Security and Privacy Implications of Numeric Identifiers Employed in Network Protocols
draft-gont-predictable-protocol-ids-00

Abstract

   This document performs an analysis of the security and privacy
   implications of different types of "numeric identifiers" used in IETF
   protocols, and tries to categorize them based on their
   interoperability requirements and the assoiated failure severity when
   such requirements are not met. It describes a number of algorithms
   that have been employed in real implementations to meet such
   requirements and analyzes their security and privacy properties.
   Additionally, it provides advice on possible algorithms that could be
   employed to satisfy the interoperability requirements of each
   identifier type, while minimizing the security and privacy
   implications, thus providing guidance to protocol designers and
   protocol implementers. Finally, it provides recommendations for
   future protocol specifications regarding the specification of the
   aforementioned numeric identifiers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   Network protocols employ a variety of numeric identifiers for
   different protocol entities, ranging from DNS Transaction IDs (TxIDs)
   to transport protocol numbers (e.g. TCP ports) or IPv6 Interface
   Identifiers (IIDs). These identifiers usually have specific
   properties that must be satisfied such that they do not result in
   negative interoperability implications (e.g. uniqueness during a
   specified period of time), and associated failure severities when
   such properties are not met, ranging from soft to hard failures.

   For more than 30 years, a large number of implementations of the TCP/
   IP protocol suite have been subject to a variety of attacks, with
   effects ranging from Denial of Service (DoS) or data injection, to
   information leakage that could be exploited for pervasive monitoring
   [RFC7528]. The root of these issues has been, in many cases, the
   poor selection of identifiers in such protocols, usually as a result
   of an insufficient or misleading specification. While it is
   generally trivial to identify an algorithm that can satisfy the
   interoperability requirements for a given identifier, there exists
   practical evidence that doing so without negatively affecting the
   security and/or privacy properties of the aforementioned protocols is
   prone to error.

   For example, implementations have been subject to security and/or
   privacy issues resulting from:

   o Predictable TCP sequence numbers

   o Predictable transport protocol numbers

   o Predictable IPv4 or IPv6 Fragment Identifiers

   o Predictable IPv6 IIDs

   o Predictable DNS TxIDs

Recent history indicate that when new protocols are standardized or
new protocol implementations are produced, the security and privacy
properties of the associated identifiers tend to be overlooked and
inappropriate algorithms to generate identifier values are either
suggested in the specification or selected by implementators. As a
result, we believe that advice in this area is warranted.

This document contains a non-exhaustive survey of identifiers
employed in various IETF protocols, and aims to categorize such
identifiers based on their interoperability requirements, and the
associated failure severity when such requirements are not met.
Subsequently, it analyzes several algorithms that have been employed
in real implementation to meet such requirements and analyzes their
security and privacy properties, and provides advice on possible
algorithms that could be employed to satisfy the interoperability
requirements of each category, while minimizing the associated
security and privacy implications. Finally, it provides
recommendations for future protocol specifications regarding the
specification of the aforementioned numeric identifiers.

2. Terminology

Identifier:
   A data object in a protocol specification that can be used to
   definetely distinguish a protocol object (a datagram, network
   interface, transport protocol endpoint, session, etc) from all
   other objects of the same type, in a given context. Identifiers
   are usually defined as a series of bits and represented using
   integer values. We note that different identifiers may have
   additional requirements or properties depending on their specific
   use in a protocol. We use the term "identifier" as a generic term
   to refer to any data object in a protocol specification that
   satisfies the identification property stated above.

Failure Severity:
   The consequences of a failure to comply with the interoperability
   requirements of a given identifier. Severity considers the worst
   potential consequence of a failure, determined by the system
   damage and/or time lost to repair the failure. In this document
   we define two types of failure severity: "soft" and "hard".

Hard Failure:
   A hard failure is a non-recoverable condition in which a protocol
   does not operate in the prescribed manner or it operates with
   excessive degradation of service. For example, an established TCP
   connection that is aborted due to an error condition constitutes,
   from the point of view of the transport protocol, a hard failure,

since it enters a state from which normal operation cannot be recovered.

Soft Failure:
   A soft failure is a recoverable condition in which a protocol does not operate in the prescribed manner but normal operation can be resumed automatically in a short period of time. For example, a simple packet-loss event that is subsequently recovered with a retransmission can be considered a soft failure.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Issues with the Specification of Identifiers

While assessing protocol specifications regarding the use of identifiers, we found that most of the issues discussed in this document arise as a result of one of the following:

o Protocol specifications which under-specify the requirements for their identifiers

o Protocol specifications that over-specify their identifiers

o Protocol implementations that simply fail to comply with the specified requirements

A number of protocol implementations (too many of them) simply overlook the security and privacy implications of identifiers. Examples of them are the specification of TCP port numbers in [RFC0793], the specification of TCP sequence numbers in [RFC0793], or the speification of the DNS TxID in [RFC1035].

On the other hand, there are a number of protocol specifications that over-specify some of their associated protocol identifiers. For example, [RFC4291] essentially results in link-layer addresses being embedded in the IPv6 Interface Identifiers (IIDs) when the interoperability requirement of uniqueness could be achieved in other ways that do not result in negative security and privacy implications [I-D.ietf-6man-ipv6-address-generation-privacy]. Similarly, [RFC2460] suggests the use of a global counter for the generation of Fragment Identification values, when the interoperability properties of uniqueness per {Src IP, Dst IP} could be achieved with other algorithms that do not result in negative security and privacy implications.

Finally, there are protocol implementations that simply fail to
comply with existing protocol specifications. For example, some
popular operating systems (notably Microsoft Windows) still fail to
implement randomization of transport protocol ephemeral ports, as
specified in [RFC6056].

