Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: January 9, 2020

F. Gont
SI6 Networks
I. Arce
Quarkslab

July 8, 2019

On the Generation of Transient Numeric Identifiers
draft-gont-numeric-ids-generation-04

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 associated failure severity when such requirements are not met. Subsequently, 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, this describes a number of algorithms that have been employed in real implementations to generate transient numeric identifiers and analyzes their security and privacy properties.

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Threat Model 5
4. Issues with the Specification of Identifiers 5
5. Protocol Failure Severity 6
6. Categorizing Identifiers 6
7. Common Algorithms for Identifier Generation 9
  7.1. Category #1: Uniqueness (soft failure) 9
  7.2. Category #2: Uniqueness (hard failure) 10
  7.3. Category #3: Uniqueness, constant within context (soft-failure) 10
  7.4. Category #4: Uniqueness, monotonically increasing within context (hard failure) 12
8. Common Vulnerabilities Associated with Transient Numeric Identifiers 17
  8.1. Network Activity Correlation 17
  8.2. Information Leakage 17
  8.3. Exploitation of Semantics of Transient Numeric Identifiers 19
  8.4. Exploitation of Collisions of Transient Numeric Identifiers 19
9. Vulnerability Analysis of Specific Transient Numeric Identifiers Categories 19
  9.1. Category #1: Uniqueness (soft failure) 19
  9.2. Category #2: Uniqueness (hard failure) 20
  9.3. Category #3: Uniqueness, constant within context (soft failure) 20
  9.4. Category #4: Uniqueness, monotonically increasing within context (hard failure) 20
10. IANA Considerations 22
11. Security Considerations 22
12. Acknowledgements 23
13. References 23

13.1. Normative References 23
13.2. Informative References 24
Appendix A. Flawed Algorithms 27
A.1. Predictable Linear Identifiers Algorithm 27
A.2. Random-Increments Algorithm 28
Authors' Addresses 30

1. Introduction

Network protocols employ a variety of numeric identifiers for different protocol entities, ranging from DNS Transaction IDs (TxIDs) to transport protocol ports (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 an associated failure severity 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 [RFC7258]. The root of these issues has been, in many cases, the poor selection of identifiers in such protocols, usually as a result of insufficient or misleading specifications. While it is generally trivial to identify an algorithm that can satisfy the interoperability requirements of 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 [I-D.gont-numeric-ids-history].

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

- Predictable TCP sequence numbers

- Predictable transport protocol port numbers

- Predictable IPv4 or IPv6 Fragment Identifiers

- Predictable IPv6 Interface Identifiers (IIDs)

- Predictable DNS Transaction Identifiers (TxIDs)

Recent history indicates 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 transient numeric identifiers

are either suggested in the specification or selected by implementers. 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 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 analyzes several algorithms that have been employed in real implementations to meet such requirements and analyzes their security and privacy properties.

2. Terminology

Identifier:
A data object in a protocol specification that can be used to definitely 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. Threat Model

Throughout this document, we assume an attacker does not have physical or logical access to the device(s) being attacked. We assume the attacker can simply send any traffic to the target devices, to e.g. sample identifiers employed by such devices.

4. 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 conditions:

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

- Protocol specifications that over-specify their identifiers

- 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 [I-D.gont-numeric-ids-history]. Examples of them are the specification of TCP port numbers in [RFC0793], the specification of TCP sequence numbers in [RFC0793], or the specification 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 [RFC7721]. Similarly, [RFC2460] suggested 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 transport port randomization, as specified in [RFC6056].

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 ISN	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 [RFC7217] [RFC8064] .

(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 Initial Sequence Numbers (ISNs), 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 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 (soft failure)	IPv6 IIDs
4	Uniqueness, monotonically increasing within context (hard failure)	TCP ISN

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.

This algorithm does not suffer from any of the issues discussed in Section 8.

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".

This algorithm does not suffer from any of the issues discussed in Section 8.

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)

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).

For obvious reasons, the transient network identifiers generated with this algorithm allow for network activity correlation within "CONTEXT". However, this is essentially a design goal of this category of transient numeric identifiers.

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

7.4.1. Per-context Counter Algorithm

One possible way to achieve low 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 not 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:

- 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.

- 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.

Otherwise, the identifiers produced by this algorithm do not suffer from the other issues discussed in Section 8.

7.4.2. 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, this algorithm produces an information leakage (as described in Section 8.2). For example, if this algorithm were used for TCP ephemeral port selection, and an attacker could 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 subtract 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).

7.4.3. 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 7.4.2 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).

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 7.4.2. 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 7.4.2).

An attacker can perform traffic analysis for any "increment space" (i.e., context) 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 Appendix A.1) and the hash-based identifiers (described in Section 7.4.2). 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.

Otherwise, this algorithm does not suffer from the issues discussed in Section 8.

8. Common Vulnerabilities Associated with Transient Numeric Identifiers

8.1. Network Activity Correlation

An identifier that is predictable or stable within a given context allows for network activity correlation within that context.

For example, a stable IPv6 Interface Identifier allows for network activity to be correlated for the context in which that address is stable [RFC7721]. A stable-per-network (as in [RFC7217] allows for network activity correlation within a network, whereas a constant IPv6 Interface Identifier allows not only network activity correlation within the same network, but also across networks ("host tracking").

Predictable transient numeric identifiers can also allow for network activity correlation. For example, a node that generates TCP ISNs with a global counter will typically allow network activity correlation even as it roams across networks, since the communicating nodes could infer the identity of the node based on the TCP ISNs employed for subsequent communication instances. Similarly, a node that generates predictable IPv6 Fragment Identification values could be subject to network activity correlation (see e.g. [Bellovin2002]).

8.2. Information Leakage

Transient numeric identifiers that are not randomized can leak out information to other communicating nodes. For example, it is common to generate identifiers like:

```
ID = offset(CONTEXT_1) + linear(CONTEXT_2);
```

This generic expression generates identifiers by adding a linear function to an offset. The offset is constant within a given context, whereas linear() is a linear function for a given context (possibly different to that of offset()). Identifiers generated with this expression will generally be predictable within CONTEXT_1. Thus, CONTEXT_1 essentially specifies e.g. the context within which network activity correlation is possible thanks to these identifiers. When CONTEXT_1 is "global" (e.g., offset() is simply a constant value), then all the corresponding transient numeric identifiers become predictable in all contexts.

> NOTE: If offset() has a global context and the specific value is known, the resulting identifiers may leak even more information. For example, the if Fragment Identification values are generated with the generic function above, and CONTEXT_1 is "global", then the corresponding identifiers will leak the number of fragmented datagrams sent for CONTEXT_2. If both CONTEXT_1 and CONTEXT_2 are "global", then Fragment Identification values would be generated with a global counter (initialized to offset()), and thus each generated Fragment Identification value would leak the number of fragmented datagrams transmitted by the node since it was bootstrapped.

On the other hand, linear() will be predictable within CONTEXT_2. The predictability of linear(), irrespective of the context and/or predictability of offset(), can leak out information that is of use to attackers. For example, a node that selects ephemeral port numbers on as in:

```
ehemeral_port = offset(Dest_IP) + linear()
```

that is, with a per-destination offset, but global linear() function (e.g., a global counter), will leak information about the number of outgoing connections that have been issued between any two issued outgoing connections.

Similarly, a node that generates Fragment Identification values as in:

```
Frag_ID = offset(Srd_IP, Dst_IP) + linear()
```

will leak out information about the number of fragmented packets that have been transmitted between any two other transmitted fragmented packets. The vulnerabilities described in [Sanfilippo1998a], [Sanfilippo1998b], and [Sanfilippo1999] are all associated with the use of a global linear() function (i.e., a global CONTEXT_2).

8.3. Exploitation of Semantics of Transient Numeric Identifiers

Identifiers that are not semantically opaque tend to be more predictable than semantically-opaque identifiers. For example, a MAC address contains an OUI (Organizationally-Unique Identifier) which identifies the vendor that manufactured the underlying network interface card. This fact may be leveraged by an attacker meaning to "predict" MAC addresses, if he has some knowledge about the possible NIC vendor.

[RFC7707] discusses a number of techniques to reduce the search space when performing IPv6 address-scanning attacks by leveraging the semantics of the IIDs produced by a number by traditional IID-generation algorithms (now replaced by [RFC8064] with [RFC7217]).

8.4. Exploitation of Collisions of Transient Numeric Identifiers

In many cases, th collision of transient network identifiers can have a hard failure severity (or result in a hard failure severity if an attacker can cause multiple collisions deterministically, one after another). For example, predictable Fragment Identification values open the door to Denial of Service (DoS) attacks (see e.g. [RFC5722]. Similarly, predictable TCP ISNs open the door to trivial connection-reset and data injection attacks (see e.g. [Joncheray1995]).

9. Vulnerability Analysis of Specific Transient Numeric Identifiers Categories

The following subsections analyze common vulnerabilities associated with the generation of identifiers for each of the categories identified in Section 6.

9.1. 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 (please see e.g. [Zalewski2001], [Zalewski2002] and [Klein2007])

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

9.2. 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 9.4 applies to this case.

9.3. 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 [RFC7721], 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).

9.4. 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_2)= linear(CONTEXT_2) + increment();
    next_id= offset(CONTEXT_1) + linear(CONTEXT_2);

    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 (CONTEXT_1).

The following aspects of the algorithm should be considered:

- 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.

- 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_key" (not shown in the figure above) as arguments.

- Another possible (but stateful) approach would be to simply generate a random "per-context" 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 7.4.1 is essentially an implementation of this type.

- 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 random integers such the increments are unpredictable.

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 (CONTEXT_2). 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. [Fyodor2004] is one example of how such information leakages can be exploited.

- 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" random 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.

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. [I-D.gont-numeric-ids-sec-considerations] formally requires protocols specifications to include an appropriate analysis of the interoperability, security, and privacy implications of the transient numeric identifiers they specify, while this document analyzes possible algorithms (and their implications) that could be employed to comply with the interoperability properties of a transient numeric identifier, while mitigating the possible security and privacy implications.

12. Acknowledgements

The authors would like to thank (in alphabetical order) Steven Bellovin, Joseph Lorenzo Hall, Gre Norcie, and Martin Thomson, for providing valuable comments on earlier versions of this document.

The authors would like to thank Diego Armando Maradona for his magic and inspiration.

13. References

13.1. Normative References

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc4862>.

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

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

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

[RFC8064] Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", RFC 8064, DOI 10.17487/RFC8064, February 2017, <https://www.rfc-editor.org/info/rfc8064>.

13.2. Informative References

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

[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, <https://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>.

[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>.

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

[I-D.gont-numeric-ids-history]
Gont, F. and I. Arce, "Unfortunate History of Transient Numeric Identifiers", draft-gont-numeric-ids-history-04 (work in progress), March 2019.

[I-D.gont-numeric-ids-sec-considerations]
Gont, F. and I. Arce, "Security Considerations for Transient Numeric Identifiers Employed in Network Protocols", draft-gont-numeric-ids-sec-considerations-03 (work in progress), April 2019.

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

[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>.

[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>.

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

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

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

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

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

[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, <https://www.rfc-editor.org/info/rfc7098>.

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

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

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

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

[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>.

[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>.

[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>.

[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/>.

Appendix A. Flawed Algorithms

The following subsections document algorithms with known negative security and privacy implications.

A.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.

For example, the following algorithm has been employed (see e.g. [Morris1985], [Shimomura1995], [Silbersack2005] and [CPNI-TCP]) 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_id + 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. However, using larger increments than necessary can result in an increased identifier reuse frequency.

