MD5 digest
       obtained in step 2 in place of the history value in step 1.

   5. Save the generated identifier as the associated randomized
       interface identifier.

   6. Take the rightmost 64-bits of the MD5 digest computed in step 2)
       and save them in stable storage as the history value to be used
       in the next iteration of the algorithm.

   MD5 was chosen for convenience, and because its particular properties
   were adequate to produce the desired level of randomization. The
   node MAY use another algorithm instead of MD5 to produce the random
   interface identifier

   In theory, generating successive randomized interface identifiers
   using a history scheme as above has no advantages over generating
   them at random. In practice, however, generating truly random

numbers can be tricky. Use of a history value is intended to avoid
the particular scenario where two nodes generate the same randomized
interface identifier, both detect the situation via DAD, but then
proceed to generate identical randomized interface identifiers via
the same (flawed) random number generation algorithm. The above
algorithm avoids this problem by having the interface identifier
(which will often be globally unique) used in the calculation that
generates subsequent randomized interface identifiers. Thus, if two
nodes happen to generate the same randomized interface identifier,
they should generate different ones on the follow-up attempt.

3.2.2. In The Absence of Stable Storage

In the absence of stable storage, no history value will be available
across system restarts to generate a pseudo-random sequence of
interface identifiers. Consequently, the initial history value used
above SHOULD be generated at random. A number of techniques might be
appropriate. Consult [RANDOM] for suggestions on good sources for
obtaining random numbers. Note that even though machines may not
have stable storage for storing a history value, they will in many
cases have configuration information that differs from one machine to
another (e.g., user identity, security keys, serial numbers, etc.).
One approach to generating a random initial history value in such
cases is to use the configuration information to generate some data
bits (which may remain constant for the life of the machine, but will
vary from one machine to another), append some random data, and
compute the MD5 digest as before.

3.2.3. Alternate Approaches

Note that there are other approaches to generate random interface
identifiers, albeit with different goals and applicability. One such
approach is Cryptographically Generated Addresses (CGAs) [CGA], which
generate a random interface identifier based on the public key of the
node. The goal of CGAs is to prove ownership of an address and to
prevent spoofing and stealing of existing IPv6 addresses. They are
used for securing neighbor discovery using [SEND]. The CGA random
interface identifier generation algorithm may not be suitable for
privacy addresses because of the following properties:

o It requires the node to have a public key. This means that the
   node can still be identified by its public key.

o The random interface identifier process is computationally
   intensive and hence discourages frequent regeneration.

3.3. Generating Temporary Addresses

   [ADDRCONF] 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 [ADDRCONF] as
   follows. When processing a Router Advertisement with a Prefix
   Information option carrying a global scope prefix for the purposes of
   address autoconfiguration (i.e., the A bit is set), the node MUST
   perform the following steps:

   1. Process the Prefix Information Option as defined in [ADDRCONF],
       either creating a new public address or adjusting the lifetimes
       of existing addresses, both public and temporary. If a received
       option will extend the lifetime of a public address, the
       lifetimes of temporary addresses should be extended, subject to
       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
       approximate 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. When a new public address is created as described in [ADDRCONF],
       the node SHOULD also create a new temporary address.

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

       * Its Valid Lifetime is the lower of the Valid Lifetime of the
          public address or TEMP_VALID_LIFETIME.

       * Its Preferred Lifetime is the lower of the Preferred Lifetime
          of the public address or 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 the
    interface's current randomized interface identifier to the prefix
    that was received.

7. The node MUST perform duplicate address detection (DAD) on the
    generated temporary address. If DAD indicates the address is
    already in use, the node MUST generate a new randomized interface
    identifier as described in Section 3.2 above, 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 node MUST log a system error and MUST
    NOT attempt to generate temporary addresses for that interface.
    Note that DAD MUST be performed on every unicast address
    generated from this randomized interface identifier.

3.4. 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 3.3, starting at step 3). 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 node 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.