4. Timeline of Vulnerability Disclosures Related to Some Sample Identifiers

This section contains a non-exhaustive timeline of vulnerability
disclosures related to some sample identifiers and other work that
has led to advances in this area. The goal of this timeline is to
illustrate:

o That vulnerabilities related to how the values for some
   identifiers are generated and assigned have affected
   implementations for an extremely long period of time.

o That such vulnerabilities, even when addressed for a given
   protocol version, were later reintroduced in new versions or new
   implementations of the same protocol.

o That standardization efforts that discuss and provide advice in
   this area can have a positive effect on protocol specifications
   and protocol implementations.

4.1. IPv4/IPv6 Identification

December 1998:
   [Sanfilippo1998a] finds that predictable IPv4 Identification
   values can be leveraged to count the number of packets sent by a
   target node. [Sanfilippo1998b] explains how to leverage the same
   vulnerability to implement a port-scanning technique known as
   dumb/idle scan. A tool that implements this attack is publicly
   released.

November 1999:
   [Sanfilippo1999] discusses how to leverage predictable IPv4
   Identification to uncover the rules of a number of firewalls.

November 1999:
   [Bellovin2002] explains how the IPv4 Identification field can be
   exploited to count the number of systems behind a NAT.

December 2003:
   [Zalewski2003] explains a technique to perform TCP data injection
   attack based on predictable IPv4 identification values which

requires less effort than TCP injection attacks performed with
bare TCP packets.

November 2005:
    [Silbersack2005] discusses shortcoming in a number of techniques
    to mitigate predictable IPv4 Identification values.

October 2007:
    [Klein2007] describes a weakness in the pseudo random number
    generator (PRNG) in use for the generation of the IP
    Identification by a number of operating systems.

June 2011:
    [Gont2011] describes how to perform idle scan attacks in IPv6.

November 2011:
    Linux mitigates predictable IPv6 Identification values
    [RedHat2011] [SUSE2011] [Ubuntu2011].

December 2011:
    [I-D.ietf-6man-predictable-fragment-id-08] describes the security
    implications of predictable IPv6 Identification values, and
    possible mitigations.

May 2012:
    [Gont2012] notes that some major IPv6 implementations still employ
    predictable IPv6 Identification values.

June 2015:
    [I-D.ietf-6man-predictable-fragment-id-08] notes that some popular
    host and router implementations still employ predictable IPv6
    Identification values.

4.2. TCP Initial Sequence Numbers (ISNs)

September 1981:
    [RFC0793], suggests the use of a global 32-bit ISN generator,
    whose lower bit is incremented roughly every 4 microseconds.
    However, such an ISN generator makes it trivial to predict the ISN
    that a TCP will use for new connections, thus allowing a variety
    of attacks against TCP.

February 1985:
    [Morris1985] was the first to describe how to exploit predictable
    TCP ISNs for forging TCP connections that could then be leveraged
    for trust relationship exploitation.

April 1989:

[Bellovin1989] discussed the security implications of predictable ISNs (along with a range of other protocol-based vulnerabilities).

February 1995:
   [Shimomura1995] reported a real-world exploitation of the attack described in 1985 (ten years before) in [Morris1985].

May 1996:
   [RFC1948] was the first IETF effort, authored by Steven Bellovin, to address predictable TCP ISNs. The same concept specified in this document for TCP ISNs was later proposed for TCP ephemeral ports [RFC6056], TCP Timestamps, and eventually even IPv6 Interface Identifiers [RFC7217].

March 2001:
   [Zalewski2001] provides a detailed analysis of statistical weaknesses in some ISN generators, and includes a survey of the algorithms in use by popular TCP implementations.

May 2001:
   Vulnerability advisories [CERT2001] [USCERT2001] are released regarding statistical weaknesses in some ISN generators, affecting popular TCP/IP implementations.

March 2002:
   [Zalewski2002] updates and complements [Zalewski2001]. It concludes that "while some vendors [...] reacted promptly and tested their solutions properly, many still either ignored the issue and never evaluated their implementations, or implemented a flawed solution that apparently was not tested using a known approach". [Zalewski2002].

February 2012:
   [RFC6528], after 27 years of Morris' original work [Morris1985], formally updates [RFC0793] to mitigate predictable TCP ISNs.

August 2014:
   [I-D.eddy-rfc793bis-04], the upcoming revision of the core TCP protocol specification, incorporates the algorithm specified in [RFC6528] as the recommended algorithm for TCP ISN generation.

5. Protocol Failure Severity

Section 2 defines the concept of "Failure Severity" and two types of failures that we employ throughout this document: soft and hard.

Our analysis of the severity of a failure is performed from the point of view of the protocol in question. However, the corresponding

severity on the upper application or protocol may not be the same as
that of the protocol in question. For example, a TCP connection that
is aborted may or may not result in a hard failure of the upper
application: if the upper application can establish a new TCP
connection without any impact on the application, a hard failure at
the TCP protocol may have no severity at the application level. On
the other hand, if a hard failure of a TCP connection results in
excessive degradation of service at the application layer, it will
also result in a hard failure at the application.

6. Categorizing Identifiers

This section includes a non-exhaustive survey of identifiers, and
proposes a number of categories that can accommodate these
identifiers based on their interoperability requirements and their
failure modes (soft or hard)

Identifier	Interoperability Requirements	Failure Severity
IPv6 Frag ID	Uniqueness (for IP address pair)	Soft/Hard (1)
IPv6 IID	Uniqueness (and constant within IPv6 prefix) (2)	Soft (3)
TCP SEQ	Monotonically-increasing	Hard (4)
TCP eph. port	Uniqueness (for connection ID)	Hard
IPv6 Flow L.	Uniqueness	None (5)
DNS TxID	Uniqueness	None (6)

Table 1: Survey of Identifiers

Notes:

(1)
    While a single collision of Fragment ID values would simply lead
    to a single packet drop (and hence a "soft" failure), repeated
    collisions at high data rates might trash the Fragment ID space,
    leading to a hard failure [RFC4963].