A.2. Random-Increments Algorithm

This algorithm offers a middle ground between the algorithms that select ephemeral ports randomly (such as those described in Section 7.1.1 and Section 7.1.2), and those that offer obfuscation but no randomization (such as those described in Section 7.4.2 and Section 7.4.3).

```
/* 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_inc) + 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:

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

- It should minimize identifier reuse frequency.

- 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.

Authors' Addresses

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

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

Ivan Arce
Quarkslab

Email: iarce@quarkslab.com
URI: https://www.quarkslab.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

F. Gont
SI6 Networks
I. Arce
Quarkslab

July 8, 2019

Unfortunate History of Transient Numeric Identifiers
draft-gont-numeric-ids-history-05

Abstract

This document analyzes the timeline of the specification of different types of "numeric identifiers" used in IETF protocols, and how the security and privacy implications of such protocols has been affected as a result of it. It provides concrete evidence that advice in this area is warranted.

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Threat Model 4
4. IPv4/IPv6 Identification 4
5. TCP Initial Sequence Numbers (ISNs) 8
6. IPv6 Interface Identifiers (IIDs) 9
7. NTP Reference IDs (REFID) 12
8. Transport Protocol Port Numbers 13
9. IANA Considerations 14
10. Security Considerations 14
11. Acknowledgements 14
12. References 15
12.1. Normative References 15
12.2. Informative References 17
Authors' Addresses 24

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 severity 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 [RFC7258]. 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 Initial Sequence Numbers (ISNs) (see e.g. [Morris1985])

- o Predictable ephemeral transport protocol numbers (see e.g. [RFC6056] and [Silbersack2005])

- o Predictable IPv4 or IPv6 Fragment Identifiers (see e.g. [RFC5722], [RFC6274], and [RFC7739])

- o Predictable IPv6 IIDs (see e.g. [RFC7721] and [RFC7707])

- o Predictable DNS TxIDs [RFC1035]

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 implementers.

This document contains a non-exhaustive timeline of vulnerability disclosures related to some sample transient numeric identifiers and other work that has led to advances in this area, with the goal of illustrating that:

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

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

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

Other related documents ([I-D.gont-numeric-ids-generation] and [I-D.gont-numeric-ids-sec-considerations]) provide guidance in this area.

2. Terminology

Identifier:
A data object in a protocol specification that can be used to definitely 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. Threat Model

Throughout this document, we assume an attacker does not have physical or logical device to the device(s) being attacked. We assume the attacker can simply send any traffic to the target devices, to e.g. sample identifiers employed by such devices.

4. IPv4/IPv6 Identification

This section presents the timeline of the Identification field both for IPv4 and for IPv6. The reason for presenting both cases in the same section is so that it becomes evident that, while the

Identification value serves the same purpose in both IPv4 and IPv6, the work and research done for the IPv4 case did not affect the IPv6 specifications or implementations.

The IPv4 Identification value is specified in [RFC0791], which specifies the interoperability requirements for the Identification field: the sender must choose the Identification field to be unique for a given source address, destination address, and protocol for the time the datagram (or any fragment of it) could be alive in the internet. It suggests that a node may keep "a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet", and suggests that "since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination". The above may be read as a suggestion to employ per-destination or global counters for the generation of Identification values. While [RFC0791] does not suggest any flawed algorithm for the generation of Identification values, it misses a discussion of the security and privacy implications of employing predictable. This has resulted in virtually all IP4 implementations generating predictable fragment Identification values by means of a global counter, at least at some point during the lifetime of such implementations.

The IPv6 Identification is specified in [RFC2460]. It serves the same purpose as its IPv4 counterpart, with the only difference residing in the length of the corresponding field, and that while the IPv4 Identification field is part of the base IPv4 header, in the IPv6 case it is part of the Fragment header (which may or may not be present in an IPv6 packet). [RFC2460] states, in Section 4.5, that the Identification must be different than that of any other fragmented packet sent recently (within the maximum likely lifetime of a packet) with the same Source Address and Destination Address. Subsequently, it notes that this requirement can be met by means of a wrap-around 32-bit counter that is incremented each time a packet must be fragmented, and that it is an implementation choice whether to use a global or a per-destination counter. Thus, the implementation of the IPv6 Identification is similar to that of the IPv4 case, with the only difference that in the IPv6 case the suggestions to use simple counters is more explicit.

September 1981:

[RFC0791] specifies the interoperability requirements for IPv4 Identification value, but does not specify any requirements in the area of security and privacy.

December 1998:

[Sanfilippo1998a] finds that predictable IPv4 Identification values (generated by most popular implementations) 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.

December 1998:
[RFC2460] suggests that a global counter be used to generate the IPv6 Identification value.

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.

September 2002:
[Fyodor2002] explains how to implement a stealth port-scanning technique by leveraging nodes that employ predictable IPv4 Identification values.

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:
[draft-gont-6man-predictable-fragment-id-00] describes the security implications of predictable IPv6 Identification values, and possible mitigations. This document is published on the

Standards Track, meaning to formally update [RFC2460], to introduce security and privacy requirements on IPv6 Identification values.

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

March 2013:
The 6man WG adopts [I-D.gont-6man-predictable-fragment-id], but changes the track to "BCP" (while still formally updating [RFC2460]), publishing the resulting document as [draft-ietf-6man-predictable-fragment-id-00].

June 2013:
A patch that implements IPv6-based idle-scan in nmap is submitted [Morbitzer2013].

December 2014:
The 6man WG changes the status of the aforementioned IETF Internet Draft to "Informational" and publishes it as [draft-ietf-6man-predictable-fragment-id-02]. As a result, it no longer formally updates [RFC2460].

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

February 2016:
[RFC7739] (based on [I-D.ietf-6man-predictable-fragment-id]) analyzes the security and privacy implications of predictable IPv6 Identification values, and provides guidance for selecting an algorithm to generate such values. However, being published on the Informational track, it does not formally update [RFC2460].

June 2016:
[I-D.ietf-6man-rfc2460bis], revision of [RFC2460], removes the suggestion from RFC2460 to employ a global counter for the generation of IPv6 Identification values, but does not specify any security and privacy requirements for the IPv6 Identification value.

July 2017:
[I-D.ietf-6man-rfc2460bis] is finally published as [RFC8200], obsoleting [RFC2460], and pointing to [RFC7739] for sample algorithms for the generation of IPv6 Fragment Identification values.

June 2019:
[IPID-DEV] notes that the IPv6 ID generator of the current version of a popular operating system is flawed.

5. TCP Initial Sequence Numbers (ISNs)

[RFC0793] suggests that the choice of the ISN of a connection is not arbitrary, but aims to reduce the chances of a stale segment from being accepted by a new incarnation of a previous connection. [RFC0793] suggests the use of a global 32-bit ISN generator that is incremented by 1 roughly every 4 microseconds. However, as a matter of fact, protection against stale segments from a previous incarnation of the connection is enforced by preventing the creation of a new incarnation of a previous connection before 2*MSL have passed since a segment corresponding to the old incarnation was last seen (where "MSL" is the "Maximum Segment Lifetime" [RFC0793]). This is accomplished by the TIME-WAIT state and TCP's "quiet time" concept (see Appendix B of [RFC1323]). Based on the assumption that ISNs are monotonically increasing across connections, many stacks (e.g., 4.2BSD-derived) use the ISN of an incoming SYN segment to perform "heuristics" that enable the creation of a new incarnation of a connection while the previous incarnation is still in the TIME-WAIT state (see p. 945 of [Wright1994]). This avoids an interoperability problem that may arise when a node establishes connections to a specific TCP end-point at a high rate [Silbersack2005].

In the case of TCP, the interoperability requirements for the ISNs are probably not clearly spelled out as one would expect. Furthermore, the suggestion of employing a global counter in [RFC0793] leads to negative security and privacy implications.

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.

6. IPv6 Interface Identifiers (IIDs)

IPv6 Interface Identifiers can be generated in multiple ways: SLAAC [RFC4862], DHCPv6 [RFC8415], and manual configuration. This section focuses on Interface Identifiers resulting from SLAAC.

The Interface Identifier of stable (traditional) IPv6 addresses resulting from SLAAC have traditionally resulted in the underlying link-layer address being embedded in the IID. IPv6 addresses resulting from SLAAC are currently required to employ Modified EUI-64 format identifiers, which essentially embed the underlying link-layer

address of the corresponding network interface. At the time, employing the underlying link-layer address for the IID was seen as a convenient way to obtain a unique address. However, recent awareness about the security and privacy implications of this approach [RFC7707] [RFC7721] has led to the replacement of such flawed scheme with an alternative one that mitigates its security and privacy implications [RFC7217] [RFC8064].

January 1997:
[RFC2073] specifies the syntax of IPv6 global addresses (referred to as "An IPv6 Provider-Based Unicast Address Format" at the time), consistent with the IPv6 addressing architecture specified in [RFC1884]. Hosts are recommended to "generate addresses using link-specific addresses as Interface ID such as 48 bit IEEE-802 MAC addresses".

July 1998:
[RFC2374] specifies "An IPv6 Aggregatable Global Unicast Address Format" (obsoleting [RFC2373]) changing the size of the Interface ID to 64 bits, and specifies that that IIDs must be constructed in IEEE EUI-64 format. How such identifiers are constructed becomes specified in the appropriate "IPv6 over <link>" specification such as "IPv6 over Ethernet".

January 2001:
[RFC3041] recognizes the problem of network activity correlation, and specifies temporary addresses. Temporary addresses are to be used along with stable addresses.

August 2003:
[RFC3587] obsoletes [RFC2374], making the TLA/NLA structure historic. The syntax and recommendations for the traditional stable IIDs remain unchanged, though.

February 2006:
[RFC4291] is published as the latest "IP Version 6 Addressing Architecture", requiring the IIDs of the traditional (stable) autoconfigured addresses to employ the Modified EUI-64 format. The details of constructing such interface identifiers are defined in the appropriate "IPv6 over <link>" specifications.

March 2008:
[RFC5157] provides hints regarding how patterns in IPv6 addresses could be leveraged for the purpose of address scanning.

December 2011:
[draft-gont-6man-stable-privacy-addresses-00] notes that the traditional scheme for generating stable addresses allows for

address scanning, and also does not prevent active node tracking. It also specifies an alternative algorithm meant to replace IIDs based on Modified EUI-64 format identifiers.

November 2012:
The 6man WG adopts [I-D.gont-6man-stable-privacy-addresses] as a working group item (as [draft-ietf-6man-stable-privacy-addresses-00]). However, the specified algorithm no longer formally replaces the Modified EUI-64 format identifiers.

February 2013:
An address-scanning tool (scan6 of [IPv6-Toolkit]) that leverages IPv6 address patterns is released [Gont2013].

July 2013:
[I-D.cooper-6man-ipv6-address-generation-privacy] elaborates on the security and privacy implications on all known algorithms for generating IPv6 IIDs.

January 2014:
The 6man wg publishes [draft-ietf-6man-default-iids-00] ("Recommendation on Stable IPv6 Interface Identifiers"), recommending [I-D.ietf-6man-stable-privacy-addresses] for the generation of stable addresses.

April 2014:
[RFC7217] is published, specifying "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)" as an alternative to (but *not* replacement of) Modified EUI-64 format IIDs.

March 2016:
[RFC7707] (formerly [I-D.gont-opsec-ipv6-host-scanning] and later [I-D.ietf-opsec-ipv6-host-scanning]), about "Network Reconnaissance in IPv6 Networks", is published.

March 2016:
[RFC7721] (formerly [I-D.cooper-6man-ipv6-address-generation-privacy] and later [I-D.ietf-6man-ipv6-address-generation-privacy]), about "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", is published.

May 2016:
[draft-gont-6man-non-stable-iids-00] is published, with the goal of specifying requirements for non-stable addresses, and updating [RFC4941] such that use of only temporary addresses is allowed.

May 2016:
[draft-gont-6man-address-usage-recommendations-00] is published, providing an analysis of how different aspects on an address (from stability to usage mode) affect their corresponding security and privacy implications, and meaning to eventually provide advice in this area.

February 2017:
The 6man wg publishes [RFC8064] ("Recommendation on Stable IPv6 Interface Identifiers") (formely [I-D.ietf-6man-default-iids]), with requirements for stable addresses and a recommendation to employ [RFC7217] for the generation of stable addresses. It formally updated a large number of RFCs.

March 2018:
[draft-fgont-6man-rfc4941bis-00] is published (as suggested by the 6man wg), to address flaws in [RFC4941] by revising it (as an alternative to the [draft-gont-6man-non-stable-iids-00] effort, published in March 2016).

July 2018:
[draft-ietf-6man-rfc4941bis-00] is adopted (as [draft-fgont-6man-rfc4941bis-00]) as a wg item of the 6man wg.

7. NTP Reference IDs (REFID)

The NTP [RFC5905] is employed to avoid timing loops degree-one timing loops in scenarios where two NTP peers are (mutually) the time source of each other.

June 2010:
[RFC5905], "Network Time Protocol Version 4: Protocol and Algorithms Specification" is published. It specifies that for NTP peers with stratum higher than 1 the REFID embeds the IPv4 Address of the time soucre or an MD5 hash of the IPv6 address of the time source.

July 2016:
[draft-stenn-ntp-not-you-refid-00] is published, describing the information leakage produced via de NTP REFID. It proposes that NTP returns a special REFID when a packet employs an IP Source Address that is not believed to be a current NTP peer, but otherwise generates and returns the traditional REFID. It is subsequently adopted by the NTP WG as [I-D.ietf-ntp-refid-updates].

April 2019:

[Gont-NTP] notes that the proposed fix specified in [draft-ietf-ntp-refid-updates-00] is, at the very least, sub-optimal.

8. Transport Protocol Port Numbers

Most (if not all) transport protocols employ "port numbers" to demultiplex packets to the corresponding transport protocol instances.

August 1980:
[RFC0768] notes that the UDP source port is optional and identifies the port of the sending process. It does not specify interoperability requirements for source port selection, nor does it suggest possible ways to select port numbers. Most popular implementations end up selecting source ports from a system-wide global counter.

September 1981:
[RFC0793] (the TCP specification) essentially describes the use of port numbers, and specifies that port numbers should result in a unique socket pair (local address, local port, remote address, remote port). How ephemeral ports (i.e. port numbers for "active opens") are selected, and the port range from which they are selected, are left unspecified.

January 2009:
[RFC5452] mandates the use of port randomization for DNS resolvers, and mandates that implementations must randomize port from the range (53 or 1024, and above) or the largest possible port range. It does not recommend possible algorithms for port randomization, although the document specifically targets DNS resolvers, for which a simple random port suffices (e.g. Algorithm 1 of [RFC6056]). This document led to the implementation of port randomization in the DNS resolver themselves, rather than in the underlying transport-protocols.

January 2011:
[RFC6056] notes that many TCP and UDP implementations result in predictable port numbers, and also notes that many implementations select port numbers from a small portion of the whole port number space. It recommends the implementation and use of ephemeral port randomization, proposes a number of possible algorithms for port randomization, and also recommends to randomize port numbers over the range 1024-65535.

March 2016:

[NIST-NTP] reports a non-normal distribution of the ephemeral port numbers employed by the NTP clients of an Internet Time Service.

April 2019:
[I-D.gont-ntp-port-randomization] notes that some NTP implementations employ the NTP service port (123) as the local port for non-symmetric modes, and aims to update the NTP such that they employ port randomization in such cases, as recommended by [RFC6056]. The proposal experiments some push-back in the relevant working group (NTP WG) [NTP-PORTR].

9. 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.

10. Security Considerations