3.5. Regeneration of Randomized Interface Identifiers

   The frequency at which temporary addresses changes 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 node (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.

Nodes following this specification SHOULD generate new temporary
addresses on a periodic basis. This can be achieved automatically by
generating a new randomized interface identifier 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
(different for each client) that ensures that clients don't
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 new randomized interface
identifier.

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 one week (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 link, a new randomized
interface identifier SHOULD be generated immediately together with a
new set of temporary addresses. If a device moves from one ethernet
to another, generating a new set of temporary addresses from a
different randomized interface identifier 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 node. The node MAY follow any process available to it, to
determine that the link change has occurred. One such process is
described by Detecting Network Attachment [DNA].

3.6. Deployment 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"
   [ULA] 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::/16 and 2002::/16,
   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 node with
   respect to the specified prefix subranges. Note that the pre-prefix
   setting can be applied at any granularity, and not necessarily on a
   per-subnet basis.

   The use of temporary addresses may cause unexpected difficulties with
   some applications. As described below, some servers refuse to accept
   communications from clients for which they cannot map the IP address
   into a DNS name. 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.
   Consequently, the use of temporary addresses SHOULD be disabled by
   default in order to minimize potential disruptions. Individual
   applications, which have specific knowledge about the normal duration
   of connections, MAY override this as appropriate.

   If a very small number of nodes (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 ensure privacy, 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 nodes.

4. Implications of Changing Interface Identifiers

   The IPv6 addressing architecture goes to some lengths to ensure that
   interface identifiers are likely to be globally unique where easy to
   do so. The widespread use of temporary addresses may result in a

significant fraction of Internet traffic not using addresses in which
the interface identifier portion is globally unique. Consequently,
usage of the algorithms in this document may complicate providing
such a future flexibility, if global uniqueness is necessary.

The desires of protecting individual privacy versus the desire to
effectively maintain and debug a network can conflict with each
other. 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 machine, or by a number of them.

Some servers refuse to grant access to clients for which no DNS name
exists. 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. As noted earlier, however, a
node's DNS name (if non-changing) serves as a constant identifier.
The wide deployment of the extension described in this document could
challenge the practice of inverse-DNS-based "authentication," which
has little validity, though it is widely implemented. In order to
meet server challenges, nodes could register temporary addresses in
the DNS using random names (for example, a string version of the
random address itself).

Use of the extensions defined in this document may complicate
debugging and other operational troubleshooting activities.
Consequently, it may be site policy that temporary addresses should
not be used. Consequently, implementations MUST provide a method for
the end user or trusted administrator to override the use of
temporary addresses.

5. Defined Constants

Constants defined in this document include:

TEMP_VALID_LIFETIME -- Default value: 1 week. 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.

REGEN_ADVANCE -- 5 seconds

MAX_DESYNC_FACTOR -- 10 minutes. Upper bound on DESYNC_FACTOR.

DESYNC_FACTOR -- A random value within the range 0 -
MAX_DESYNC_FACTOR. It is computed once at system start (rather than
each time it is used) and must never be greater than
(TEMP_VALID_LIFETIME - REGEN_ADVANCE).

TEMP_IDGEN_RETRIES -- Default value: 3

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 [ADDRCONF], 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.

The determination as to whether to use public 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. Suitable API extensions will likely
need to be developed to enable individual applications to indicate
with sufficient granularity their needs with regards to the use of
temporary addresses. Recommendations on DNS practices to avoid the
problem described in Section 4 when reverse DNS lookups fail may be
needed. [DNSOP] contains a more detailed discussion of the DNS-
related issues.

While this document discusses ways of obscuring a user's permanent IP
address, the method described is believed to be ineffective against
sophisticated forms of traffic analysis. To increase effectiveness,
one may need to consider use of more advanced techniques, such as
Onion Routing [ONION].

7. Security Considerations

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
nodes, 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.