(2)

   While the interoperability requirements are simply that the
   Interface ID results in a unique IPv6 address, for operational
   reasons it is typically desirable that the resulting IPv6 address
   (and hence the corresponding Interface ID) be constant within each
   network [I-D.ietf-6man-default-iids] [RFC7217].

(3)

   While IPv6 Interface IDs must result in unique IPv6 addresses,
   IPv6 Duplicate Address Detection (DAD) [RFC4862] allows for the
   detection of duplicate Interface IDs/addresses, and hence such
   Interface ID collisions can be recovered.

(4)

   In theory there are no interoperability requirements for TCP
   sequence numbers, since the TIME-WAIT state and TCP's "quiet time"
   take care of old segments from previous incarnations of the
   connection. However, a widespread optimization allows for a new
   incarnation of a previous connection to be created if the Initial
   Sequence Number (ISN) of the incoming SYN is larger than the last
   sequence number seen in that direction for the previous
   incarnation of the connection. Thus, monotonically-increasing TCP
   sequence numbers allow for such optimization to work as expected
   [RFC6528].

(5)

   The IPv6 Flow Label is typically employed for load sharing
   [RFC7098], along with the Source and Destination IPv6 addresses.
   Reuse of a Flow Label value for the same set {Source Address,
   Destination Address} would typically cause both flows to be
   multiplexed into the same link. However, as long as this does not
   occur deterministically, it will not result in any negative
   implications.

(6)

   DNS TxIDs are employed, together with the Source Address,
   Destination Address, Source Port, and Destination Port, to match
   DNS requests and responses. However, since an implementation
   knows which DNS requests were sent for that set of {Source
   Address, Destination Address, Source Port, and Destination Port,
   DNS TxID}, a collision of TxID would result, if anything, in a
   small performance penalty (the response would be discarded when it
   is found that it does not answer the query sent in the
   corresponding DNS query).

   Based on the survey above, we can categorize identifiers as follows:

```
+-----+----------------------------------------+--------------------+
| Cat |                Category                |  Sample Proto IDs  |
|  #  |                                        |                    |
+-----+----------------------------------------+--------------------+
|  1  |         Uniqueness (soft failure)      |  IPv6 Flow L., DNS |
|     |                                        |        TxIDs       |
+-----+----------------------------------------+--------------------+
|  2  |         Uniqueness (hard failure)      |  IPv6 Frag ID, TCP |
|     |                                        |    ephemeral port  |
+-----+----------------------------------------+--------------------+
|  3  |   Uniqueness, constant within context  |      IPv6 IIDs     |
|     |             (soft failure)             |                    |
+-----+----------------------------------------+--------------------+
|  4  |   Uniqueness, monotonically increasing |      TCP ISN       |
|     |       within context (hard failure)    |                    |
+-----+----------------------------------------+--------------------+
```

                      Table 2: Identifier Categories

   We note that Category #4 could be considered a generalized case of
   category #3, in which a monotonically increasing element is added to
   a constant (within context) element, such that the resulting
   identifiers are monotonically increasing within a specified context.
   That is, the same algorithm could be employed for both #3 and #4,
   given appropriate parameters.

7. Common Algorithms for Identifier Generation

   The following subsections describe common algorithms found for
   Protocol ID generation for each of the categories above.

7.1. Category #1: Uniqueness (soft failure)

7.1.1. Simple Randomization Algorithm

```
    /* Ephemeral port selection function */
    id_range = max_id - min_id + 1;
    next_id = min_id + (random() % id_range);
    count = next_id;

    do {
        if(check_suitable_id(next_id))
            return next_id;

        if (next_id == max_id) {
            next_id = min_id;
        } else {
            next_id++;
        }

        count--;
    } while (count > 0);

    return ERROR;
```

   Note:
      random() is a function that returns a pseudo-random unsigned
      integer number of appropriate size. Note that the output needs to
      be unpredictable, and typical implementations of POSIX random()
      function do not necessarily meet this requirement. See [RFC4086]
      for randomness requirements for security.

      The function check_suitable_id() can check, when possible, whether
      this identifier is e.g. already in use. When already used, this
      algorithm selects the next available protocol ID.

      All the variables (in this and all the algorithms discussed in
      this document) are unsigned integers.

7.1.2. Another Simple Randomization Algorithm

   The following pseudo-code illustrates another algorithm for selecting
   a random identifier in which, in the event the identifier is found to
   be not suitable (e.g., already in use), another identifier is
   selected randomly:

```
        id_range = max_id - min_id + 1;
        next_id = min_id + (random() % id_range);
        count = id_range;

        do {
            if(check_suitable_id(next_id))
                  return next_id;

            next_id = min_id + (random() % id_range);
            count--;
        } while (count > 0);

        return ERROR;
```

   This algorithm might be unable to select an identifier (i.e., return
   "ERROR") even if there are suitable identifiers available, when there
   are a large number of identifiers "in use".

7.2. Category #2: uniqueness (hard failure)

   One of the most trivial approaches for achieving uniqueness for an
   identifier (with a hard failure mode) is to implement a linear
   function. As a result, all of the algorithms described in
   Section 7.4 are of use for complying the requirements of this
   identifier category.