This document analyzes the timeline of the specification of different types of "numeric identifiers" used in IETF protocols, and how the security and privacy implications of such protocols has been affected as a result of it. It provides concrete evidence that advice in this area is warranted. [I-D.gont-numeric-ids-sec-considerations] formally requires protocol specifications to do a warranted analysis of the interoperability implications of the transient numeric identifiers they specify, and to recommend possible algorithms for their generation, such that possible security and privacy implications are mitigated. [I-D.gont-numeric-ids-generation] analyzes categorizes transient numeric identifiers based on their interoperability requirements and their associated failure modes, and recommends possible algorithms to that can comply with the associated requirements while mitigating possible security and privacy implications.

11. Acknowledgements

The authors would like to thank (in alphabetical order) Dave Crocker, Christian Huitema, and Joe Touch, for providing valuable comments on earlier versions of this document.

The authors would like to thank (in alphabetical order) Steven Bellovin, Joseph Lorenzo Hall, Gre Norcie, and Martin Thomson, for providing valuable comments on [I-D.gont-predictable-numeric-ids], on which this document is based.

Section 5 of this document borrows text from [RFC6528], authored by Fernando Gont and Steven Bellovin.

The authors would like to thank Diego Armando Maradona for his magic and inspiration.

12. References

12.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

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

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <https://www.rfc-editor.org/info/rfc1323>.

[RFC1884] Hinden, R., Ed. and S. Deering, Ed., "IP Version 6 Addressing Architecture", RFC 1884, DOI 10.17487/RFC1884, December 1995, <https://www.rfc-editor.org/info/rfc1884>.

[RFC2073] Rekhter, Y., Lothberg, P., Hinden, R., Deering, S., and J. Postel, "An IPv6 Provider-Based Unicast Address Format", RFC 2073, DOI 10.17487/RFC2073, January 1997, <https://www.rfc-editor.org/info/rfc2073>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, DOI 10.17487/RFC2373, July 1998, <https://www.rfc-editor.org/info/rfc2373>.

[RFC2374] Hinden, R., O'Dell, M., and S. Deering, "An IPv6 Aggregatable Global Unicast Address Format", RFC 2374, DOI 10.17487/RFC2374, July 1998, <https://www.rfc-editor.org/info/rfc2374>.

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

[RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, DOI 10.17487/RFC3041, January 2001, <https://www.rfc-editor.org/info/rfc3041>.

[RFC3587] Hinden, R., Deering, S., and E. Nordmark, "IPv6 Global Unicast Address Format", RFC 3587, DOI 10.17487/RFC3587, August 2003, <https://www.rfc-editor.org/info/rfc3587>.

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

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

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

[RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers", RFC 5452, DOI 10.17487/RFC5452, January 2009, <https://www.rfc-editor.org/info/rfc5452>.

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

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

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

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

12.2. Informative References

[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>.

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

[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>.

[draft-fgont-6man-rfc4941bis-00]
Gont, F., Krishnan, S., Narten, T., and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", draft-fgont-6man-rfc4941bis-00 (work in progress), March 2018.

[draft-gont-6man-address-usage-recommendations-00]
Gont, F. and W. Liu, "IPv6 Address Usage Recommendations", draft-gont-6man-address-usage-recommendations-00 (work in progress), May 2016.

[draft-gont-6man-non-stable-iids-00]
Gont, F. and W. Liu, "Recommendation on Non-Stable IPv6 Interface Identifiers", draft-gont-6man-non-stable-iids-00 (work in progress), May 2016.

[draft-gont-6man-predictable-fragment-id-00]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-gont-6man-predictable-fragment-id-00 (work in progress), December 2011.

[draft-gont-6man-stable-privacy-addresses-00]
Gont, F., "A method for Generating Stable Privacy-Enhanced Addresses with IPv6 Stateless Address Autoconfiguration (SLAAC)", draft-gont-6man-stable-privacy-addresses-00 (work in progress), December 2011.

[draft-ietf-6man-default-iids-00]
Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", draft-ietf-6man-default-iids-00 (work in progress), July 2014.

[draft-ietf-6man-predictable-fragment-id-00]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-ietf-6man-predictable-fragment-id-00 (work in progress), March 2013.

[draft-ietf-6man-predictable-fragment-id-02]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-ietf-6man-predictable-fragment-id-02 (work in progress), December 2014.

[draft-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.

[draft-ietf-6man-rfc4941bis-00]
Gont, F., Krishnan, S., Narten, T., and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", draft-ietf-6man-rfc4941bis-00 (work in progress), July 2018.

[draft-ietf-6man-stable-privacy-addresses-00]
Gont, F., "A method for Generating Stable Privacy-Enhanced Addresses with IPv6 Stateless Address Autoconfiguration (SLAAC)", draft-ietf-6man-stable-privacy-addresses-00 (work in progress), May 2012.

[draft-ietf-ntp-refid-updates-00]
Goldberg, S. and H. Stenn, "Network Time Protocol Not You REFID", draft-ietf-ntp-refid-updates-00 (work in progress), November 2016.

[draft-stenn-ntp-not-you-refid-00]
Goldberg, S. and S. KrishnansTENN, "Network Time Protocol Not You REFID", draft-stenn-ntp-not-you-refid-00 (work in progress), July 2016.

[Fyodor2002]
Fyodor, "Idle scanning and related IP ID games", 2002, <http://www.insecure.org/nmap/idlescan.html>.

[Gont-NTP]
Gont, F., "[Ntp] Comments on draft-ietf-ntp-refid-updates-05", Post to the NTP WG mailing list Message-ID: <d871d66d-4043-d8d0-f924-2191ebb2e2ce@si6networks.com>, April 2019, <https://mailarchive.ietf.org/arch/msg/ntp/NkfTHxUUOdp14Agh3h1IPqfcRRg>.

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

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

[Gont2013]
Gont, F., "Beta release of the SI6 Network's IPv6 Toolkit (help wanted!)", Message posted to the IPv6 Hackers mailing-list Message-ID: <51184548.3030105@si6networks.com>, 1995, <https://lists.si6networks.com/pipermail/ipv6hackers/2013-February/000947.html>.

[I-D.cooper-6man-ipv6-address-generation-privacy]
Cooper, A., Gont, F., and D. Thaler, "Privacy Considerations for IPv6 Address Generation Mechanisms", draft-cooper-6man-ipv6-address-generation-privacy-00 (work in progress), July 2013.

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

[I-D.gont-6man-predictable-fragment-id]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-gont-6man-predictable-fragment-id-03 (work in progress), January 2013.

[I-D.gont-6man-stable-privacy-addresses]
Gont, F., "A method for Generating Stable Privacy-Enhanced Addresses with IPv6 Stateless Address Autoconfiguration (SLAAC)", draft-gont-6man-stable-privacy-addresses-01 (work in progress), March 2012.

[I-D.gont-ntp-port-randomization]
Gont, F. and G. Gont, "Port Randomization in the Network Time Protocol Version 4", draft-gont-ntp-port-randomization-02 (work in progress), July 2019.

[I-D.gont-numeric-ids-generation]
Gont, F. and I. Arce, "On the Generation of Transient Numeric Identifiers", draft-gont-numeric-ids-generation-03 (work in progress), March 2019.

[I-D.gont-numeric-ids-sec-considerations]
Gont, F. and I. Arce, "Security Considerations for Transient Numeric Identifiers Employed in Network Protocols", draft-gont-numeric-ids-sec-considerations-03 (work in progress), April 2019.

[I-D.gont-opsec-ipv6-host-scanning]
Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", draft-gont-opsec-ipv6-host-scanning-02 (work in progress), October 2012.

[I-D.gont-predictable-numeric-ids]
Gont, F. and I. Arce, "Security and Privacy Implications of Numeric Identifiers Employed in Network Protocols", draft-gont-predictable-numeric-ids-03 (work in progress), March 2019.

[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-16 (work in progress), September 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-6man-predictable-fragment-id]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-ietf-6man-predictable-fragment-id-10 (work in progress), October 2015.

[I-D.ietf-6man-rfc2460bis]
Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", draft-ietf-6man-rfc2460bis-13 (work in progress), May 2017.

[I-D.ietf-6man-stable-privacy-addresses]
Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", draft-ietf-6man-stable-privacy-addresses-17 (work in progress), January 2014.

[I-D.ietf-ntp-refid-updates]
Stenn, H. and S. Goldberg, "Network Time Protocol REFID Updates", draft-ietf-ntp-refid-updates-05 (work in progress), March 2019.

[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.

[IPID-DEV]
Klein, A. and B. Pinkas, "From IP ID to Device ID and KASLR Bypass (Extended Version)", June 2019, <https://arxiv.org/pdf/1906.10478.pdf>.

[IPv6-Toolkit]
SI6 Networks, "SI6 Networks' IPv6 Toolkit", <https://www.si6networks.com/tools/ipv6toolkit>.

[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>.

[Morbitzer2013]
Morbitzer, M., "[PATCH] TCP Idle Scan in IPv6", Message posted to the nmap-dev mailing-list, 2013, <http://seclists.org/nmap-dev/2013/q2/394>.

[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>.

[NIST-NTP]
Sherman, J. and J. Levine, "Usage Analysis of the NIST Internet Time Service", Journal of Research of the National Institute of Standards and Technology Volume 121, March 2016, <https://tf.nist.gov/general/pdf/2818.pdf>.

[NTP-PORTR]
Gont, F., "[Ntp] New rev of the NTP port randomization I-D (Fwd: New Version Notification for draft-gont-ntp-port-randomization-01.txt)", 2019, <https://mailarchive.ietf.org/arch/browse/ntp/?gbt=1&index=n09Sb61WkH03lSRtamkELXwEQN4>.

[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>.

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

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

[RFC5157] Chown, T., "IPv6 Implications for Network Scanning", RFC 5157, DOI 10.17487/RFC5157, March 2008, <https://www.rfc-editor.org/info/rfc5157>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6274] Gont, F., "Security Assessment of the Internet Protocol Version 4", RFC 6274, DOI 10.17487/RFC6274, July 2011, <https://www.rfc-editor.org/info/rfc6274>.

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

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

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

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

[RFC8064] Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", RFC 8064, DOI 10.17487/RFC8064, February 2017, <https://www.rfc-editor.org/info/rfc8064>.

[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>.

[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>.

[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>.

[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>.

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

[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>.

[Wright1994]
Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1994.

[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/>.

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

Authors' Addresses

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

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

Ivan Arce
Quarkslab

Email: iarce@quarkslab.com
URI: https://www.quarkslab.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 12, 2019

J. Hall
CDT
M. Aaron
CU Boulder
S. Adams
CDT
B. Jones
N. Feamster
Princeton

March 11, 2019

A Survey of Worldwide Censorship Techniques
draft-hall-censorship-tech-07

Abstract

This document describes the technical mechanisms used by censorship regimes around the world to block or impair Internet traffic. It aims to make designers, implementers, and users of Internet protocols aware of the properties being exploited and mechanisms used to censor end-user access to information. This document makes no suggestions on individual protocol considerations, and is purely informational, intended to be a reference.

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 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Technical Prescription 3
3. Technical Identification 4
3.1. Points of Control 4
3.2. Application Layer 5
3.2.1. HTTP Request Header Identification 5
3.2.2. HTTP Response Header Identification 6
3.2.3. Instrumenting Content Providers 7
3.2.4. Deep Packet Inspection (DPI) Identification 8
3.3. Transport Layer 10
3.3.1. Shallow Packet Inspection and TCP/IP Header Identification 10
3.3.2. Protocol Identification 11
4. Technical Interference 12
4.1. Application Layer 12
4.1.1. DNS Interference 12
4.2. Transport Layer 14
4.2.1. Performance Degradation 14
4.2.2. Packet Dropping 15
4.2.3. RST Packet Injection 15
4.3. Multi-layer and Non-layer 16
4.3.1. Distributed Denial of Service (DDoS) 16
4.3.2. Network Disconnection or Adversarial Route Announcement 17
5. Non-Technical Prescription 18
6. Non-Technical Interference 18
6.1. Self-Censorship 18
6.2. Domain Name Reallocation 19
6.3. Server Takedown 19
6.4. Notice and Takedown 19
7. Contributors 19
8. Informative References 20
Authors' Addresses 29

1. Introduction

Censorship is where an entity in a position of power - such as a government, organization, or individual - suppresses communication that it considers objectionable, harmful, sensitive, politically incorrect or inconvenient. (Although censors that engage in censorship must do so through legal, military, or other means, this document focuses largely on technical mechanisms used to achieve network censorship.)

This document describes the technical mechanisms that censorship regimes around the world use to block or degrade Internet traffic (see [RFC7754] for a discussion of Internet blocking and filtering in terms of implications for Internet architecture, rather than end-user access to content and services).

We describe three elements of Internet censorship: prescription, identification, and interference. Prescription is the process by which censors determine what types of material they should block, i.e. they decide to block a list of pornographic websites. Identification is the process by which censors classify specific traffic to be blocked or impaired, i.e. the censor blocks or impairs all webpages containing "sex" in the title or traffic to www.sex.example. Interference is the process by which the censor intercedes in communication and prevents access to censored materials by blocking access or impairing the connection.

2. Technical Prescription

Prescription is the process of figuring out what censors would like to block [Glanville-2008]. Generally, censors aggregate information "to block" in blacklists or using real-time heuristic assessment of content [Ding-1999]. There are indications that online censors are starting to use machine learning techniques as well [Tang-2016].

There are typically three types of blacklists: Keyword, domain name, or Internet Protocol (IP) address. Keyword and domain name blocking take place at the application level (e.g. HTTP), whereas IP blocking tends to take place using routing data in TCP/IP headers. The mechanisms for building up these blacklists are varied. Censors can purchase from private industry "content control" software, such as SmartFilter, which allows filtering from broad categories that they would like to block, such as gambling or pornography. In these cases, these private services attempt to categorize every semi-questionable website as to allow for meta-tag blocking (similarly, they tune real-time content heuristic systems to map their assessments onto categories of objectionable content).

Countries that are more interested in retaining specific political control, a desire which requires swift and decisive action, often have ministries or organizations, such as the Ministry of Industry and Information Technology in China or the Ministry of Culture and Islamic Guidance in Iran, which maintain their own blacklists.

3. Technical Identification

3.1. Points of Control

Internet censorship, necessarily, takes place over a network. Network design gives censors a number of different points-of-control where they can identify the content they are interested in filtering. An important aspect of pervasive technical interception is the necessity to rely on software or hardware to intercept the content the censor is interested in. This requirement, the need to have the interception mechanism located somewhere, logically or physically, implicates various general points-of-control:

- *Internet Backbone:* If a censor controls the gateways into a region, they can filter undesirable traffic that is traveling into and out of the region by packet sniffing and port mirroring at the relevant exchange points. Censorship at this point of control is most effective at controlling the flow of information between a region and the rest of the Internet, but is ineffective at identifying content traveling between the users within a region.

- *Internet Service Providers:* Internet Service Providers are perhaps the most natural point of control. They have a benefit of being easily enumerable by a censor paired with the ability to identify the regional and international traffic of all their users. The censor's filtration mechanisms can be placed on an ISP via governmental mandates, ownership, or voluntary/coercive influence.

- *Institutions:* Private institutions such as corporations, schools, and cyber cafes can put filtration mechanisms in place. These mechanisms are occasionally at the request of a censor, but are more often implemented to help achieve institutional goals, such as to prevent the viewing of pornography on school computers.

- *Personal Devices:* Censors can mandate censorship software be installed on the device level. This has many disadvantages in terms of scalability, ease-of-circumvention, and operating system requirements. The emergence of mobile devices exacerbate these feasibility problems.

- *Services:* Application service providers can be pressured, coerced, or legally required to censor specific content or flows of data. Service providers naturally face incentives to maximize their potential customer base and potential service shutdowns or legal liability due to censorship efforts may seem much less attractive than potentially excluding content, users, or uses of their service.

- *Certificate Authorities:* Authorities that issue cryptographically secured resources can be a significant point of control. Certificate Authorities that issue certificates to domain holders for TLS/HTTPS or Regional/Local Internet Registries that issue Route Origination Authorizations to BGP operators can be forced to issue rogue certificates that may allow compromises in confidentiality guarantees - allowing censorship software to engage in identification and interference where not possible before - or integrity guarantees - allowing, for example, adversarial routing of traffic.