8. Significant Changes from RFC 3041

This section summarizes the changes in this document relative to RFC
3041 that an implementer of RFC 3041 should be aware of.

   1. Excluded certain interface identifiers from the range of
       acceptable interface identifiers. Interface IDs such as those
       for reserved anycast addresses [RFC2526], etc.

   2. Added a configuration knob that provides the end user with a way
       to enable or disable the use of temporary addresses on a per-
       prefix basis.

   3. Added a check for denial of service attacks using low valid
       lifetimes in router advertisements.

   4. DAD is now run on all temporary addresses, not just the first one
       generated from an interface identifier.

   5. Changed the default setting for usage of temporary addresses to
       be disabled.

   6. The node is now allowed to generate different interface
       identifiers for different prefixes, if it so desires.

   7. The algorithm used for generating random interface identifiers is
       no longer restricted to just MD5.

   8. Reduced default number of retries to 3 and added a configuration
       variable.

   9. Router advertisement (RA) processing algorithm is no longer
       included in the document, and is replaced by a reference to
       [ADDRCONF].

9. Acknowledgments

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.

Suresh Krishnan was the sole author of this version of the document. 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.

10. References

10.1. Normative References

[ADDRARCH]
         Hinden, R. and S. Deering, "IP Version 6 Addressing
         Architecture", RFC 4291, February 2006.

[ADDRCONF]
         Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless
         Address Autoconfiguration", RFC 4862, September 2007.

[DISCOVERY]
         Narten, T., Nordmark, E., Simpson, W., and H. Soliman,
         "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861,
         September 2007.

[MD5]   Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321,
         April 1992.

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

10.2. Informative References

[ADDR_SELECT]
         Draves, R., "Default Address Selection for Internet
         Protocol version 6 (IPv6)", RFC 3484, February 2003.

[CGA]   Aura, T., "Cryptographically Generated Addresses (CGA)",
         RFC 3972, March 2005.

[COOKIES] Kristol, D. and L. Montulli, "HTTP State Management
         Mechanism", RFC 2965, October 2000.

[DDNS]   Vixie, P., Thomson, S., Rekhter, Y., and J. Bound,
         "Dynamic Updates in the Domain Name System (DNS UPDATE)",
         RFC 2136, April 1997.

[DHCP]   Droms, R., "Dynamic Host Configuration Protocol",
         RFC 2131, March 1997.

   [DHCPV6]   Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C.,
                and M. Carney, "Dynamic Host Configuration Protocol for
                IPv6 (DHCPv6)", RFC 3315, July 2003.

   [DNA]   Choi, JH. and G. Daley, "Goals of Detecting Network
                Attachment in IPv6", RFC 4135, August 2005.

   [DNSOP]   Durand, A., Ihren, J., and P. Savola, "Operational
                Considerations and Issues with IPv6 DNS", RFC 4472, April
                2006.

   [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.

   [RANDOM]   Eastlake, D., Schiller, J., and S. Crocker, "Randomness
                Requirements for Security", BCP 106, RFC 4086, June 2005.

   [RFC2526]   Johnson, D. and S. Deering, "Reserved IPv6 Subnet Anycast
                Addresses", RFC 2526, March 1999.

   [SEND]   Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure
                Neighbor Discovery (SEND)", RFC 3971, March 2005.

   [ULA]   Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast
                Addresses", RFC 4193, October 2005.

Authors' Addresses

   Fernando Gont
   SI6 Networks / UTN-FRH
   Evaristo Carriego 2644
   Haedo, Provincia de Buenos Aires 1706
   Argentina

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

   Suresh Krishnan
   Ericsson Research
   8400 Decarie Blvd.
   Town of Mount Royal, QC
   Canada

   Email: suresh.krishnan@ericsson.com

Thomas Narten
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC
USA

Email: narten@us.ibm.com

Richard Draves
Microsoft Research
One Microsoft Way
Redmond, WA
USA

Email: richdr@microsoft.com