7.3. Category #3: Uniqueness, constant within context (soft-failure)

   Here we should describe a generic version of RFC7217, and state that
   it is essentially the algorithm in Section 7.4.3 without a counter.

   The goal of this algorithm is to produce identifiers that are
   constant for a given context, but that change when the aforementioned
   context changes.

   Keeping one value for each possible "context" may in many cases be
   considered too onerous in terms of memory requirements. As a
   workaround, the following algorithm employs a calculated technique
   (as opposed to keeping state in memory) to maintain the constant
   identifier for each given context.

   In the following algorithm, the function F() provides (statelessly) a
   constant identifier for each given context.

```
       /* Protocol ID selection function  */
       id_range = max_id - min_id + 1;

       counter = 0;

       do {
           offset = F(CONTEXT, counter, secret_key);
           next_id = min_id + (offset % id_range);

           if(check_suitable_id(next_id))
               return next_id;

           counter++;

       } while (counter <= MAX_RETRIES);

       return ERROR;
```

The function F() provides a "per-CONTEXT" constant identifier for a
given context. 'offset' may take any value within the storage type
range since we are restricting the resulting identifier to be in the
range [min_id, max_id] in a similar way as in the algorithm described
in Section 7.1.1. Collisions can be recovered by incrementing the
'counter' variable and recomputing F().

The function F() should be a cryptographic hash function like SHA-256
[FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for F()
[RFC6151]. CONTEXT is the concatenation of all the elements that
define a given context. For example, if this algorithm is expected
to produce identifiers that are unique per network interface card
(NIC) and SLAAC autoconfiguration prefix, the CONTEXT should be the
concatenation of e.g. the interface index and the SLAAC
autoconfiguration prefix (please see [RFC7217] for an implementation
of this algorithm for the generation of IPv6 IIDs).

The secret should be chosen to be as random as possible (see
[RFC4086] for recommendations on choosing secrets).

7.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)

7.4.1. Predictable Linear Identifiers Algorithm

One of the most trivial ways to achieve uniqueness with a low
identifier reuse frequency is to produce a linear sequence. This
obviously assumes that each identifier will be used for a similar
period of time.

For example, the following algorithm has been employed in a number of
operating systems for selecting IP fragment IDs, TCP ephemeral ports,
etc.

```
/* Initialization at system boot time. Could be random */
next_id = min_id;
id_inc= 1;

/* Identifier selection function */
count = max_id - min_id + 1;

do {
    if (next_id == max_id) {
        next_id = min_id;
    }
    else {
        next_id = next_ifd + id_inc;
    }

    if (check_suitable_id(next_id))
        return next_id;

    count--;

} while (count > 0);

return ERROR;
```

Note:
   check_suitable_id() is a function that checks whether the
   resulting identifier is acceptable (e.g., whether its in use,
   etc.).

For obvious reasons, this algorithm results in predicable sequences.
If a global counter is used (such as "next_id" in the example above),
a node that learns one protocol identifier can also learn or guess
values employed by past and future protocol instances. On the other
hand, when the value of increments is known (such as "1" in this
case), an attacker can sample two values, and learn the number of
identifiers that were generated in-between.

Where identifier reuse would lead to a hard failure, one typical
approach to generate unique identifiers (while minimizing the
security and privacy implications of predictable identifiers) is to
obfuscate the resulting protocol IDs by either:

o Replace the global counter with multiple counters (initialized to
   a random value)

   o Randomizing the "increments"

   Avoiding global counters essentially means that learning one
   identifier for a given context (e.g., one TCP ephemeral port for a
   given {src IP, Dst IP, Dst Port}) is of no use for learning or
   guessing identifiers for a different context (e.g., TCP ephemeral
   ports that involve other peers). However, this may imply keeping one
   additional variable/counter per context, which may be prohibitive in
   some environments. The choice of id_inc has implications on both the
   security and privacy properties of the resulting identifiers, but
   also on the corresponding interoperability properties. On one hand,
   minimizing the increments (as in "id_inc = 1" in our case) generally
   minimizes the identifier reuse frequency, albeit at increased
   predictability. On the other hand, if the increments are randomized
   predictability of the resulting identifiers is reduced, and the
   information leakage produced by global constant increments is
   mitigated.

7.4.2. Per-context Counter Algorithm

   One possible way to achieve similar (or even lower) identifier reuse
   frequency while still avoiding predictable sequences would be to
   employ a per-context counter, as opposed to a global counter. Such
   an algorithm could be described as follows:

```
      /* Initialization at system boot time. Could be random */
      id_inc= 1;

      /* Identifier selection function */
      count = max_id - min_id + 1;

      if(lookup_counter(CONTEXT) == ERROR){
          create_counter(CONTEXT);
      }

      next_id= lookup_counter(CONTEXT);

      do {
          if (next_id == max_id) {
              next_id = min_id;
          }
          else {
              next_id = next_id + id_inc;
          }

          if (check_suitable_id(next_id)){
              store_counter(CONTEXT, next_id);
              return next_id;
          }

          count--;

      } while (count > 0);

       store_counter(CONTEXT, next_id);
       return ERROR;
```

   NOTE:
      lookup_counter() returns the current counter for a given context,
      or an error condition if such a counter does nto exist.

      create_counter() creates a counter for a given context, and
      initializes such counter to a random value.

      store_counter() saves (updates) the current counter for a given
      context.

      check_suitable_id() is a function that checks whether the
      resulting identifier is acceptable (e.g., whether its in use,
      etc.).

   Essentially, whenever a new identifier is to be selected, the
   algorithm checks whether there there is a counter for the

corresponding context. If there is, such counter is incremented to
obtain the new identifier, and the new identifier updates the
corresponding counter. If there is no counter for such context, a
new counter is created an initialized to a random value, and used as
the new identifier.