- *Content Distribution Networks (CDNs):* CDNs seek to collapse network topology in order to better locate content closer to the service's users in order to improve quality of service. These can be powerful points of control for censors, especially if the location of a CDN results in easier interference.

At all levels of the network hierarchy, the filtration mechanisms used to detect undesirable traffic are essentially the same: a censor sniffs transmitting packets and identifies undesirable content, and then uses a blocking or shaping mechanism to prevent or impair access. Identification of undesirable traffic can occur at the application, transport, or network layer of the IP stack. Censors are almost always concerned with web traffic, so the relevant protocols tend to be filtered in predictable ways. For example, a subversive image would always make it past a keyword filter, but the IP address of the site serving the image may be blacklisted when identified as a provider of undesirable content.

3.2. Application Layer

3.2.1. HTTP Request Header Identification

An HTTP header contains a lot of useful information for traffic identification; although "host" is the only required field in an HTTP request header (for HTTP/1.1 and later), an HTTP method field is necessary to do anything useful. As such, "method" and "host" are the two fields used most often for ubiquitous censorship. A censor can sniff traffic and identify a specific domain name (host) and usually a page name (GET /page) as well. This identification

technique is usually paired with TCP/IP header identification (see Section 3.3.1) for a more robust method.

Tradeoffs: Request Identification is a technically straight-forward identification method that can be easily implemented at the Backbone or ISP level. The hardware needed for this sort of identification is cheap and easy-to-acquire, making it desirable when budget and scope are a concern. HTTPS will encrypt the relevant request and response fields, so pairing with TCP/IP identification (see Section 3.3.1) is necessary for filtering of HTTPS. However, some countermeasures such as URL obfuscation [RSF-2005] can trivially defeat simple forms of HTTP Request Header Identification.

Empirical Examples: Studies exploring censorship mechanisms have found evidence of HTTP header/ URL filtering in many countries, including Bangladesh, Bahrain, China, India, Iran, Malaysia, Pakistan, Russia, Saudi Arabia, South Korea, Thailand, and Turkey [Verkamp-2012] [Nabi-2013] [Aryan-2012]. Commercial technologies such as the McAfee SmartFilter and NetSweeper are often purchased by censors [Dalek-2013]. These commercial technologies use a combination of HTTP Request Identification and TCP/IP Header Identification to filter specific URLs. Dalek et al. and Jones et al. identified the use of these products in the wild [Dalek-2013] [Jones-2014].

3.2.2. HTTP Response Header Identification

While HTTP Request Header Identification relies on the information contained in the HTTP request from client to server, response identification uses information sent in response by the server to client to identify undesirable content.

Tradeoffs: As with HTTP Request Header Identification, the techniques used to identify HTTP traffic are well-known, cheap, and relatively easy to implement, but is made useless by HTTPS, because the response in HTTPS is encrypted, including headers.

The response fields are also less helpful for identifying content than request fields, as "Server" could easily be identified using HTTP Request Header identification, and "Via" is rarely relevant. HTTP Response censorship mechanisms normally let the first n packets through while the mirrored traffic is being processed; this may allow some content through and the user may be able to detect that the censor is actively interfering with undesirable content.

Empirical Examples: In 2009, Jong Park et al. at the University of New Mexico demonstrated that the Great Firewall of China (GFW) has used this technique [Crandall-2010]. However, Jong Park et al. found

that the GFW discontinued this practice during the course of the study. Due to the overlap in HTTP response filtering and keyword filtering (see Section 3.2.3), it is likely that most censors rely on keyword filtering over TCP streams instead of HTTP response filtering.

3.2.3. Instrumenting Content Providers

In addition to censorship by the state, many governments pressure content providers to censor themselves. Due to the extensive reach of government censorship, we need to define content provider as any service that provides utility to users, including everything from web sites to locally installed programs. The defining factor of keyword identification by content providers is the choice of content providers to detect restricted terms on their platform. The terms to look for may be provided by the government or the content provider may be expected to come up with their own list.

Tradeoffs: By instrumenting content providers to identify restricted content, the censor can gain new information at the cost of political capital with the companies it forces or encourages to participate in censorship. For example, the censor can gain insight about the content of encrypted traffic by coercing web sites to identify restricted content, but this may drive away potential investment. Coercing content providers may encourage self-censorship, an additional advantage for censors. The tradeoffs for instrumenting content providers are highly dependent on the content provider and the requested assistance.

Empirical Examples: Researchers have discovered keyword identification by content providers on platforms ranging from instant messaging applications [Senft-2013] to search engines [Rushe-2015] [Cheng-2010] [Whittaker-2013] [BBC-2013] [Condliffe-2013]. To demonstrate the prevalence of this type of keyword identification, we look to search engine censorship.

Search engine censorship demonstrates keyword identification by content providers and can be regional or worldwide. Implementation is occasionally voluntary, but normally is based on laws and regulations of the country a search engine is operating in. The keyword blacklists are most likely maintained by the search engine provider. China is known to require search engine providers to "voluntarily" maintain search term blacklists to acquire/keep an Internet content provider (ICP) license [Cheng-2010]. It is clear these blacklists are maintained by each search engine provider based on the slight variations in the intercepted searches [Zhu-2011] [Whittaker-2013]. The United Kingdom has been pushing search engines to self-censor with the threat of litigation if they don’t do it

themselves: Google and Microsoft have agreed to block more than 100,000 queries in U.K. to help combat abuse [BBC-2013] [Condliffe-2013].

Depending on the output, search engine keyword identification may be difficult or easy to detect. In some cases specialized or blank results provide a trivial enumeration mechanism, but more subtle censorship can be difficult to detect. In February 2015, Microsoft's search engine, Bing, was accused of censoring Chinese content outside of China [Rushe-2015] because Bing returned different results for censored terms in Chinese and English. However, it is possible that censorship of the largest base of Chinese search users, China, biased Bing's results so that the more popular results in China (the uncensored results) were also more popular for Chinese speakers outside of China.

3.2.4. Deep Packet Inspection (DPI) Identification

Deep Packet Inspection has become computationally feasible as a censorship mechanism in recent years [Wagner-2009]. Unlike other techniques, DPI reassembles network flows to examine the application "data" section, as opposed to only the header, and is therefore often used for keyword identification. DPI also differs from other identification technologies because it can leverage additional packet and flow characteristics, i.e. packet sizes and timings, to identify content. To prevent substantial quality of service (QoS) impacts, DPI normally analyzes a copy of data while the original packets continue to be routed. Typically, the traffic is split using either a mirror switch or fiber splitter, and analyzed on a cluster of machines running Intrusion Detection Systems (IDS) configured for censorship.

Tradeoffs: DPI is one of the most expensive identification mechanisms and can have a large QoS impact [Porter-2010]. When used as a keyword filter for TCP flows, DPI systems can cause also major overblocking problems. Like other techniques, DPI is less useful against encrypted data, though DPI can leverage unencrypted elements of an encrypted data flow (e.g., the Server Name Indicator (SNI) sent in the clear for TLS) or statistical information about an encrypted flow (e.g., video takes more bandwidth than audio or textual forms of communication) to identify traffic.

Other kinds of information can be inferred by comparing certain unencrypted elements exchanged during TLS handshakes to similar data points from known sources. This practice, called TLS fingerprinting, allows a probabilistic identification of a party's operating system, browser, or application based on a comparison of the specific combinations of TLS version, ciphersuites, compression options, etc.

sent in the ClientHello message to similar signatures found in unencrypted traffic [Husak-2016].

Despite these problems, DPI is the most powerful identification method and is widely used in practice. The Great Firewall of China (GFW), the largest censorship system in the world, has used DPI to identify restricted content over HTTP and DNS and inject TCP RSTs and bad DNS responses, respectively, into connections [Crandall-2010] [Clayton-2006] [Anonymous-2014].

Empirical Examples: Several studies have found evidence of DPI being used to censor content and tools. Clayton et al. Crandal et al., Anonymous, and Khattak et al., all explored the GFW and Khattak et al. even probed the firewall to discover implementation details like how much state it stores [Crandall-2010] [Clayton-2006] [Anonymous-2014] [Khattak-2013]. The Tor project claims that China, Iran, Ethiopia, and others must have used DPI to block the obsf2 protocol [Wilde-2012]. Malaysia has been accused of using targeted DPI, paired with DDoS, to identify and subsequently knockout pro-opposition material [Wagstaff-2013]. It also seems likely that organizations not so worried about blocking content in real-time could use DPI to sort and categorically search gathered traffic using technologies such as NarusInsight [Hepting-2011].

3.2.4.1. Server Name Indication

In encrypted connections using Transport Layer Security (TLS), there may be servers that host multiple "virtual servers" at a give network address, and the client will need to specify in the (unencrypted) Client Hello message which domain name it seeks to connect to (so that the server can respond with the appropriate TLS certificate) using the Server Name Indication (SNI) TLS extension [RFC6066]. Since SNI is sent in the clear, censors and filtering software can use it as a basis for blocking, filtering, or impairment by dropping connections to domains that match prohibited content (e.g., bad.foo.example may be censored while good.foo.example is not) [Shbair-2015].

Domain fronting has been one popular way to avoid identification by censors [Fifield-2015]. To avoid identification by censors, applications using domain fronting put a different domain name in the SNI extension than the one encrypted by HTTPS. The visible SNI would indicate an unblocked domain, while the blocked domain remains hidden in the encrypted application header. Some encrypted messaging services relied on domain fronting to enable their provision in countries employing SNI-based filtering. These services used the cover provided by domains for which blocking at the domain level would be undesirable to hide their true domain names. However, the

companies holding the most popular domains have since reconfigured their software to prevent this practice. It may be possible to achieve similar results using potential future options to encrypt SNI in TLS 1.3.

Tradeoffs: Some clients do not send the SNI extension (e.g., clients that only support versions of SSL and not TLS) or will fall back to SSL if a TLS connection fails, rendering this method ineffective. In addition, this technique requires deep packet inspection techniques that can be computationally and infrastructurally expensive and improper configuration of an SNI-based block can result in significant overblocking, e.g., when a second-level domain like populardomain.example is inadvertently blocked. In the case of encrypted SNI, pressure to censor may transfer to other points of intervention, such as content and application providers.

Empirical Examples: While there are many examples of security firms that offer SNI-based filtering [Trustwave-2015] [Sophos-2015] [Shbair-2015], the government of South Korea was recently observed using SNI-based filtering. Cite to Gatlan https://www.bleepingcomputer.com/news/security/south-korea-is-censoring-the-internet-by-snooping-on-sni-traffic/

3.3. Transport Layer

3.3.1. Shallow Packet Inspection and TCP/IP Header Identification

Of the various shallow packet inspection methods, TCP/IP Header Identification is the most pervasive, reliable, and predictable type of identification. TCP/IP headers contain a few invaluable pieces of information that must be transparent for traffic to be successfully routed: destination and source IP address and port. Destination and Source IP are doubly useful, as not only does it allow a censor to block undesirable content via IP blacklisting, but also allows a censor to identify the IP of the user making the request. Port is useful for whitelisting certain applications.

Trade-offs: TCP/IP identification is popular due to its simplicity, availability, and robustness.

TCP/IP identification is trivial to implement, but is difficult to implement in backbone or ISP routers at scale, and is therefore typically implemented with DPI. Blacklisting an IP is equivalent to installing a /32 route on a router and due to limited flow table space, this cannot scale beyond a few thousand IPs at most. IP blocking is also relatively crude, leading to overblocking, and cannot deal with some services like Content Distribution Networks

(CDN), that host content at hundreds or thousands of IP addresses. Despite these limitations, IP blocking is extremely effective because the user needs to proxy their traffic through another destination to circumvent this type of identification.

Port-blocking is generally not useful because many types of content share the same port and it is possible for censored applications to change their port. For example, most HTTP traffic goes over port 80, so the censor cannot differentiate between restricted and allowed content solely on the basis of port. Port whitelisting is occasionally used, where a censor limits communication to approved ports, such as 80 for HTTP traffic and is most effective when used in conjunction with other identification mechanisms. For example, a censor could block the default HTTPS port, port 443, thereby forcing most users to fall back to HTTP.

3.3.2. Protocol Identification

Censors sometimes identify entire protocols to be blocked using a variety of traffic characteristics. For example, Iran impairs the performance of HTTPS traffic, a protocol that prevents further analysis, to encourage users to switch to HTTP, a protocol that they can analyze [Aryan-2012]. A simple protocol identification would be to recognize all TCP traffic over port 443 as HTTPS, but more sophisticated analysis of the statistical properties of payload data and flow behavior, would be more effective, even when port 443 is not used [Hjelmvik-2010] [Sandvine-2014].

If censors can detect circumvention tools, they can block them, so censors like China are extremely interested in identifying the protocols for censorship circumvention tools. In recent years, this has devolved into an arms race between censors and circumvention tool developers. As part of this arms race, China developed an extremely effective protocol identification technique that researchers call active probing or active scanning.

In active probing, the censor determines whether hosts are running a circumvention protocol by trying to initiate communication using the circumvention protocol. If the host and the censor successfully negotiate a connection, then the censor conclusively knows that host is running a circumvention tool. China has used active scanning to great effect to block Tor [Winter-2012].

Trade-offs: Protocol Identification necessarily only provides insight into the way information is traveling, and not the information itself.

Protocol identification is useful for detecting and blocking circumvention tools, like Tor, or traffic that is difficult to analyze, like VoIP or SSL, because the censor can assume that this traffic should be blocked. However, this can lead to over-blocking problems when used with popular protocols. These methods are expensive, both computationally and financially, due to the use of statistical analysis, and can be ineffective due to its imprecise nature.

Empirical Examples: Protocol identification can be easy to detect if it is conducted in real time and only a particular protocol is blocked, but some types of protocol identification, like active scanning, are much more difficult to detect. Protocol identification has been used by Iran to identify and throttle SSH traffic to make it unusable [Anonymous-2007] and by China to identify and block Tor relays [Winter-2012]. Protocol Identification has also been used for traffic management, such as the 2007 case where Comcast in the United States used RST injection to interrupt BitTorrent Traffic [Winter-2012].

4. Technical Interference

4.1. Application Layer

4.1.1. DNS Interference

There are a variety of mechanisms that censors can use to block or filter access to content by altering responses from the DNS [AFNIC-2013] [ICANN-SSAC-2012], including blocking the response, replying with an error message, or responding with an incorrect address.

"DNS mangling" is a network-level technique where an incorrect IP address is returned in response to a DNS query to a censored destination. An example of this is what some Chinese networks do (we are not aware of any other wide-scale uses of mangling). On those Chinese networks, every DNS request in transit is examined (presumably by network inspection technologies such as DPI) and, if it matches a censored domain, a false response is injected. End users can see this technique in action by simply sending DNS requests to any unused IP address in China (see example below). If it is not a censored name, there will be no response. If it is censored, an erroneous response will be returned. For example, using the command-line dig utility to query an unused IP address in China of 192.0.2.2 for the name "www.uncensored.example" compared with "www.censored.example" (censored at the time of writing), we get an erroneous IP address "198.51.100.0" as a response:

```
% dig +short +nodnssec @192.0.2.2 A www.uncensored.example
;; connection timed out; no servers could be reached

% dig +short +nodnssec @192.0.2.2 A www.censored.example
198.51.100.0
```

There are also cases of what is colloquially called "DNS lying", where a censor mandates that the DNS responses provided - by an operator of a recursive resolver such as an Internet access provider - be different than what authoritative resolvers would provide [Bortzmayer-2015].

DNS cache poisoning refers to a mechanism where a censor interferes with the response sent by an authoritative DNS resolver to a recursive resolver by responding more quickly than the authoritative resolver can respond with an alternative IP address [Halley-2008]. Cache poisoning occurs after the requested site's name servers resolve the request and attempt to forward the true IP back to the requesting device; on the return route the resolved IP is recursively cached by each DNS server that initially forwarded the request. During this caching process if an undesirable keyword is recognized, the resolved IP is "poisoned" and an alternative IP (or NXDOMAIN error) is returned more quickly than the upstream resolver can respond, causing an erroneous IP address to be cached (and potentially recursively so). The alternative IPs usually direct to a nonsense domain or a warning page. Alternatively, Iranian censorship appears to prevent the communication en-route, preventing a response from ever being sent [Aryan-2012].

Trade-offs: These forms of DNS interference require the censor to force a user to traverse a controlled DNS hierarchy (or intervening network on which the censor serves as a Active Pervasive Attacker [RFC7624] to rewrite DNS responses) for the mechanism to be effective. It can be circumvented by a technical savvy user that opts to use alternative DNS resolvers (such as the public DNS resolvers provided by Google, OpenDNS, Telcomix, or FDN) or Virtual Private Network technology. DNS mangling and cache poisoning also imply returning an incorrect IP to those attempting to resolve a domain name, but in some cases the destination may be technically accessible; over HTTP, for example, the user may have another method of obtaining the IP address of the desired site and may be able to access it if the site is configured to be the default server listening at this IP address. Target blocking has also been a problem, as occasionally users outside of the censors region will be directed through DNS servers or DNS-rewriting network equipment controlled by a censor, causing the request to fail. The ease of circumvention paired with the large risk of content blocking and target blocking make DNS interference a partial, difficult, and less

than ideal censorship mechanism. Additionally, the above mechanisms rely on DNSSEC not being deployed or DNSSEC validation not being active on the client or recursive resolver.

Empirical Examples: DNS interference, when properly implemented, is easy to identify based on the shortcomings identified above. Turkey relied on DNS interference for its country-wide block of websites such Twitter and YouTube for almost week in March of 2014 but the ease of circumvention resulted in an increase in the popularity of Twitter until Turkish ISPs implementing an IP blacklist to achieve the governmental mandate [Zmijewki-2014]. Ultimately, Turkish ISPs started hijacking all requests to Google and Level 3's international DNS resolvers [Zmijewki-2014]. DNS interference, when incorrectly implemented, has resulted in some of the largest "censorship disasters". In January 2014, China started directing all requests passing through the Great Fire Wall to a single domain, dongtaiwang.com, due to an improperly configured DNS poisoning attempt; this incident is thought to be the largest Internet-service outage in history [AFP-2014] [Anon-SIGCOMM12]. Countries such as China, Iran, Turkey, and the United States have discussed blocking entire TLDs as well, but only Iran has acted by blocking all Israeli (.il) domains [Albert-2011].

4.2. Transport Layer

4.2.1. Performance Degradation

While other interference techniques outlined in this section mostly focus on blocking or preventing access to content, it can be an effective censorship strategy in some cases to not entirely block access to a given destination, or service but instead degrade the performance of the relevant network connection. The resulting user experience for a site or service under performance degradation can be so bad that users opt to use a different site, service, or method of communication, or may not engage in communication at all if there are no alternatives. Traffic shaping techniques that rate-limit the bandwidth available to certain types of traffic is one example of a performance degradation.

Trade offs: While implementing a performance degradation will not always eliminate the ability of people to access a desire resource, it may force them to use other means of communication where censorship (or surveillance) is more easily accomplished.

Empirical Examples: Iran has been known to shape the bandwidth available to HTTPS traffic to encourage unencrypted HTTP traffic [Aryan-2012].

4.2.2. Packet Dropping

Packet dropping is a simple mechanism to prevent undesirable traffic. The censor identifies undesirable traffic and chooses to not properly forward any packets it sees associated with the traversing undesirable traffic instead of following a normal routing protocol. This can be paired with any of the previously described mechanisms so long as the censor knows the user must route traffic through a controlled router.

Trade offs: Packet Dropping is most successful when every traversing packet has transparent information linked to undesirable content, such as a Destination IP. One downside Packet Dropping suffers from is the necessity of blocking all content from otherwise allowable IPs based on a single subversive sub-domain; blogging services and github repositories are good examples. China famously dropped all github packets for three days based on a single repository hosting undesirable content [Anonymous-2013]. The need to inspect every traversing packet in close to real time also makes Packet Dropping somewhat challenging from a QoS perspective.

Empirical Examples: Packet Dropping is a very common form of technical interference and lends itself to accurate detection given the unique nature of the time-out requests it leaves in its wake. The Great Firewall of China has been observed using packet dropping as one of its primary mechanisms of technical censorship [Ensafi-2013]. Iran has also used Packet Dropping as the mechanisms for throttling SSH [Aryan-2012]. These are but two examples of a ubiquitous censorship practice.

4.2.3. RST Packet Injection

Packet injection, generally, refers to a man-in-the-middle (MITM) network interference technique that spoofs packets in an established traffic stream. RST packets are normally used to let one side of TCP connection know the other side has stopped sending information, and thus the receiver should close the connection. RST Packet Injection is a specific type of packet injection attack that is used to interrupt an established stream by sending RST packets to both sides of a TCP connection; as each receiver thinks the other has dropped the connection, the session is terminated.

Trade-offs: RST Packet Injection has a few advantages that make it extremely popular as a censorship technique. RST Packet Injection is an out-of-band interference mechanism, allowing the avoidance of the the QoS bottleneck one can encounter with inline techniques such as Packet Dropping. This out-of-band property allows a censor to inspect a copy of the information, usually mirrored by an optical

splitter, making it an ideal pairing for DPI and Protocol Identification [Weaver-2009] (this asynchronous version of a MITM is often called a Man-on-the-Side (MOTS)). RST Packet Injection also has the advantage of only requiring one of the two endpoints to accept the spoofed packet for the connection to be interrupted.

The difficult part of RST Packet Injection is spoofing "enough" correct information to ensure one end-point accepts a RST packet as legitimate; this generally implies a correct IP, port, and (TCP) sequence number. Sequence number is the hardest to get correct, as [RFC0793] specifies an RST Packet should be in-sequence to be accepted, although the RFC also recommends allowing in-window packets as "good enough". This in-window recommendation is important, as if it is implemented it allows for successful Blind RST Injection attacks [Netsec-2011]. When in-window sequencing is allowed, It is trivial to conduct a Blind RST Injection, a blind injection implies the censor doesn’t know any sensitive (encrypted) sequencing information about the TCP stream they are injecting into, they can simply enumerate the ~70000 possible windows; this is particularly useful for interrupting encrypted/obfuscated protocols such as SSH or Tor. RST Packet Injection relies on a stateful network, making it useless against UDP connections. RST Packet Injection is among the most popular censorship techniques used today given its versatile nature and effectiveness against all types of TCP traffic.

Empirical Examples: RST Packet Injection, as mentioned above, is most often paired with identification techniques that require splitting, such as DPI or Protocol Identification. In 2007, Comcast was accused of using RST Packet Injection to interrupt traffic it identified as BitTorrent [Schoen-2007], this later led to a US Federal Communications Commission ruling against Comcast [VonLohmann-2008]. China has also been known to use RST Packet Injection for censorship purposes. This interference is especially evident in the interruption of encrypted/obfuscated protocols, such as those used by Tor [Winter-2012].

4.3. Multi-layer and Non-layer

4.3.1. Distributed Denial of Service (DDoS)

Distributed Denial of Service attacks are a common attack mechanism used by "hacktivists" and malicious hackers, but censors have used DDoS in the past for a variety of reasons. There is a huge variety of DDoS attacks [Wikip-DoS], but on a high level two possible impacts tend to occur; a flood attack results in the service being unusable while resources are being spent to flood the service, a crash attack aims to crash the service so resources can be reallocated elsewhere without "releasing" the service.

Trade-offs: DDoS is an appealing mechanism when a censor would like to prevent all access to undesirable content, instead of only access in their region for a limited period of time, but this is really the only uniquely beneficial feature for DDoS as a censorship technique. The resources required to carry out a successful DDoS against major targets are computationally expensive, usually requiring renting or owning a malicious distributed platform such as a botnet, and imprecise. DDoS is an incredibly crude censorship technique, and appears to largely be used as a timely, easy-to-access mechanism for blocking undesirable content for a limited period of time.

Empirical Examples: In 2012 the U.K.'s GCHQ used DDoS to temporarily shutdown IRC chat rooms frequented by members of Anonymous using the Syn Flood DDoS method; Syn Flood exploits the handshake used by TCP to overload the victim server with so many requests that legitimate traffic becomes slow or impossible [Schone-2014] [CERT-2000]. Dissenting opinion websites are frequently victims of DDoS around politically sensitive events in Burma [Villeneuve-2011]. Controlling parties in Russia [Kravtsova-2012], Zimbabwe [Orion-2013], and Malaysia [Muncaster-2013] have been accused of using DDoS to interrupt opposition support and access during elections. In 2015, China launched a DDoS attack using a true MITM system collocated with the Great Firewall, dubbed "Great Cannon", that was able to inject JavaScript code into web visits to a Chinese search engine that commandeered those user agents to send DDoS traffic to various sites [Marczak-2015].

4.3.2. Network Disconnection or Adversarial Route Announcement

While it is perhaps the crudest of all censorship techniques, there is no more effective way of making sure undesirable information isn't allowed to propagate on the web than by shutting off the network. The network can be logically cut off in a region when a censoring body withdraws all of the Boarder Gateway Protocol (BGP) prefixes routing through the censor's country.

Trade-offs: The impact to a network disconnection in a region is huge and absolute; the censor pays for absolute control over digital information with all the benefits the Internet brings; this is never a long-term solution for any rational censor and is normally only used as a last resort in times of substantial unrest.

Empirical Examples: Network Disconnections tend to only happen in times of substantial unrest, largely due to the huge social, political, and economic impact such a move has. One of the first, highly covered occurrences was with the Junta in Myanmar employing Network Disconnection to help Junta forces quash a rebellion in 2007

[Dobie-2007]. China disconnected the network in the Xinjiang region during unrest in 2009 in an effort to prevent the protests from spreading to other regions [Heacock-2009]. The Arab Spring saw the the most frequent usage of Network Disconnection, with events in Egypt and Libya in 2011 [Cowie-2011] [Cowie-2011b], and Syria in 2012 [Thomson-2012]. Russia has indicated that it will attempt to disconnect all Russian networks from the global internet in April 2019 as part of a test of the nation's network independence. Reports also indicate that, as part of the test disconnect, Russian telecom firms must route all traffic to state-operated monitoring points. cite ZD Net https://www.zdnet.com/article/russia-to-disconnect-from-the-internet-as-part-of-a-planned-test/

5. Non-Technical Prescription

As the name implies, sometimes manpower is the easiest way to figure out which content to block. Manual Filtering differs from the common tactic of building up blacklists in that it doesn't necessarily target a specific IP or DNS, but instead removes or flags content. Given the imprecise nature of automatic filtering, manually sorting through content and flagging dissenting websites, blogs, articles and other media for filtration can be an effective technique. This filtration can occur on the Backbone/ISP level - China's army of monitors is a good example [BBC-2013b] - but more commonly manual filtering occurs on an institutional level. Internet Content Providers such as Google or Weibo, require a business license to operate in China. One of the prerequisites for a business license is an agreement to sign a "voluntary pledge" known as the "Public Pledge on Self-discipline for the Chinese Internet Industry". The failure to "energetically uphold" the pledged values can lead to the ICPs being held liable for the offending content by the Chinese government [BBC-2013b].

6. Non-Technical Interference

6.1. Self-Censorship

Self-censorship is one of the most interesting and effective types of censorship; a mix of Bentham's Panopticon, cultural manipulation, intelligence gathering, and meatspace enforcement. Simply put, self-censorship is when a censor creates an atmosphere where users censor themselves. This can be achieved through controlling information, intimidating would-be dissidents, swaying public thought, and creating apathy. Self-censorship is difficult to document, as when it is implemented effectively the only noticeable tracing is a lack of undesirable content; instead one must look at the tools and techniques used by censors to encourage self-censorship. Controlling Information relies on traditional censorship techniques, or by

forcing all users to connect through an intranet, such as in North Korea. Intimidation is often achieved through allowing Internet users to post "whatever they want," but arresting those who post about dissenting views, this technique is incredibly common [Calamur-2013] [AP-2012] [Hopkins-2011] [Guardian-2014] [Johnson-2010]. A good example of swaying public thought is China's "50-Cent Party," reported to be composed of somewhere between 20,000 [Bristow-2013] and 300,000 [Fareed-2008] contributors who are paid to "guide public thought" on local and regional issues as directed by the Ministry of Culture. Creating apathy can be a side-effect of successfully controlling information over time and is ideal for a censorship regime [Gao-2014].

6.2. Domain Name Reallocation

Because domain names are resolved recursively, if a root name server reassigns or delists a domain, all other DNS servers will be unable to properly forward and cache the site. Domain name registration is only really a risk where undesirable content is hosted on TLD controlled by the censoring country, such as .cn or .ru [Anderson-2011] or where legal processes in countries like the United States result in domain name seizures and/or DNS redirection by the government [Kopel-2013].

6.3. Server Takedown

Servers must have a physical location somewhere in the world. If undesirable content is hosted in the censoring country the servers can be physically seized or the hosting provider can be required to prevent access [Anderson-2011].

6.4. Notice and Takedown

In some countries, legal mechanisms exist where an individual can issue a legal request to a content host that requires the host to take down content. Examples include the voluntary systems employed by companies like Google to comply with "Right to be Forgotten" policies in the European Union [Google-RTBF] and the copyright-oriented notice and takedown regime of the United States Digital Millennium Copyright Act (DMCA) Section 512 [DMLP-512].

7. Contributors

This document benefited from discussions with Stephane Bortzmeyer, Nick Feamster, and Martin Nilsson.

8. Informative References

[AFNIC-2013]
AFNIC, "Report of the AFNIC Scientific Council: Consequences of DNS-based Internet filtering", 2013, <http://www.afnic.fr/medias/documents/conseilscientifique/SC-consequences-of-DNS-based-Internet-filtering.pdf>.

[AFP-2014]
AFP, "China Has Massive Internet Breakdown Reportedly Caused By Their Own Censoring Tools", 2014, <http://www.businessinsider.com/chinas-internet-breakdown-reportedly-caused-by-censoring-tools-2014-1>.

[Albert-2011]
Albert, K., "DNS Tampering and the new ICANN gTLD Rules", 2011, <https://opennet.net/blog/2011/06/dns-tampering-and-new-icann-gtld-rules>.

[Anderson-2011]
Anderson, R. and S. Murdoch, "Access Denied: Tools and Technology of Internet Filtering", 2011, <http://access.opennet.net/wp-content/uploads/2011/12/accessdenied-chapter-3.pdf>.

[Anon-SIGCOMM12]
Anonymous, "The Collateral Damage of Internet Censorship by DNS Injection", 2012, <http://www.sigcomm.org/sites/default/files/ccr/papers/2012/July/2317307-2317311.pdf>.

[Anonymous-2007]
Anonymous, "How to Bypass Comcast’s Bittorrent Throttling", 2012, <https://torrentfreak.com/how-to-bypass-comcast-bittorrent-throttling-071021>.

[Anonymous-2013]