This algorithm produces a per-context counter, which results in one
linear function for each context. Since the origin of each "line" is
a random value, the resulting values are unknown to an off-path
attacker.

This algorithm has the following drawbacks:

o If, as a result of resource management, the counter for a given
   context must be removed, the last identifier value used for that
   context will be lost. Thus, if subsequently an identifier needs
   to be generated for such context, that counter will need to be
   recreated and reinitialized to random value, thus possibly leading
   to reuse/collistion of identifiers.

o If the identifiers are predictable by the destination system
   (e.g., the destination host represents the context), a vulnerable
   host might possibly leak to third parties the identifiers used by
   other hosts to send traffic to it (i.e., a vulnerable Host B could
   leak to Host C the identifier values that Host A is using to send
   packets to Host B). Appendix A of [RFC7739] describes one
   possible scenario for such leakage in detail.

7.4.3. Simple Hash-Based Algorithm

The goal of this algorithm is to produce monotonically-increasing
sequences, with a randomized initial value, for each given context.
For example, if the identifiers being generated must be unique for
each {src IP, dst IP} set, then each possible combination of {src IP,
dst IP} should have a corresponding "next_id" value.

Keeping one value for each possible "context" may in many cases be
considered too onerous in terms of memory requirements. As a
workaround, the following algorithm employs a calculated technique
(as opposed to keeping state in memory) to maintain the random offset
for each possible context.

In the following algorithm, the function F() provides (statelessly) a
random offset for each given context.

```
    /* Initialization at system boot time. Could be random. */
    counter = 0;

    /* Protocol ID selection function  */
    id_range = max_id - min_id + 1;
    offset = F(CONTEXT, secret_key);
    count = id_range;

    do {
        next_id = min_id +
                (counter + offset) % id_range;

        counter++;

        if(check_suitable_id(next_id))
            return next_id;

        count--;

    } while (count > 0);

    return ERROR;
```

The function F() provides a "per-CONTEXT" fixed offset within the
identifier space. Both the 'offset' and 'counter' variables may take
any value within the storage type range since we are restricting the
resulting identifier to be in the range [min_id, max_id] in a similar
way as in the algorithm described in Section 7.1.1. This allows us
to simply increment the 'counter' variable and rely on the unsigned
integer to wrap around.

The function F() should be a cryptographic hash function like SHA-256
[FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for F()
[RFC6151]. CONTEXT is the concatenation of all the elements that
define a given context. For example, if this algorithm is expected
to produce identifiers that are monotonically-increasing for each set
(Source IP Address, Destination IP Address), the CONTEXT should be
the concatenation of these two values.

The secret should be chosen to be as random as possible (see
[RFC4086] for recommendations on choosing secrets).

It should be noted that, since this algorithm uses a global counter
("counter") for selecting identifiers, if an attacker could, e.g.,
force a client to periodically establish a new TCP connection to an
attacker-controlled machine (or through an attacker-observable
routing path), the attacker could substract consecutive source port

   values to obtain the number of outgoing TCP connections established
   globally by the target host within that time period (up to wrap-
   around issues and five-tuple collisions, of course).

7.4.4. Double-Hash Algorithm

   A trade-off between maintaining a single global 'counter' variable
   and maintaining 2**N 'counter' variables (where N is the width of the
   result of F()) could be achieved as follows. The system would keep
   an array of TABLE_LENGTH integers, which would provide a separation
   of the increment of the 'counter' variable. This improvement could
   be incorporated into the algorithm from Section 7.4.3 as follows:

```
    /* Initialization at system boot time */
    for(i = 0; i < TABLE_LENGTH; i++)
        table[i] = random();

    id_inc = 1;


    /* Protocol ID selection function */
    id_range = max_id - min_id + 1;
    offset = F(CONTEXT, secret_key1);
    index = G(CONTEXT, secret_key2);
    count = id_range;

    do {
        next_id = min_id + (offset + table[index]) % id_range;
        table[index] = table[index] + id_inc;

        if(check_suitable_id(next_id))
            return next_id;

        count--;

    } while (count > 0);

    return ERROR;
```

   'table[]' could be initialized with random values, as indicated by
   the initialization code in pseudo-code above. The function G()
   should be a cryptographic hash function. It should use the same
   CONTEXT as F(), and a secret key value to compute a value between 0
   and (TABLE_LENGTH-1). Alternatively, G() could take an "offset" as
   input, and perform the exclusive-or (XOR) operation between all the
   bytes in 'offset'.

   The array 'table[]' assures that successive identifiers for a given
   context will be monotonically-increasing. However, the increments
   space is separated into TABLE_LENGTH different spaces, and thus
   identifier reuse frequency will be (probabilistically) lower than
   that of the algorithm in Section 7.4.3. That is, the generation of
   identifier for one given context will not necessarily result in
   increments in the identifiers for other contexts.

   It is interesting to note that the size of 'table[]' does not limit
   the number of different identifier sequences, but rather separates
   the *increments* into TABLE_LENGTH different spaces. The identifier
   sequence will result from adding the corresponding entry of 'table[]'
   to the variable 'offset', which selects the actual identifier
   sequence (as in the algorithm from Section 7.4.3).

   An attacker can perform traffic analysis for any "increment space"
   into which the attacker has "visibility" -- namely, the attacker can
   force a node to generate identifiers where G(offset) identifies the
   target "increment space". However, the attacker's ability to perform
   traffic analysis is very reduced when compared to the predictable
   linear identifiers (described in Section 7.4.1) and the hash-based
   identifiers (described in Section 7.4.3). Additionally, an
   implementation can further limit the attacker's ability to perform
   traffic analysis by further separating the increment space (that is,
   using a larger value for TABLE_LENGTH) and/or by randomizing the
   increments.