Anonymous, "GitHub blocked in China - how it happened, how to get around it, and where it will take us", 2013, <https://en.greatfire.org/blog/2013/jan/github-blocked-china-how-it-happened-how-get-around-it-and-where-it-will-take-us>.

[Anonymous-2014]
Anonymous, "Towards a Comprehensive Picture of the Great Firewall’s DNS Censorship", 2014, <https://www.usenix.org/system/files/conference/foci14/foci14-anonymous.pdf>.

[AP-2012] Associated Press, "Sattar Beheshit, Iranian Blogger, Was Beaten In Prison According To Prosecutor", 2012, <http://www.huffingtonpost.com/2012/12/03/sattar-beheshit-iran_n_2233125.html>.

[Aryan-2012] Aryan, S., Aryan, H., and J. Halderman, "Internet Censorship in Iran: A First Look", 2012, <https://jhalderm.com/pub/papers/iran-foci13.pdf>.

[BBC-2013] BBC News, "Google and Microsoft agree steps to block abuse images", 2013, <http://www.bbc.com/news/uk-24980765>.

[BBC-2013b] BBC, "China employs two million microblog monitors state media say", 2013, <http://www.bbc.com/news/world-asia-china-2439695>.

[Bortzmayer-2015] Bortzmayer, S., "DNS Censorship (DNS Lies) As Seen By RIPE Atlas", 2015, <https://labs.ripe.net/Members/stephane_bortzmeyer/dns-censorship-dns-lies-seen-by-atlas-probes>.

[Bristow-2013] Bristow, M., "China's internet 'spin doctors'", 2013, <http://news.bbc.co.uk/2/hi/asia-pacific/7783640.stm>.

[Calamur-2013] Calamur, K., "Prominent Egyptian Blogger Arrested", 2013, <http://www.npr.org/blogs/thetwo-way/2013/11/29/247820503/prominent-egyptian-blogger-arrested>.

[CERT-2000] CERT, "TCP SYN Flooding and IP Spoofing Attacks", 2000, <http://www.cert.org/historical/advisories/CA-1996-21.cfm>.

[Cheng-2010] Cheng, J., "Google stops Hong Kong auto-redirect as China plays hardball", 2010, <http://arstechnica.com/tech-policy/2010/06/google-tweaks-china-to-hong-kong-redirect-same-results/>.

[Clayton-2006] Clayton, R., "Ignoring the Great Firewall of China", 2006, <http://link.springer.com/chapter/10.1007/11957454_2>.

[Condliffe-2013]
Condliffe, J., "Google Announces Massive New Restrictions on Child Abuse Search Terms", 2013, <http://gizmodo.com/google-announces-massive-new-restrictions-on-child-abus-1466539163>.

[Cowie-2011]
Cowie, J., "Egypt Leaves the Internet", 2011, <http://www.renesys.com/2011/01/egypt-leaves-the-internet/>.

[Cowie-2011b]
Cowie, J., "Libyan Disconnect", 2011, <http://www.renesys.com/2011/02/libyan-disconnect-1/>.

[Crandall-2010]
Crandall, J., "Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China", 2010, <http://www.cs.unm.edu/~crandall/icdcs2010.pdf>.

[Dalek-2013]
Dalek, J., "A Method for Identifying and Confirming the Use of URL Filtering Products for Censorship", 2013, <http://www.cs.stonybrook.edu/~phillipa/papers/imc112s-dalek.pdf>.

[Ding-1999]
Ding, C., Chi, C., Deng, J., and C. Dong, "Centralized Content-Based Web Filtering and Blocking: How Far Can It Go?", 1999, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.3302&rep=rep1&type=pdf>.

[DMLP-512]
Digital Media Law Project, "Protecting Yourself Against Copyright Claims Based on User Content", 2012, <http://www.dmlp.org/legal-guide/protecting-yourself-against-copyright-claims-based-user-content>.

[Dobie-2007]
Dobie, M., "Junta tightens media screw", 2007, <http://news.bbc.co.uk/2/hi/asia-pacific/7016238.stm>.

[Ensafi-2013]
Ensafi, R., "Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels", 2013, <http://arxiv.org/pdf/1312.5739v1.pdf>.

[Fareed-2008]
Fareed, M., "China joins a turf war", 2008, <http://www.theguardian.com/media/2008/sep/22/chinathemedia.marketingandpr>.

[Fifield-2015]
Fifield, D., Lan, C., Hynes, R., Wegmann, P., and V. Paxson, "Blocking-resistant communication through domain fronting", 2015, <https://petsymposium.org/2015/papers/03_Fifield.pdf>.

[Gao-2014]
Gao, H., "Tiananmen, Forgotten", 2014, <http://www.nytimes.com/2014/06/04/opinion/tiananmen-forgotten.html>.

[Glanville-2008]
Glanville, J., "The Big Business of Net Censorship", 2008, <http://www.theguardian.com/commentisfree/2008/nov/17/censorship-internet>.

[Google-RTBF]
Google, Inc., "Search removal request under data protection law in Europe", 2015, <https://support.google.com/legal/contact/lr_eudpa?product=websearch>.

[Guardian-2014]
The Gaurdian, "Chinese blogger jailed under crackdown on 'internet rumours'", 2014, <http://www.theguardian.com/world/2014/apr/17/chinese-blogger-jailed-crackdown-internet-rumours-qin-zhihui>.

[Halley-2008]
Halley, B., "How DNS cache poisoning works", 2014, <https://www.networkworld.com/article/2277316/tech-primers/tech-primers-how-dns-cache-poisoning-works.html>.

[Heacock-2009]
Heacock, R., "China Shuts Down Internet in Xinjiang Region After Riots", 2009, <https://opennet.net/blog/2009/07/china-shuts-down-internet-xinjiang-region-after-riots>.

[Hepting-2011]
Electronic Frontier Foundation, "Hepting vs. AT&T", 2011, <https://www.eff.org/cases/hepting>.

[Hjelmvik-2010]
Hjelmvik, E., "Breaking and Improving Protocol Obfuscation", 2010, <https://www.iis.se/docs/hjelmvik_breaking.pdf>.

[Hopkins-2011]
Hopkins, C., "Communications Blocked in Libya, Qatari Blogger Arrested: This Week in Online Tyranny", 2011, <http://readwrite.com/2011/03/03/communications_blocked_in_libya_this_week_in_onlin>.

[Husak-2016]
Husak, M., Cermak, M., Jirsik, T., and P. Celeda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting", 2016, <https://link.springer.com/article/10.1186/s13635-016-0030-7>.

[ICANN-SSAC-2012]
ICANN Security and Stability Advisory Committee (SSAC), "SAC 056: SSAC Advisory on Impacts of Content Blocking via the Domain Name System", 2012, <https://www.icann.org/en/system/files/files/sac-056-en.pdf>.

[Johnson-2010]
Johnson, L., "Torture feared in arrest of Iraqi blogger", 2011, <http://seattlepostglobe.org/2010/02/05/torture-feared-in-arrest-of-iraqi-blogger/>.

[Jones-2014]
Jones, B., "Automated Detection and Fingerprinting of Censorship Block Pages", 2014, <http://conferences2.sigcomm.org/imc/2014/papers/p299.pdf>.

[Khattak-2013]
Khattak, S., "Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion", 2013, <http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/12389-foci13-khattak.pdf>.

[Kopel-2013]
Kopel, K., "Operation Seizing Our Sites: How the Federal Government is Taking Domain Names Without Prior Notice", 2013, <http://dx.doi.org/doi:10.15779/Z384Q3M>.

[Kravtsova-2012]
Kravtsova, Y., "Cyberattacks Disrupt Opposition's Election", 2012, <http://www.themoscowtimes.com/news/article/cyberattacks-disrupt-oppositions-election/470119.html>.

[Marczak-2015]
Marczak, B., Weaver, N., Dalek, J., Ensafi, R., Fifield, D., McKune, S., Rey, A., Scott-Railton, J., Deibert, R., and V. Paxson, "An Analysis of China's "Great Cannon"", 2015, <https://www.usenix.org/system/files/conference/foci15/foci15-paper-marczak.pdf>.

[Muncaster-2013]
Muncaster, P., "Malaysian election sparks web blocking/DDoS claims", 2013, <http://www.theregister.co.uk/2013/05/09/malaysia_fraud_elections_ddos_web_blocking/>.

[Nabi-2013]
Nabi, Z., "The Anatomy of Web Censorship in Pakistan", 2013, <http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/12387-foci13-nabi.pdf>.

[Netsec-2011]
n3t2.3c, "TCP-RST Injection", 2011, <https://nets.ec/TCP-RST_Injection>.

[Orion-2013]
Orion, E., "Zimbabwe election hit by hacking and DDoS attacks", 2013, <http://www.theinquirer.net/inquirer/news/2287433/zimbabwe-election-hit-by-hacking-and-ddos-attacks>.

[Porter-2010]
Porter, T., "The Perils of Deep Packet Inspection", 2010, <http://www.symantec.com/connect/articles/perils-deep-packet-inspection>.

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

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <https://www.rfc-editor.org/info/rfc7624>.

[RFC7754] Barnes, R., Cooper, A., Kolkman, O., Thaler, D., and E. Nordmark, "Technical Considerations for Internet Service Blocking and Filtering", RFC 7754, DOI 10.17487/RFC7754, March 2016, <https://www.rfc-editor.org/info/rfc7754>.

[RSF-2005] Reporters Sans Frontieres, "Technical ways to get around censorship", 2005, <http://archives.rsf.org/print-blogs.php3?id_article=15013>.

[Rushe-2015] Rushe, D., "Bing censoring Chinese language search results for users in the US", 2013, <http://www.theguardian.com/technology/2014/feb/11/bing-censors-chinese-language-search-results>.

[Sandvine-2014] Sandvine, "Technology Showcase on Traffic Classification: Why Measurements and Freeform Policy Matter", 2014, <https://www.sandvine.com/downloads/general/technology/sandvine-technology-showcases/sandvine-technology-showcase-traffic-classification.pdf>.

[Schoen-2007] Schoen, S., "EFF tests agree with AP: Comcast is forging packets to interfere with user traffic", 2007, <https://www.eff.org/deeplinks/2007/10/eff-tests-agree-ap-comcast-forging-packets-to-interfere>.

[Schone-2014] Schone, M., Esposito, R., Cole, M., and G. Greenwald, "Snowden Docs Show UK Spies Attacked Anonymous, Hackers", 2014, <http://www.nbcnews.com/feature/edward-snowden-interview/exclusive-snowden-docs-show-uk-spies-attacked-anonymous-hackers-n21361>.

[Senft-2013]
Senft, A., "Asia Chats: Analyzing Information Controls and Privacy in Asian Messaging Applications", 2013, <https://citizenlab.org/2013/11/asia-chats-analyzing-information-controls-privacy-asian-messaging-applications/>.

[Shbair-2015]
Shbair, W., Cholez, T., Goichot, A., and I. Chrisment, "Efficiently Bypassing SNI-based HTTPS Filtering", 2015, <https://hal.inria.fr/hal-01202712/document>.

[Sophos-2015]
Sophos, "Understanding Sophos Web Filtering", 2015, <https://www.sophos.com/en-us/support/knowledgebase/115865.aspx>.

[Tang-2016]
Tang, C., "In-depth analysis of the Great Firewall of China", 2016, <https://www.cs.tufts.edu/comp/116/archive/fall2016/ctang.pdf>.

[Thomson-2012]
Thomson, I., "Syria Cuts off Internet and Mobile Communication", 2012, <http://www.theregister.co.uk/2012/11/29/syria_internet_blackout/>.

[Trustwave-2015]
Trustwave, "Filter: SNI extension feature and HTTPS blocking", 2015, <https://www3.trustwave.com/software/8e6/hlp/r3000/files/1system_filter.html>.

[Verkamp-2012]
Verkamp, J. and M. Gupta, "Inferring Mechanics of Web Censorship Around the World", 2012, <https://www.usenix.org/system/files/conference/foci12/foci12-final1.pdf>.

[Villeneuve-2011]
Villeneuve, N., "Open Access: Chapter 8, Control and Resistance, Attacks on Burmese Opposition Media", 2011, <http://access.opennet.net/wp-content/uploads/2011/12/accesscontested-chapter-08.pdf>.

[VonLohmann-2008]
VonLohmann, F., "FCC Rules Against Comcast for BitTorrent Blocking", 2008, <https://www.eff.org/deeplinks/2008/08/fcc-rules-against-comcast-bit-torrent-blocking>.

[Wagner-2009]
Wagner, B., "Deep Packet Inspection and Internet Censorship: International Convergence on an 'Integrated Technology of Control'", 2009, <http://advocacy.globalvoicesonline.org/wp-content/uploads/2009/06/deeppacketinspectionandinternet-censorship2.pdf>.

[Wagstaff-2013]
Wagstaff, J., "In Malaysia, online election battles take a nasty turn", 2013, <http://www.reuters.com/article/2013/05/04/uk-malaysia-election-online-idUKBRE94309G20130504>.

[Weaver-2009]
Weaver, N., Sommer, R., and V. Paxson, "Detecting Forged TCP Packets", 2009, <http://www.icir.org/vern/papers/reset-injection.ndss09.pdf>.

[Whittaker-2013]
Whittaker, Z., "1,168 keywords Skype uses to censor, monitor its Chinese users", 2013, <http://www.zdnet.com/1168-keywords-skype-uses-to-censor-monitor-its-chinese-users-7000012328/>.

[Wikip-DoS]
Wikipedia, "Denial of Service Attacks", 2016, <https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=710558258>.

[Wilde-2012]
Wilde, T., "Knock Knock Knockin' on Bridges Doors", 2012, <https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors>.

[Winter-2012]
Winter, P., "How China is Blocking Tor", 2012, <http://arxiv.org/pdf/1204.0447v1.pdf>.

[Zhu-2011]
Zhu, T., "An Analysis of Chinese Search Engine Filtering", 2011, <http://arxiv.org/ftp/arxiv/papers/1107/1107.3794.pdf>.

[Zmijewki-2014]
Zmijewki, E., "Turkish Internet Censorship Takes a New Turn", 2014, <http://www.renesys.com/2014/03/turkish-internet-censorship/>.

Authors' Addresses

Joseph Lorenzo Hall
CDT

Email: joe@cdt.org

Michael D. Aaron
CU Boulder

Email: michael.aaron@colorado.edu

Stan Adams
CDT

Email: sadams@cdt.org

Ben Jones
Princeton

Email: bj6@cs.princeton.edu

Nick Feamster
Princeton

Email: feamster@cs.princeton.edu

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

B. Wiley
Operator Foundation
D. Oliver
Guardian Project

July 08, 2019

Enabling Network Traffic Obfuscation - Pluggable Transports
draft-oliver-pluggable-transports-00

Abstract

Pluggable Transports (PTs) are a mechanism enabling the rapid development and deployment of network traffic obfuscation techniques used to circumvent surveillance and censorship. This specification does not define or limit the techniques themselves, but rather focuses on the startup, shutdown, and inter-process communication mechanisms required to make these technologies interoperable with applications.

This document is based heavily on [PT2.1].

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Conventions and Definitions 3
3. Background 3
4. Architecture Overview 4
5. Specification 6
5.1. Pluggable Transport Naming 6
5.2. Transports API Interface 6
5.2.1. Goals for interface design 6
5.2.2. Abstract Interfaces 7
6. Adapters 8
6.1. API to IPC Adapter 8
6.2. PT 1.0 Compatibility 9
6.3. Cross-language Linking 9
6.3.1. Using the Dispatcher IPC Interface In-process 9
6.4. Anonymity Considerations 10
7. References 10
7.1. Normative References 10
7.2. Informative References 10
Acknowledgments 12
Authors' Addresses 12

1. Introduction

The increased interest in network traffic obfuscation technologies mirrors the increase in usage of Deep Packet Inspection (DPI) to actively monitor the content of application data in addition to that data's routing information. Deep Packet Inspection inspects each packet based on the header of its request and the data it carries. It can identify the type of protocol the connection is using even if it was encrypted. DPI is not a mechanism to decrypt what is inside packets but to identify the 'protocol' or the application it represents.

Deep packet inspection has become the prime tool of censors and surveillance entities who block, log, and/or traffic-shape access to sites and services they deem undesirable.
As deep packet inspection has become more routine, the sophistication of monitoring has increased to include active probing that fingerprints and classifies application protocols. Thus, even as conventional care in application design has improved (via encryption

and other protocol design features that encourage privacy), network traffic is still under attack.

The techniques of network monitoring are changing and improving day by day. The development of traffic obfuscation techniques that foil these efforts is slowed by the lack of common agreement on how these techniques are invoked, made easily interoperable with applications, and deployed quickly. This specification addresses those issues.

This specification describes a method for decoupling protocol-level obfuscation from an application's client/server code, in a manner that promotes rapid development of obfuscation/circumvention tools and promotes reuse across privacy tools such as VPNs and secure proxies.

This decoupling is accomplished by utilizing helper code, either in-process through a language-specific API or in a separate sub-processes, that implements the necessary forward/reverse proxy services that handle the censorship circumvention, with a well defined and standardized configuration and management interface. Any application code that implements the interfaces as specified in this document will be able to use all specification-compliant Pluggable Transports.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Background

We define an Internet censor as any network intermediary that seeks to block, divert or traffic-manage Internet network connections for the purpose of eliminating, frustrating and/or logging access to Internet resources that have been deemed (by the censor) to be undesirable (either on a temporary or permanent basis). A variety of techniques are commonly applied by Internet censors to block such traffic. These include:

1. DNS Blocking

2. IP Blocking

3. Port Blocking

These techniques are applicable to a connection's metadata (IP routing information) and do not require inspecting the connection's datastream.

DPI, in contrast, actually looks at the connection's datastream - often, specifically, the initial data elements in the stream (or within blocks of the stream). These elements of the stream can contain clues as to the application-level protocol employed, even when the data itself is encrypted. Through observation over time, these clues ("fingerprints") can be learned by the censor and (along with the routing information) used to block or divert targeted traffic.

A defense against this type of active probing is traffic obfuscation - disguising the application data itself in a manner that is less-easily fingerprinted. However, in early experiments it quickly became clear that repeated use of the same obfuscation technique would, itself, be learned. Methods were developed by which a single obfuscation technique could transform on its own TODO: cite FTE proxy, ScrambleSuit, Dust. This approach proved expensive in terms of computational load. Interest gathered in solving this problem and as more ideas arose so to did the need for a mechanism supporting rapid deploying of new obfuscation techniques.

While intense work on network traffic obfuscation commenced initially and continues within the Tor Project (and across a wider set of external parties using Tor as a vehicle for research), vendors of other privacy-enhancing software (such as VPNs) quickly found their products also foiled by DPI. Thus, it becomes important to see transport pluggability as a mechanism implemented in a manner independent of a specific product or service. The notion of "Pluggable Transports" (PT) was born from these requirements.

4. Architecture Overview

The PT Server software exposes a public proxy that accepts connections from PT Clients. The PT Client transforms the traffic before it hits the public Internet and the PT Server reverses this transformation before passing the traffic on to its next destination. The PT Server directly forwards this data to the Server App, but the Server App itself may itself be a proxy server and expect the forwarded traffic it receives to conform to a proxy communication protocol such as SOCKS or TURN. There is also an optional lightweight protocol to facilitate communicating connection metadata that would otherwise be lost such as the source IP address and port EXTORPORT.

When using an in-process, language-specific API ("Transport API Interface") to integrate PTs into an application on both client and server, the PT Client Library is integrated directly into the Client App and the PT Server Library is integrated directly into the Server App. The Client App and Server App communicate through socket-like APIs, with all communication between them going through the PT library, which only sends transformed traffic over the public Internet.

```
+------------+                     +---------------------------+
| Client App +-- Socket-like API --+ PT Client (Library)       +--+
+------------+                     +---------------------------+  |
                                                                   |
             Public Internet (Obfuscated/Transformed traffic) ==>  |
                                                                   |
+------------+                       +-------------------------+  |
| Server App +--  Socket-like API  --+ PT Server (Library)     +--+
+------------+                       +-------------------------+
```

Figure 1. API Architecture Overview

When using the transports as a separate process on both client and server, the Dispatcher IPC Interface is used. On the client device, the PT Client software exposes a local proxy to the client application, and transforms traffic before forwarding it to the PT Server. The PT Dispatcher can be configured to provide different proxy types, supporting proxying of both TCP and UDP traffic.

```
+------------+                     +---------------------------+
| Client App +---- Local Proxy ----+ PT Client (Dispatcher)    +--+
+------------+                     +---+--------------------+---+  |
                                       | PT Client Library |        |
                                       +--------------------+        |
                                                                     |
            Public Internet (Transformed/Proxied traffic) =====>     |
                                                                     |
+------------+                     +---------------------------+  |
| Server App +---- Local Proxy ----+ PT Server (Dispatcher )   +--+
+------------+                     +---+--------------------+---+
                                         |PT Server (Library)|
                                         +--------------------+
```

Figure 2. IPC Architecture Overview

A PT client-server setup may also mix and match interfaces, using Dispatcher IPC on one end of the connection and the Transport API on the other, as below (or vice-versa):