7.4.5. Random-Increments Algorithm

   This algorithm offers a middle ground between the algorithms that
   select ephemeral ports randomly (such as those described in Sections
   Section 7.1.1 and Section 7.1.2), and those that offer obfuscation
   but no randomization (such as those described in Sections
   Section 7.4.3 and Section 7.4.4).

```
    /* Initialization code at system boot time. */
    next_id = random();         /* Initialization value */
    id_inc = 500;         /* Determines the trade-off */

    /* Identifier selection function */
    id_range = max_id - min_id + 1;

    count = id_range;

    do {
        /* Random increment */
        next_id = next_id + (random() % id_increment) + 1;

        /* Keep the identifier within acceptable range */
        next_id = min_id + (next_id % id_range);

        if(check_suitable_id(next_id))
            return next_id;

        count--;
    } while (count > 0);

    return ERROR;
```

This algorithm aims at producing a monotonically increasing sequence
of identifiers, while avoiding the use of fixed increments, which
would lead to trivially predictable sequences. The value "id_inc"
allows for direct control of the trade-off between the level of
obfuscation and the ID reuse frequency. The smaller the value of
"id_inc", the more similar this algorithm is to a predicable, global
monotonically-increasing ID generation algorithm. The larger the
value of "id_inc", the more similar this algorithm is to the
algorithm described in Section 7.1.1 of this document.

When the identifiers wrap, there is the risk of collisions of
identifiers (i.e., identifier reuse). Therefore, "id_inc" should be
selected according to the following criteria:

o It should maximize the wrapping time of the identifier space.

o It should minimize identifier reuse frequency.

o It should maximize obfuscation.

Clearly, these are competing goals, and the decision of which value
of "id_inc" to use is a trade-off. Therefore, the value of "id_inc"

should be configurable so that system administrators can make the
trade-off for themselves.

8. Common Vulnerabilities Associated with Identifiers

This section analyzes common vulnerabilities associated with the
generation of identifiers for each of the categories identified in
Section 6.

8.1. Category #1: Uniqueness (soft failure)

Possible vulnerabilities associated with identifiers of this category
are:

o Use of trivial algorithms (e.g. global counters) that generate
   predictable identifiers

o Use of flawed PRNGs.

Since the only interoperability requirement for these identifiers is
uniqueness, the obvious approach to generate them is to employ a
PRNG. An implementer should consult [RFC4086] regarding randomness
requirements for security, and consult relevant documentation when
employing a PRNG provided by the underlying system.

Use algorithms other than PRNGs for generating identifiers of this
category is discouraged.

8.2. Category #2: uniqueness (hard failure)

As noted in Section 7.2 this category typically employs the same
algorithms as Category #4, since a monotonically-increasing sequence
tends to minimize the identifier reuse frequency. Therefore, the
vulnerability analysis of Section 8.4 applies to this case.

8.3. Category #3: Uniqueness, constant within context (soft failure)

There are two main vulnerabilities that may be associated with
identifiers of this category:

1. Use algorithms or sources that result in predictable identifiers

2. Employing the same identifier across contexts in which constantcy
    is not required

At times, an implementation or specification may be tempted to employ
a source for the identifier which is known to provide unique values.
However, while unique, the associated identifiers may have other

properties such as being predictable or leaking information about the
node in question. For example, as noted in
[I-D.ietf-6man-ipv6-address-generation-privacy], embedding link-layer
addresses for generating IPv6 IIDs not only results in predictable
values, but also leaks information about the manufacturer of the
network interface card.

On the other hand, using an identifier across contexts where
constantcy is not required can be leveraged for correlation of
activities. On of the most trivial examples of this is the use of
IPv6 IIDs that are constant across networks (such as IIDs that embed
the underlying link-layer address).

8.4. **Category #4: Uniqueness, monotonically increasing within context**
      (hard failure)

A simple way to generalize algorithms employed for generating
identifiers of Category #4 would be as follows:

```
/* Identifier selection function */
count = max_id - min_id + 1;

do {
            linear(CONTEXT)= linear(CONTEXT) + increment();
    next_id= offset(CONTEXT) + linear(CONTEXT);

    if(check_suitable_id(next_id))
        return next_id;

    count--;
} while (count > 0);

return ERROR;
```

Essentially, an identifier (next_id) is generated by adding a linear
function (linear()) to an offset value, which is unknown to the
attacker, and constant for given context.

The following aspects of the algorithm should be considered:

o For the most part, it is the offset() function that results in
   identifiers that are unpredictable by an off-path attacker. While
   the resulting sequence will be monotonically-increasing, the use
   of an offset value that is unknown to the attacker makes the
   resulting values unknown to the attacker.

o The most straightforward "stateless" implementation of offset
   would be that in which offset() is the result of a
   cryptographically-secure hash-function that takes the values that
   identify the context and a "secret" (not shown in the figure
   above) as arguments.

o Another possible (but stateful) approach would be to simply
   generate a random offset and store it in memory, and then look-up
   the corresponding context when a new identifier is to be selected.
   The algorithm in Section 7.4.2 is essentially an implementation of
   this type.

o The linear function is incremented according to increment(). In
   the most trivial case increment() could always return the constant
   "1". But it could also possibly return small integers such the
   increments are randomized.

Considering the generic algorithm illustrated above we can identify
the following possible vulnerabilities:

o If the offset value spans more than the necessary context,
   identifiers could be unnecessarily predictable by other parties,
   since the offset value would be unnecessarily leaked to them. For
   example, an implementation that means to produce a per-destination
   counter but replaces offset() with a constant number (i.e.,
   employs a global counter), will unnecessarily result in
   predictable identifiers.

o The function linear() could be seen as representing the number of
   identifiers that have so far been generated for a given context.
   If linear() spans more than the necessary context, the
   "increments" could be leaked to other parties, thus disclosing
   information about the number of identifiers that have so far been
   generated. For example, an implementation in which linear() is
   implemented as a single global counter will unnecessarily leak
   information the number of identifiers that have been produced.

o increment() determines how the linear() is incremented for each
   identifier that is selected. In the most trivial case,
   increment() will return the integer "1". However, an
   implementation may have increment() return a "small" integer value
   such that even if the current value employed by the generator is
   guessed (see Appendix A of [RFC7739]), the exact next identifier
   to be selected will be slightly harder to identify.

9. Security and Privacy Requirements for Identifiers

   Protocol specifications that specify identifiers should:

   1. Clearly specify the interoperability requirements for selecting
       the aforementioned identifiers.

   2. Provide a security and privacy analysis of the aforementioned
       identifiers.

   3. Recommend an algorithm for generating the aforementioned
       identifiers that mitigates security and privacy issues, such as
       those discussed in Section 8.

10. 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.

11. Security Considerations

   The entire document is about the security and privacy implications of
   identifiers.

12. Acknowledgements

   The authors would like to thank (in alphabetical order) [TBD] for
   providing valuable comments on earlier versions of this document.

13. References

13.1. 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,
              <http://www.rfc-editor.org/info/rfc2119>.

   [RFC0793] Postel, J., "Transmission Control Protocol", STD 7,
              RFC 793, DOI 10.17487/RFC0793, September 1981,
              <http://www.rfc-editor.org/info/rfc793>.

   [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence
              Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February
              2012, <http://www.rfc-editor.org/info/rfc6528>.

   [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6
              (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460,
              December 1998, <http://www.rfc-editor.org/info/rfc2460>.

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

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

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

   [RFC5722] Krishnan, S., "Handling of Overlapping IPv6 Fragments",
              RFC 5722, DOI 10.17487/RFC5722, December 2009,
              <http://www.rfc-editor.org/info/rfc5722>.

   [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,
              <http://www.rfc-editor.org/info/rfc6151>.

   [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,
              <http://www.rfc-editor.org/info/rfc7217>.

   [RFC7098] Carpenter, B., Jiang, S., and W. Tarreau, "Using the IPv6
              Flow Label for Load Balancing in Server Farms", RFC 7098,
              DOI 10.17487/RFC7098, January 2014,
              <http://www.rfc-editor.org/info/rfc7098>.

13.2. Informative References

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

   [RFC7528] Higgs, P. and J. Piesing, "A Uniform Resource Name (URN)
              Namespace for the Hybrid Broadcast Broadband TV (HbbTV)
              Association", RFC 7528, DOI 10.17487/RFC7528, April 2015,
              <http://www.rfc-editor.org/info/rfc7528>.

[RFC1948]   Bellovin, S., "Defending Against Sequence Number Attacks",
            RFC 1948, DOI 10.17487/RFC1948, May 1996,
            <http://www.rfc-editor.org/info/rfc1948>.

[CPNI-TCP]
            Gont, F., "Security Assessment of the Transmission Control
            Protocol (TCP)", United Kingdom's Centre for the
            Protection of National Infrastructure (CPNI) Technical
            Report, 2009, <http://www.gont.com.ar/papers/
            tn-03-09-security-assessment-TCP.pdf>.

[Zalewski2001]
            Zalewski, M., "Strange Attractors and TCP/IP Sequence
            Number Analysis", 2001,
            <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>.

[Zalewski2002]
            Zalewski, M., "Strange Attractors and TCP/IP Sequence
            Number Analysis - One Year Later", 2001,
            <http://lcamtuf.coredump.cx/newtcp/>.

[Bellovin1989]
            Bellovin, S., "Security Problems in the TCP/IP Protocol
            Suite", Computer Communications Review, vol. 19, no. 2,
            pp. 32-48, 1989, <https://www.cs.columbia.edu/~smb/papers/
            ipext.pdf>.

[Joncheray1995]
            Joncheray, L., "A Simple Active Attack Against TCP", Proc.
            Fifth Usenix UNIX Security Symposium, 1995.

[Morris1985]
            Morris, R., "A Weakness in the 4.2BSD UNIX TCP/IP
            Software", CSTR 117, AT&T Bell Laboratories, Murray Hill,
            NJ, 1985, <https://pdos.csail.mit.edu/~rtm/
            papers/117.pdf>.

[USCERT2001]
            US-CERT, , "US-CERT Vulnerability Note VU#498440: Multiple
            TCP/IP implementations may use statistically predictable
            initial sequence numbers", 2001,
            <http://www.kb.cert.org/vuls/id/498440>.

[CERT2001]
            CERT, , "CERT Advisory CA-2001-09: Statistical Weaknesses
            in TCP/IP Initial Sequence Numbers", 2001,
            <http://www.cert.org/advisories/CA-2001-09.html>.

[Shimomura1995]
          Shimomura, T., "Technical details of the attack described
          by Markoff in NYT", Message posted in USENET's
          comp.security.misc newsgroup Message-ID:
          <3g5gkl$5j1@ariel.sdsc.edu>, 1995,
          <http://www.gont.com.ar/docs/post-shimomura-usenet.txt>.

[I-D.eddy-rfc793bis-04]
          Eddy, W., "Transmission Control Protocol Specification",
          draft-eddy-rfc793bis-04 (work in progress), August 2014.