```
+------------+                     +---------------------------+
| Client App +---- Local Proxy ----+ PT Dispatcher Client      +-+
+------------+                     +---+-------------------+---+ |
                                                                  |
                                                                  |
+------------+                     +--------------------------+  |
| Server App +-- Socket-like API --+ PT Server (Library )     +--+
+------------+                     +--------------------------+
```

Figure 3. Mixed IPC and Transport API example

Each invocation of a PT MUST be either a client OR a server.

PT dispatchers MAY support any of the following proxy modes: PT 1.0 with SOCKS4, PT 1.0 with SOCKS5, or any of the PT 2.1 modes: transparent TCP, transparent UDP, or STUN-aware UDP. Clients SHOULD prefer PT 2.1 over PT 1.0.

5. Specification

5.1. Pluggable Transport Naming

Pluggable Transport names serve as unique identifiers, and every PT MUST have a unique name. PT names MUST be valid C identifiers, which means that PT names MUST begin with a letter or underscore, and the remaining characters MUST be ASCII letters, numbers or underscores. No length limit is imposed. PT names MUST therefore satisfy the regular expression [a-zA-Z_][a-zA-Z0-9_]*.

5.2. Transports API Interface

5.2.1. Goals for interface design

The goal for the interface design is to achieve the following properties:

- Transport implementers have to do the minimum amount of work in addition to implementing the core transform logic.

- Transport users have to do the minimum amount of work to add PT support to code that uses standard networking primitives from the language or platform.

- Transports require an explicit destination address to be specified. However, this can be either an explicit PT server destination with the Server App is already known implicitly, or an explicit Server App destination with the PT server destination already known implicity.

- Transports may or may not generate, send, receive, store, and/or update persistent or ephemeral state.

- Transports that do not need persistence or negotiation can interact with the application through the simplest possible interface

- Transports that do need persistence or negotiation can rely on the application to provide it through the specified interface, so the transport does not need to implement persistence or negotiation internally.

- Applications should be able to use a PT Client implementation to establish several independent transport connections with different parameters, with a minimum of complexity and latency.

- The interface in each language should be idiomatic and performant, including reproducing blocking behavior and interaction with nonblocking IO subsystems when possible.

5.2.2. Abstract Interfaces

This section presents high-level pseudocode descriptions of the interfaces exposed by different types of transport components. Implementations for different languages should provide equivalent functionality, but should use the idioms for each language, mimicking the existing networking libraries.

5.2.2.1. Transport

- Transport takes a transport configuration and provides a Client Factory and a Server Factory.

- Transports may provide additional language-specific configuration methods.

- The only way to obtain Client Factories and Server Factories is from the Transport.

- The Server Factory of the Transport can fail if the Transport does not provide a server-side implementation. However, most transports provide both a client and server implementation.

- The transport configuration is specific to each Transport. Using a Transport requires knowing the correct parameters to initialize that Transport.

5.2.2.1.1. Client Factory

- Client Factory takes the connection settings and produces a Connection to that server.

- The connection settings are specific to each transport. Some transports will also require an argument indicating the destination endpoint. Producing a Connection may fail if the server is unreachable or if the transport configuration was incorrect.

5.2.2.1.2. Server Factory

- Server Factory takes the address on which the PT server should listen for incoming client connections and produces a Listener for that address

5.2.2.1.3. Listener

- Listener produces a stream of Connections

- New Connections are available whenever an incoming network connection from the PT client has been established. The language-specific API can adopt either a blocking or non-blocking API for accepting new connections, depending on what is idiomatic for the language. 3.2.2.2. Connection

- Connection provides an API similar to the environment's native socket type

- Connection is what is used to read and write data over the transport connection

- The transport-specific logic for obfuscating network traffic is implemented inside the Connection.

6. Adapters

This section covers the various different ways that the Pluggable Transport interfaces (both API and IPC) can be adapted to different use cases.

6.1. API to IPC Adapter

When an application and the transports it uses are written in the same language, either the Transports API or Dispatcher IPC can be used. When they are in different languages, they must communicate through the Dispatcher IPC interface. For maximum flexibility and to

minimize duplication of effort across languages, dispatcher can be implemented by wrapping transport implementations that implement the Transports API. For an example of this approach, see the Shapeshifter Dispatcher [PT2-DISPATCHER], which wraps transports implementing the Transports API in the Go language and provides a Dispatcher IPC interface to use them from other languages.

6.2. PT 1.0 Compatibility

The only interface defined in the PT 1.0 specification is an IPC interface. No standard API is defined. Therefore, PT 1.0 compatibility refers to compatibility between applications and transports where one side conforms to the PT 1.0 specification and the other conforms to the PT 2.1 specification. Fortunately, an adapter is not needed in this case as both the PT 1.0 and PT 2.1 specifications allow for version negotiation. The TOR_PT_MANAGED_TRANSPORT_VER environment variable or -ptversion command line flag is used by the application to specify a list of supported versions, for instance "1.0,2.1". The PT provider responds with the VERSION command on stdout in order to specify which version is supported by the PT provider, for instance "VERSION 2.1". Since the application can specify a list of supported versions, the PT provider can respond dynamically, supporting PT 1.0 when required and automatically upgrading to a PT 2.1 implementation when that is an available option. It is up to applications whether they want to support PT 2.1 exclusively or maintain backwards compatibility with PT 1.0 implementations.

6.3. Cross-language Linking

If two languages are compatible via cross-language linking, then a suitable adapter can be written that wraps the implementation of the Transports API in one language with an API for a compatible language. For example, on Android the Go implementation of the Transports API is wrapped in a Java API to create Java language bindings without the need for a native Java implementation or use of Dispatcher IPC.

6.3.1. Using the Dispatcher IPC Interface In-process

When using a transport that exposes the Dispatcher IPC interface, it may be more convenient to run the transport in a separate thread but in the same process as the application. Packets can still be routed through the transport’s SOCKS5 or TURN port on localhost. However, it may be inconvenient or impossible to use STDIN and STDOUT for communication between these two threads. Therefore, in some languages it may be appropriate to produce an "inter-thread interface" that reproduces the Dispatcher IPC interface’s semantics, but replaces STDIN and STDOUT with language-native function-call and

event primitives. This is the approach used by OnionBrowser [ONION_BROWSER], the Tor implementation on iOS. This approach is used because Tor uses the Dispatcher IPC mechanism to talk to the transports instead of the Transports API. However, iOS does not allow for applications to have multiple processes. Therefore, an in-process Dispatcher IPC approach must be used instead of traditional separate process Dispatcher IPC. An alternative would be to use the Transports API directly instead of Dispatcher IPC.

6.4. Anonymity Considerations

When designing and implementing a Pluggable Transport, care should be taken to preserve the privacy of clients and to avoid leaking personally identifying information. Examples of client related considerations are:

- Not logging client IP addresses to disk.

- Not leaking DNS addresses except when necessary.

- Ensuring that "TOR_PT_PROXY"'s "fail closed" behavior is implemented correctly.

Additionally, certain obfuscation mechanisms rely on information such as the server IP address and port being confidential, so clients also need to take care to preserve server side information confidential when applicable.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[ONION_BROWSER] "Onion Browser", 2019, <https://github.com/OnionBrowser/OnionBrowser>.

[PT2-DISPATCHER]
Wiley, B., "Shapeshifter Dispatcher", 2018, <https://github.com/OperatorFoundation/shapeshifter-dispatcher>.

[PT2.1] Wiley, B., "Pluggable Transport Base Specification", 2018, <https://github.com/Pluggable-Transports/Pluggable-Transports-spec/blob/master/releases/PTSpecV2.1Draft1/Pluggable%20Transport%20Specification%20v2.1%20-%20Base%20Specification%20v2.1%2C%20Draft%201.pdf>.

Acknowledgments

Many people contributed to the PT 2.1 specification. Major contributions were made by Dr. Brandon Wiley (Operator Foundation), Nick Mathewson (Tor), and Ben Schwartz (Jigsaw). Valuable feedback was provided by the attendees at the Pluggable Transport Implementers Meetings and the traffic-obf and tor-dev mailing lists. The PT 2.1 specification expands upon the "Pluggable Transport Specification (Version 1)" document authored by Yawning Angel (Tor). Inspiration for the PT 2.1 Go API was also inspired by the obfs4proxy implementation of the PT 1.0 specification in Go, also developed by Yawning Angel (Tor).

Authors' Addresses

Brandon Wiley
Operator Foundation

EMail: brandon@operatorfoundation.org
URI: https://operatorfoundation.org

David M. Oliver
Guardian Project

EMail: david@guardianproject.info
URI: https://guardianproject.info