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-
          Protocol Port Randomization", BCP 156, RFC 6056,
          DOI 10.17487/RFC6056, January 2011,
          <http://www.rfc-editor.org/info/rfc6056>.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927,
          DOI 10.17487/RFC5927, July 2010,
          <http://www.rfc-editor.org/info/rfc5927>.

[I-D.gont-6man-flowlabel-security]
          Gont, F., "Security Assessment of the IPv6 Flow Label",
          draft-gont-6man-flowlabel-security-03 (work in progress),
          March 2012.

[RFC1035] Mockapetris, P., "Domain names - implementation and
          specification", STD 13, RFC 1035, DOI 10.17487/RFC1035,
          November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC7739] Gont, F., "Security Implications of Predictable Fragment
          Identification Values", RFC 7739, DOI 10.17487/RFC7739,
          February 2016, <http://www.rfc-editor.org/info/rfc7739>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly
          Errors at High Data Rates", RFC 4963,
          DOI 10.17487/RFC4963, July 2007,
          <http://www.rfc-editor.org/info/rfc4963>.

[Bellovin2002]
          Bellovin, S., "A Technique for Counting NATted Hosts",
          IMW'02 Nov. 6-8, 2002, Marseille, France, 2002.

[Fyodor2004]
          Fyodor, , "Idle scanning and related IP ID games", 2004,
          <http://www.insecure.org/nmap/idlescan.html>.

   [Sanfilippo1998a]
              Sanfilippo, S., "about the ip header id", Post to Bugtraq
              mailing-list, Mon Dec 14 1998,
              <http://seclists.org/bugtraq/1998/Dec/48>.

   [Sanfilippo1998b]
              Sanfilippo, S., "Idle scan", Post to Bugtraq mailing-list,
              1998, <http://www.kyuzz.org/antirez/papers/dumbscan.html>.

   [Sanfilippo1999]
              Sanfilippo, S., "more ip id", Post to Bugtraq mailing-
              list, 1999,
              <http://www.kyuzz.org/antirez/papers/moreipid.html>.

   [Silbersack2005]
              Silbersack, M., "Improving TCP/IP security through
              randomization without sacrificing interoperability",
              EuroBSDCon 2005 Conference, 2005,
              <http://citeseerx.ist.psu.edu/viewdoc/
              download?doi=10.1.1.91.4542&rep=rep1&type=pdf>.

   [Zalewski2003]
              Zalewski, M., "A new TCP/IP blind data injection
              technique?", 2003,
              <http://lcamtuf.coredump.cx/ipfrag.txt>.

   [Klein2007]
              Klein, A., "OpenBSD DNS Cache Poisoning and Multiple O/S
              Predictable IP ID Vulnerability", 2007,
              <http://www.trusteer.com/files/OpenBSD_DNS_Cache_Poisoning
              _and_Multiple_OS_Predictable_IP_ID_Vulnerability.pdf>.

   [Gont2011]
              Gont, F., "Hacking IPv6 Networks (training course)", Hack
              In Paris 2011 Conference Paris, France, June 2011.

   [RedHat2011]
              RedHat, , "RedHat Security Advisory RHSA-2011:1465-1:
              Important: kernel security and bug fix update", 2011,
              <https://rhn.redhat.com/errata/RHSA-2011-1465.html>.

   [Ubuntu2011]
              Ubuntu, , "Ubuntu: USN-1253-1: Linux kernel
              vulnerabilities", 2011,
              <http://www.ubuntu.com/usn/usn-1253-1/>.

   [SUSE2011]
              SUSE, , "SUSE Security Announcement: Linux kernel security
              update (SUSE-SA:2011:046)", 2011,
              <http://lists.opensuse.org/
              opensuse-security-announce/2011-12/msg00011.html>.

   [Gont2012]
              Gont, F., "Recent Advances in IPv6 Security", BSDCan 2012
              Conference Ottawa, Canada. May 11-12, 2012, May 2012.

   [I-D.ietf-6man-predictable-fragment-id-08]
              Gont, F., "Security Implications of Predictable Fragment
              Identification Values", draft-ietf-6man-predictable-
              fragment-id-08 (work in progress), June 2015.

   [I-D.ietf-6man-default-iids]
              Gont, F., Cooper, A., Thaler, D., and S. LIU,
              "Recommendation on Stable IPv6 Interface Identifiers",
              draft-ietf-6man-default-iids-09 (work in progress),
              January 2016.

   [I-D.ietf-6man-ipv6-address-generation-privacy]
              Cooper, A., Gont, F., and D. Thaler, "Privacy
              Considerations for IPv6 Address Generation Mechanisms",
              draft-ietf-6man-ipv6-address-generation-privacy-08 (work
              in progress), September 2015.

   [I-D.ietf-opsec-ipv6-host-scanning]
              Gont, F. and T. Chown, "Network Reconnaissance in IPv6
              Networks", draft-ietf-opsec-ipv6-host-scanning-08 (work in
              progress), August 2015.

   [FIPS-SHS]
              FIPS, , "Secure Hash Standard (SHS)", Federal Information
              Processing Standards Publication 180-4, March 2012,
              <http://csrc.nist.gov/publications/fips/fips180-4/
              fips-180-4.pdf>.

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

Ivan Arce
Fundacion Dr. Manuel Sadosky
Av. Cordoba 744 Piso 5 Oficina I
Ciudad Autonoma de Buenos Aires, Buenos Aires C1054AAT
Argentina

Phone: +54 11 4328 5164
Email: stic@fundacionsadosky.org.ar
URI: http://www.fundacionsadosky.org.ar